

**UNIVERSIDAD DEL BÍO-BÍO**  
**FACULTAD DE CIENCIAS EMPRESARIALES**  
**DEPARTAMENTO DE SISTEMAS DE INFORMACIÓN**



**UNIVERSIDAD DEL BÍO-BÍO**

“Cluster de alto rendimiento para renderizado gráfico utilizando CPU y GPU”

**“High Performance Cluster for GPU and CPU Graphics Render”**

Proyecto de Software Aplicado presentado en conformidad a los requisitos para obtener el título de Ingeniero de Ejecución en Computación e Informática.

**Alumnos:**  
**Juan Lautaro Silva Ortiz**  
**Rodrigo Alexis Venegas Muñoz**

Profesor Guía:  
Sr. Juan Carlos Parra Márquez

Concepción, Enero de 2012

## Agradecimientos

Durante los últimos meses recibimos bastante apoyo para la realización de este documento, queremos dar gracias a los profesores organizadores del proyecto Alfa III Gaviota en la Universidad del Bío-Bío, así como al apoyo recibido por parte del Laboratorio de Estudios Urbanos y de todos los relacionados con la adquisición del hardware necesario para lograr realizar nuestro proyecto.

También quedemos dar gracias al profesor Dino Risso por la exhaustiva revisión de la primera versión del documento, además a nuestro compañero del Grupo de Experimentación Tridimensional, César Romero, quién participo activamente con nosotros en la aplicación del cluster en el renderizado, así como en la creación de los mismos modelos.

A nuestras familias, quienes nos apoyaron desde el inicio de nuestra etapa como estudiantes, damos gracias a la Universidad del Bío-Bío por brindar el apoyo y confianza depositado en nosotros como aprendices de su casa de estudios.

Esperamos que la realización y finalización de nuestro proyecto sea de gran utilidad para fomentar el desarrollo de futuras investigaciones.

Muchas Gracias.

Juan Lautaro Silva Ortiz.

Rodrigo Alexis Venegas Muñoz.

## **Resumen**

La investigación y puesta en marcha de este cluster de alto rendimiento fue realizada dentro del marco del proyecto Alfa III Gaviota cuyo objetivo es consolidar una red en el desarrollo de aplicaciones de realidad virtual. El proyecto contempla la implementación y adquisición de equipamiento necesario para su ejecución. Esta investigación describe las ventajas de las tecnologías de computo paralelo y sus beneficios para el área de realidad virtual gracias al poder de computo de una gran cantidad de CPU y GPU trabajando en conjunto, los cuales son ocupados por medio de un cluster de alto rendimiento como parte de las necesidades para la ejecución del proyecto y así poder generar imágenes de modelos 3D creados en blender.

Estas ventajas se demuestran mediante la realización de pruebas comparativas para tareas específicas de renderizado de imágenes sobre blender y sus alternativas en motores de renderizado, así como en las implementaciones propias de estos motores para mejorar sus capacidades. Además son realizadas pruebas de calculo y estrés computacional para medir la capacidad del cluster y su aplicación a otras áreas. Finalmente se analizan mediante gráficas los beneficios reales del procesamiento paralelo.

Palabras Clave: Cluster, Computación Paralela, Render, GPGPU, MOSIX.

## **Abstract**

Research and implementation of this high-performance cluster was conducted around the Alfa III Gaviota project. This project is designed to create a network for the development of virtual reality applications. The project includes the implementation and acquisition of equipment necessary for its implementation. This research describes the advantages of parallel computing technology and its benefits to the area of virtual reality through the power of computing a lot of CPU and GPU working together, which are occupied by a high-performance cluster as part of the requirements for the project execution and to be able to generate images of 3D models created in Blender.

These advantages are demonstrated by performing specific tasks benchmarks for rendering of images and its alternatives in blender render engines, as well as a personalised implementations of these engines to improve their capabilities. There was also conducted stress tests and computer calculations to measure the ability of the cluster and its application to other areas. Finally, were analyzed by graphing the real benefits of parallel processing.

Keywords: Cluster, Parallel Computing, Render, GPGPU, MOSIX.

# Índice de contenido

1 Introducción.....	8
2 Marco teórico.....	10
2.1 ¿Qué es un Cluster?.....	10
2.1.1 Historia.....	11
2.1.2 Beneficios de la Tecnología Cluster.....	13
2.1.3 Clasificación de los Clusters.....	14
2.1.4 Componentes de un Cluster.....	16
2.2 Procesamiento Paralelo.....	21
2.2.1 Multiprocesamiento Simétrico (SMP).....	22
2.2.1.1 Multiprocesamiento Simétrico de Hilos (SMT).....	23
2.2.2 Procesamiento Masivamente Paralelo (MPP).....	24
2.2.3 Procesamiento Paralelo Escalable (SPP).....	27
2.3 Procesamiento Paralelo en GPU.....	27
2.3.1 Historia de la GPU.....	29
2.3.2 Funcionamiento.....	29
2.3.3 Velocidad de Acceso.....	30
2.3.4 CUDA.....	30
2.3.5 OpenCL.....	31
3 Descripción del Problema.....	33
4 Origen del Proyecto.....	35
4.1 Objetivos.....	35
4.2 Metodología.....	36
5 Solución del Problema.....	37
5.1 Resumen del Problema.....	37
5.2 Factibilidad de la Solución.....	37
5.2.1 Estudio de Mercado.....	37
5.2.2 Estudio Técnico.....	39
5.2.3 Estudio Financiero.....	41
5.2.3.1 Horas hombre.....	42
5.2.3.2 Licencias de Software.....	42
5.3 Tipos de Configuraciones de Cluster.....	43
5.3.1 Tipo de Enfoque para el Procesamiento Paralelo.....	43
5.3.1.1 Multiprocesamiento Simétrico (SMP).....	43
5.3.1.2 Procesamiento Masivamente Paralelo (MPP).....	43
5.3.1.3 Procesamiento Paralelo Escalable (SPP).....	44
5.3.2 Optimizar Rendimiento, Distribución de Trabajos.....	44
5.3.3 Lenguajes de Programación Paralela.....	44
5.3.4 Middleware para la implementación del Cluster.....	46
5.3.4.1 CONDOR.....	47
5.3.4.2 MOSIX.....	48
5.3.4.3 OpenMOSIX.....	50
5.3.4.4 OpenSSI.....	51
6 Configuración del Cluster.....	52
6.1 Características Técnicas del Hardware.....	52
6.1.1 Red del Laboratorio:.....	53
6.1.2 Hardware del Head Node.....	54
6.1.3 Hardware de los Render Nodes.....	54
6.2 Características del Software.....	55
6.2.1 Sistema Operativo (Debian).....	55
6.2.2 Middleware (MOSIX).....	57

6.2.3 Blender y sus motores de renderizado.....	59
6.2.3.1 Blender.....	61
6.2.3.2 Cycles.....	61
6.2.3.3 NetRender.....	61
6.2.3.4 Blender Game Engine.....	61
7 Implementación de la Solución.....	62
7.1 Instalación de Debian.....	62
7.2 Instalación y Configuración del núcleo MOSIX.....	77
7.2.1 Requerimientos para la ejecución del Cluster MOSIX.....	77
7.2.2 Descarga de Paquetes.....	78
7.2.3 Configuración del Núcleo.....	79
7.2.4 Compilación e Instalación.....	80
7.3 Configuración MOSIX como Middleware de Administración del Cluster.....	82
7.3.1 Descripción de Archivos de MOSIX.....	83
7.3.2 Administración básica de MOSIX.....	87
7.4 Instalación del soporte CUDA y OpenCL.....	88
7.5 Instalación de MOSIX-VCL.....	91
7.5.1 Requerimientos MOSIX-VCL.....	92
7.5.2 Instalando MOSIX-VCL.....	93
7.5.3 Ejecución de MOSIX-VCL en sistema Debian.....	94
7.6 Configuración adicional de los nodos.....	95
7.7 Instalación de monitor web - Ganglia .....	96
7.7.1 Instalación en el Server.....	96
7.7.2 Configuración de Apache2.....	97
7.7.3 Instalación en los Nodos.....	97
8 Aplicación de Renderizado (BlenderBG).....	99
9 Benchmarks.....	101
9.1 Pruebas Prácticas.....	103
9.1.1 Render CPU – Blender.....	104
9.1.2 Render CPU – Blender Background.....	104
9.1.3 Render GPU – Cycles.....	105
9.1.4 Render GPU – Cycles Background.....	106
9.1.5 Resultados.....	107
9.1.6 Conclusiones pruebas prácticas.....	108
9.2 Pruebas de rendimiento.....	109
9.2.1 DistKeyGen.....	110
9.2.2 AWK.....	111
9.2.3 Forkit.....	112
9.3 Conclusiones pruebas rendimiento.....	114
10 Oportunidades Futuras.....	115
11 Conclusiones.....	116
APENDICE I - Herramientas de MOSIX.....	119
APENDICE II - Problemas Presentados Durante la Realización del Proyecto.....	126
APENDICE III – Código de aplicación Blender BG.....	127
APENDICE IV - Modificaciones VCL.....	130
APENDICE V – Script de Automatización Nodo.....	144
APENDICE VI – Pruebas de Rendimiento.....	148
APENDICE VII – Diagrama del proyecto Alfa III Gaviota.....	154
Glosario.....	155
Bibliografía.....	156
Literatura.....	156
Web.....	157

# Índice de ilustraciones

Ilustración 1: Modelo SMP.....	22
Ilustración 2: Procesamiento no escalable.....	23
Ilustración 3: Modelo Mpp.....	24
Ilustración 4: Procesamiento escalable.....	25
Ilustración 5: MPP - Distribución de memoria.....	25
Ilustración 6: Modelo SPP.....	27
Ilustración 7: Procesadores CPU y GPU.....	28
Ilustración 8: CUDA - Diseñado para varios lenguajes y APIs.....	31
Ilustración 9: Laboratorio Alfa III Gaviota - Universidad del Bío-Bío.....	52
Ilustración 10: Modelado 3D en blender.....	60
Ilustración 11: Debian - Selección de idioma.....	63
Ilustración 12: Debian - Selección de ubicación.....	64
Ilustración 13: Debian - Teclado.....	65
Ilustración 14: Debian - Nombre de la maquina.....	66
Ilustración 15: Debian - Nombre del dominio.....	67
Ilustración 16: Debian - Cuentas de usuario.....	68
Ilustración 17: Debian - contraseña.....	69
Ilustración 18: Debian - Zona horaria.....	70
Ilustración 19: Debian - Particionado.....	71
Ilustración 20: Debian - Instalación de paquetes.....	72
Ilustración 21: Debian - País de los repositorios.....	73
Ilustración 22: Debian - Replica a instalar.....	73
Ilustración 23: Debian - Selección de programas.....	74
Ilustración 24: Debian - Instalar GRUB.....	75
Ilustración 25: Debian - Instalación completa.....	76
Ilustración 26: Compilación de kernel mosix.....	80
Ilustración 27: mosix.install.....	82
Ilustración 28: VCLconf.....	93
Ilustración 29: Monitor web Ganglia.....	98
Ilustración 30: Algoritmo de ejecución de procesos de renderizado.....	99
Ilustración 31: Resumen Pruebas de render.....	107
Ilustración 32: mon.....	109
Ilustración 33: DistKeyGen.....	110
Ilustración 34: AWK.....	111
Ilustración 35: Forkit.....	113

## Índice de tablas

Tabla 1: Carta Gantt.....	40
Tabla 2: Estudio Financiero - Horas Hombre.....	42
Tabla 3: Estudio Financiero - Licencias de Software.....	42
Tabla 4: Pruebas Prácticas.....	101
Tabla 5: Pruebas prácticas por modelo.....	102
Tabla 6: Pruebas de rendimiento.....	102
Tabla 7: Render CPU - Blender.....	104
Tabla 8: Render CPU - Blender Background.....	105
Tabla 9: Render GPU - Cycles.....	105
Tabla 10: Render GPU - Cycles Background.....	106
Tabla 11: DistKeyGen.....	110
Tabla 12: AWK.....	111
Tabla 13: Forkit.....	112

# 1 Introducción

La velocidad de cálculo de los procesadores actuales ha ido aumentando exponencialmente debido a los avances en la tecnología utilizada en su proceso de manufactura, permitiendo cada vez crear dispositivos de menor tamaño, menor consumo y gran poder de cálculo.

Sin embargo, esto no siempre es suficiente, dado que las necesidades de procesamiento para tareas específicas requieren un mayor poder de cálculo, es por esto que surgió la necesidad de crear un sistema de procesamiento superior al mejor procesador del mercado. En base a esta necesidad surge la idea de la escalabilidad mediante la utilización más de un procesador para realizar una tarea específica. En los años 60' Gene Amdahl planteo mediante un modelo matemático el aceleramiento que se puede esperar paralelizando cualquier otra serie de tareas realizadas en una arquitectura paralela.

Las primeras implementaciones de arquitecturas paralelas surgieron combinando la capacidad de procesamiento de más de un procesador a la vez en la misma placa madre, esto además de aumentar la capacidad de procesamiento consume más electricidad y produce más calor, por lo que fue necesario crear una nueva tecnología más económica, los procesadores multi-núcleos. Aún así tales ventajas de manufactura y reducción de tamaño de los chips no fueron suficientes para procesos que requerían de alto rendimiento, alta disponibilidad y alta eficiencia.

Dado este escenario se plantea utilizar una solución distribuyendo la carga de trabajo entre varios equipos en conjunto, a los cuales se les llama granja de computadores o cluster, donde cada equipo es un nodo de ella. Esta distribución de la carga de trabajo aumenta la eficiencia y productividad. El resultado obtenido para un proceso ejecutado en un cluster es equivalente al obtenido en un solo procesador, pero mermando los tiempos de ejecución.

Normalmente se ha utilizado el CPU como la unidad central de procesamiento para todo tipo de tareas. Paralelamente ha evolucionado la industria de los chips gráficos creando GPUs de alto rendimiento capaces de realizar procesos para los que no fueron diseñadas originalmente, superando en algunos casos a los CPU en la velocidad de procesamiento. Este tipo de hardware incluso cuenta con cientos de unidades de proceso en el mismo chip, mientras que los CPU no suelen superar los 8 núcleos en el mejor de los casos.

También es posible realizar una combinación de las tecnologías basadas en cluster de CPU y GPU's obteniendo un notorio y mejor rendimiento en las aplicaciones dedicadas a hacer uso intensivo de recursos. La creación de un cluster que utilice heterogéneamente múltiples tecnologías de procesamiento nos permitirá en el transcurso de esta investigación realizar las tareas dedicadas al renderizado de modelos tridimensionales en un menor tiempo sin perder calidad en la animación renderizada.

## 2 Marco teórico

### 2.1 ¿Qué es un Cluster?

El término cluster se aplica a los conjuntos o granjas de computadoras construidas mediante la utilización de componentes de hardware en común y que se comportan como si fuesen una única computadora.

Hoy en día desempeñan un papel importante en la solución de problemas de las ciencias, las ingenierías y del comercio.

La tecnología de clusters ha evolucionado en apoyo de actividades que van desde aplicaciones de supercómputo y software especializados, servidores web y comercio electrónico, hasta bases de datos de alto rendimiento, entre otros usos.

El cómputo con clusters surge como resultado de la convergencia de varias tendencias actuales que incluyen la disponibilidad de microprocesadores económicos de alto rendimiento y redes de alta velocidad, el desarrollo de herramientas de software para cómputo distribuido de alto rendimiento, así como la creciente necesidad de potencia computacional para aplicaciones que la requieran.

Simplemente, un cluster es un grupo de múltiples computadores unidos mediante una red de alta velocidad, de tal forma que el conjunto es visto como una única máquina, más potente que los comunes de escritorio.

Los clusters son usualmente empleados para mejorar el rendimiento y/o la disponibilidad por encima de la que es provista por un solo computador, típicamente siendo más económico que computadores individuales de rapidez y disponibilidad comparables.

De un cluster se espera que presente combinaciones de los siguientes servicios:

- Alto rendimiento
- Alta disponibilidad
- Balanceo de carga
- Escalabilidad

La construcción de los nodos del cluster es más fácil y económica debido a su flexibilidad: pueden tener todos la misma configuración de hardware y sistema operativo, a esto se le llama cluster homogéneo, diferente rendimiento pero con arquitecturas y sistemas operativos similares, cluster (semi-homogéneo), o tener diferente hardware y sistema operativo (cluster heterogéneo), lo que hace más fácil y económica su construcción.

Para que un cluster funcione como tal, no basta solo con conectar entre sí los nodos, sino que es necesario proveer un sistema de manejo del cluster, el cual se encargue de interactuar con el usuario y los procesos que corren en él para optimizar el funcionamiento.

### **2.1.1 Historia**

El origen del término y del uso de este tipo de tecnología es desconocido pero se puede considerar que comenzó a finales de los años 50 y principios de los años 60.

La base formal de la ingeniería informática de la categoría como un medio de hacer trabajos paralelos de cualquier tipo fue posiblemente inventado por Gene Amdahl de IBM, que en 1967 publicó lo que ha llegado a ser considerado como el artículo inicial de procesamiento paralelo: la Ley de Amdahl que describe matemáticamente el aceleramiento que se puede esperar paralelizando cualquier otra serie de tareas realizadas en una arquitectura paralela. Este artículo define las bases para la ingeniería de la computación basada tanto en multiprocesador y computación cluster, en donde el principal papel diferenciador es si las comunicaciones interprocesador cuentan con el apoyo "dentro" de la computadora, por ejemplo, en una configuración personalizada para el bus o la red de las comunicaciones internas o "fuera" del ordenador en una red "commodity".

En consecuencia, la historia de los primeros grupos de computadoras está más o menos directamente ligada a la historia de principios de las redes, como una de las principales motivaciones para el desarrollo de una red para enlazar los recursos de computación, de hecho la creación de un cluster de computadoras. Las redes de conmutación de paquetes fueron conceptualmente inventados por la corporación RAND en 1962.

Utilizando el concepto de una red de conmutación de paquetes, el proyecto ARPANET logró crear en 1969 lo que fue posiblemente la primera red básica de computadoras basadas en cluster.

El proyecto ARPANET creció y se convirtió en lo que es ahora Internet que se puede considerar como "la madre de todos los clusters" (como la unión de casi todos los recursos de cómputo, incluidos los clusters, que pasarían a ser conectados).

También estableció el paradigma de uso de computadoras clusters en el mundo de hoy: el uso de las redes de conmutación de paquetes para realizar las comunicaciones entre procesadores localizados en marcos de otro modo desconectados.

El proceso de construcción de PC's fue avanzando y se investigó paralelamente el desarrollo de las redes y el sistema operativo Unix desde principios de la década de los años 70, ejemplos de esto fueron el desarrollo de TCP/IP y la creación de la Xerox PARC (Palo Alto Research Company). Estos avances fueron formalizados como proyectos para la creación de protocolos basados en la red de comunicaciones.

Sin embargo, no fue sino hasta alrededor de 1983 en que fueron definidos los protocolos y herramientas para el trabajo remoto que facilitasen la distribución y el uso compartido de archivos (en gran medida dentro del contexto de BSD Unix, e implementados por Sun Microsystems) y, por lo tanto se llega a disponer comercialmente de este tipo de herramientas, junto con una compartición del sistema de ficheros.

El primer producto comercial de tipo cluster fue ARCnet, desarrollada en 1977 por Datapoint pero no obtuvo un éxito comercial y los clusters no consiguieron tener éxito hasta que en 1984 VAXcluster produjeron el sistema operativo VAX/VMS.

El ARCnet y VAXcluster no sólo son productos que apoyan la computación paralela, pero también comparten los sistemas de archivos y dispositivos periféricos.

La idea era proporcionar las ventajas del procesamiento paralelo, al tiempo que se mantiene la fiabilidad de los datos y el carácter singular. VAXcluster, VMScluster está todavía disponible en los sistemas de HP OpenVMS corriendo en sistemas Itanium y Alpha. Otros dos principios comerciales de clusters notables fueron el Tandem Himalaya (alrededor de 1994 con productos de alta disponibilidad) y el IBM S/390 Parallel Sysplex (también alrededor de 1994, principalmente para el uso de la empresa).

La historia de los clusters de computadoras estaría incompleta sin señalar el papel fundamental desempeñado por el desarrollo del software de Parallel Virtual Machine (PVM). Este software de

código abierto basado en comunicaciones TCP/IP permitió la creación de un supercomputador virtual - un cluster HPC- realizada desde cualquiera de los sistemas conectados TCP/IP.

De forma libre los clusters heterogéneos han constituido la cima de este modelo logrando aumentar rápidamente en FLOPS globalmente y superando con creces la disponibilidad incluso de los más caros supercomputadores.

PVM y el empleo de PC y redes de bajo costo llevó, en 1993, a un proyecto de la NASA para construir súper computadoras de clusters.

En 1995, la invención de la Beowulf -un estilo de cluster- una granja de computación diseñado según un producto básico de la red con el objetivo específico de "ser un súper computador" capaz de realizar cálculos complejos paralelizados en HPC. Esto estimuló el desarrollo independiente de la computación en forma de granjas como una entidad, a pesar de que el estilo giraba en torno al del sistema operativo Unix y el Arpanet.

## **2.1.2      Beneficios de la Tecnología Cluster**

Las aplicaciones paralelas escalables requieren: buen rendimiento, baja latencia, comunicaciones que dispongan de gran ancho de banda, redes escalables y acceso rápido a archivos. Un cluster puede satisfacer estos requerimientos usando los recursos que tiene asociados a él.

Los clusters ofrecen las siguientes características a un costo relativamente bajo:

- Alto rendimiento
- Alta disponibilidad
- Alta eficiencia
- Escalabilidad

La tecnología cluster permite a las organizaciones incrementar su capacidad de procesamiento usando tecnología estándar, tanto en componentes de hardware como de software que pueden adquirirse a un costo relativamente bajo.

## 2.1.3 Clasificación de los Clusters

El término cluster tiene diferentes connotaciones para diferentes grupos de personas. Los tipos de clusters, establecidos de acuerdo con el uso que se suministre y los servicios que ofrecen, determinan el significado del término para el grupo que lo utiliza. Los clusters pueden clasificarse según sus características: se pueden tener clusters de alto rendimiento (HPCC – High Performance Computing Clusters), clusters de alta disponibilidad (HA – High Availability) o clusters de alta eficiencia (HT – High Throughput).

**Alto Rendimiento:**

Un cluster de alto rendimiento es un conjunto de computadores que está diseñado para dar altas prestaciones en cuanto a capacidad de cálculo, ejecutan tareas que requieren de gran capacidad computacional, grandes cantidades de memoria, o ambos a la vez. El llevar a cabo estas tareas puede comprometer los recursos del cluster por largos periodos de tiempo. Los motivos para utilizar un cluster de alto rendimiento son:

- el tamaño del problema por resolver
- el precio de la máquina necesaria para resolverlo.

Por medio de un cluster se pueden conseguir capacidades de cálculo superiores a las de un computador de costo mayor que el costo conjunto de los ordenadores del cluster.

Para garantizar esta capacidad de cálculo, los problemas necesitan ser paralelizados, ya que el método con el que los clusters agilizan el procesamiento es dividir el problema en problemas más pequeños y calcularlos en los nodos, por lo tanto, si el problema no cumple con esta característica, no puede utilizarse el cluster para su cálculo.

Para resolver un problema paralelizado se debe hacer uso de bibliotecas especiales.

**Alta Disponibilidad:**

Un cluster de alta disponibilidad es un conjunto de dos o más máquinas que se caracterizan por mantener una serie de servicios compartidos y por estar constantemente monitorizándose entre sí. Su objetivo de diseño es el de proveer disponibilidad y confiabilidad. Estos clusters tratan de brindar la

máxima disponibilidad de los servicios que ofrecen. La confiabilidad se provee mediante software que detecta fallos y permite recuperarse frente a los mismos, mientras que en hardware se evita tener un único punto de fallos. Podemos dividirlo en dos clases:

Alta disponibilidad de infraestructura: Si se produce un fallo de hardware en alguna de las máquinas del cluster, el software de alta disponibilidad es capaz de arrancar automáticamente los servicios en cualquiera de las otras máquinas y cuando la máquina que ha fallado se recupera, los servicios son nuevamente migrados a la máquina original. Esta capacidad de recuperación automática de servicios nos garantiza la alta disponibilidad de los servicios ofrecidos por el cluster, minimizando así la percepción del fallo por parte de los usuarios.

Alta disponibilidad de aplicación: Si se produce un fallo del hardware o de las aplicaciones de alguna de las máquinas del cluster, el software de alta disponibilidad es capaz de arrancar automáticamente los servicios que han fallado en cualquiera de las otras máquinas y cuando la máquina que ha fallado se recupera, los servicios son nuevamente migrados a la máquina original. Esta capacidad de recuperación automática de servicios nos garantiza la integridad de la información, ya que no hay pérdida de datos, y además evita molestias a los usuarios, haciendo el inconveniente transparente para ellos.

No hay que confundir un cluster de alta disponibilidad con un cluster de alto rendimiento. El segundo es una configuración de equipos diseñado para proporcionar capacidades de cálculo mucho mayores que la que proporcionan los equipos individuales, mientras que el primer tipo de cluster está diseñado para garantizar el funcionamiento ininterrumpido de ciertas aplicaciones.

### **Alta Eficiencia:**

Son clusters cuyo objetivo de diseño es el ejecutar la mayor cantidad de tareas en el menor tiempo posible. Existe independencia de datos entre las tareas individuales. El retardo entre los nodos del cluster no es considerado un gran problema.

Los clusters pueden también clasificar como Clusters Comerciales (Alta disponibilidad, Alta eficiencia) y Clusters Científicos (Alto rendimiento). A pesar de las discrepancias a nivel de requerimientos de las aplicaciones, muchas de las características de las arquitecturas de hardware y software, que están por debajo de las aplicaciones en todos estos clusters, son las mismas. Más aún, un cluster de determinado tipo, puede también presentar características de los otros.

## 2.1.4 Componentes de un Cluster

En general, un cluster necesita de varios componentes de software y hardware para poder funcionar. Entre ellos:

- Nodos
- Almacenamiento
- Sistemas Operativos
- Conexiones de Red
- Middleware
- Protocolos de Comunicación y servicios
- Aplicaciones
- Ambientes de Programación Paralela

### Nodos

Pueden ser simples computadores, sistemas multi-procesador o estaciones de trabajo (workstations). En informática, de forma muy general, un nodo es un punto de intersección o unión de varios elementos que confluyen en el mismo lugar. Ahora bien, la palabra nodo puede referirse a conceptos diferentes, en nuestro caso nos referimos al nodo según el concepto de redes.

- En redes de computadoras cada una de las máquinas es un nodo, y si la red es Internet, cada servidor constituye también un nodo.

El cluster puede estar conformado por nodos dedicados o por nodos no dedicados.

En un cluster con nodos dedicados, los nodos no disponen de teclado, ratón ni monitor y su uso está exclusivamente dedicado a realizar tareas relacionadas con el cluster. Mientras que, en un cluster con nodos no dedicados, los nodos disponen de teclado, ratón y monitor y su uso no está exclusivamente dedicado a realizar tareas relacionadas con el cluster, el cluster hace uso de los ciclos de reloj que el usuario del computador no está utilizando para realizar sus tareas.

Cabe aclarar que a la hora de diseñar un cluster conviene que los nodos tengan características similares, es decir, deben guardar cierta similitud de arquitectura y sistemas operativos, ya que si se

conforma un cluster con nodos totalmente heterogéneos (existe una diferencia grande entre capacidad de procesadores, memoria, disco duro) será ineficiente debido a que el Middleware delegará o asignará todos los procesos al nodo de mayor capacidad de cómputo y solo distribuirá cuando este se encuentre saturado de procesos, por eso es recomendable construir un grupo de computadores lo más similares posible.

## Almacenamiento

El almacenamiento puede consistir en una NAS, una SAN, o almacenamiento interno en el servidor. El protocolo más comúnmente utilizado es NFS (Network File System), sistema de ficheros compartido entre servidor y los nodos. Sin embargo existen sistemas de ficheros específicos para clusters como Lustre (CFS) y PVFS2.

Tecnologías en el soporte del almacenamiento en discos duros:

- IDE o ATA: velocidades de 33, 66, 100, 133 y 166 MB/s
- SATA: velocidades de 150, 300 y 600 MB/s
- SCSI: velocidades de 160, 320, 640 MB/s. Proporciona altos rendimientos.
- SAS: aúna SATA-II y SCSI. Velocidades de 300 y 600 MB/s
- Unidades de cinta (DLT) son utilizadas para copias de seguridad por su bajo coste.

NAS (Network Attached Storage) es un dispositivo específico dedicado al almacenamiento a través de red (normalmente TCP/IP) que hace uso de un sistema operativo optimizado para dar acceso a través de protocolos CIFS, NFS, FTP o TFTP.

Por su parte, DAS (Direct Attached Storage) consiste en conectar unidades externas de almacenamiento SCSI o una SAN (Storage Area Network) a través de un canal de fibra. Estas conexiones son dedicadas.

Mientras NAS permite compartir el almacenamiento, utilizar la red, y tiene una gestión más sencilla, DAS proporciona mayor rendimiento y mayor fiabilidad al no compartir el recurso.

## Sistema Operativo

Un sistema operativo debe ser multiproceso y multiusuario. Otras características deseables son la facilidad de uso y acceso. Un sistema operativo es un programa o conjunto de programas de

computadora destinado a permitir una gestión eficaz de sus recursos.

Algunos de los que podemos encontrar son:

- GNU/Linux
  - ABC GNU/Linux2
  - OpenMosaic
  - Rocks3
  - Kerrighed
  - Cóndor
  - Sun Grid Engine
- Unix
  - Solaris
  - HP-UX
  - Aix
- Windows
  - NT
  - 2000 Server
  - 2003 Server
  - 2008 Server
- Mac OS X
  - Xgrid
- Solaris
- FreeBSD

## Conexiones de Red

Los nodos de un cluster pueden conectarse mediante una simple red Ethernet con placas comunes (adaptadores de red o NICs), o utilizarse tecnologías especiales de alta velocidad como Fast Ethernet, Gigabit Ethernet, Myrinet, InfiniBand, SCI, etc.

- Ethernet
  - Son las redes más utilizadas en la actualidad, debido a su relativo bajo coste. No obstante, su tecnología limita el tamaño de paquete, realizan excesivas comprobaciones de error y sus protocolos no son eficientes, y sus velocidades de

transmisión pueden limitar el rendimiento de los clusters. Para aplicaciones con paralelismo de grano grueso puede suponer una solución acertada.

- La opción más utilizada en la actualidad es Gigabit Ethernet (1 Gbit/s), siendo emergente la solución 10 Gigabit Ethernet (10 Gbit/s). La latencia de estas tecnologías está en torno a los 30-100  $\mu$ s, dependiendo del protocolo de comunicación empleado.
- En todo caso, es la red de administración por excelencia, así que aunque no sea la solución de red de altas prestaciones para las comunicaciones, es la red dedicada a las tareas administrativas.
- Myrinet (Myrinet 2000 y Myri-10G)
  - Su latencia es de 1,3/10  $\mu$ s, y su ancho de Banda de 2/10Gbps, respectivamente para Myrinet 2000 y Myri-10G.
  - Es la red de baja latencia más utilizada en la actualidad, tanto en clusters como en MPPs estando presente en más de la mitad de los sistemas del top500. Tiene dos bibliotecas de comunicación a bajo nivel (GM y MX). Sobre estas bibliotecas están implementadas MPICH-GM, MPICH-MX, Sockets-GM y Sockets MX, para aprovechar las excelentes características de Myrinet. Existen también emulaciones IP sobre TCP/IP, IPoGM e IpMX.
- InfiniBand
  - Es una red surgida de un estándar desarrollado específicamente para realizar la comunicación en clusters. Una de sus mayores ventajas es que mediante la agregación de canales (x1, x4 y x12) permite obtener anchos de banda muy elevados. La conexión básica es de 2Gbps efectivos y con 'quad connection' x12 alcanza los 96Gbps. No obstante, los startups no son muy altos, se sitúan en torno a los 10  $\mu$ s.
  - Define una conexión entre un nodo de computación y un nodo de I/O. La conexión va desde un Host Channel Adapter (HCA) hasta un Target Channel Adapter (TCA). Se está usando principalmente para acceder a arrays de discos SAS.
- SCI (Scalable Coherent Interface) IEEE standar 1596-1992

- Su latencia teórica es de 1.43  $\mu$ s y su ancho de banda de 5333 Mbps bidireccional. Al poder configurarse con topologías de anillo (1D), toro (2D) e hipercubo (3D) sin necesidad de switch, se tiene una red adecuada para clusters de pequeño y mediano tamaño.
- Al ser una red de extremadamente baja latencia, presenta ventajas frente a Myrinet en clusters de pequeño tamaño al tener una topología punto a punto y no ser necesaria la adquisición de un commutador. El software sobre SCI está menos desarrollado que sobre Myrinet, pero los rendimientos obtenidos son superiores, destacando SCI Sockets (que obtiene startups de 3 microsegundos) y ScaMPI, una biblioteca MPI de elevadas prestaciones.
- Además, a través del mecanismo de pre-loading (LD\_PRELOAD) se puede conseguir que todas las comunicaciones del sistema vayan a través de SCI-SOCKETS (transparencia para el usuario).

## Middleware

El middleware es un software que generalmente actúa entre el sistema operativo y las aplicaciones con la finalidad de proveer a un cluster lo siguiente:

- Una interfaz única de acceso al sistema, denominada SSI (Single System Image), la cual genera la sensación al usuario de que utiliza un único ordenador muy potente;
- Herramientas para la optimización y mantenimiento del sistema: migración de procesos, checkpoint-restart (congelar uno o varios procesos, mudarlos de servidor y continuar su funcionamiento en el nuevo host), balanceo de carga, tolerancia a fallos, etc.;
- Escalabilidad: debe poder detectar automáticamente nuevos servidores conectados al cluster para proceder a su utilización.

Existen diversos tipos de middleware, como por ejemplo: MOSIX, OpenMOSIX, Condor, OpenSSI, etc.

El middleware recibe los trabajos entrantes al cluster y los redistribuye de manera que el proceso se ejecute más rápido y el sistema no sufra sobrecargas en un servidor. Esto se realiza mediante políticas definidas en el sistema (automáticamente o por un administrador) que le indican dónde y cómo debe distribuir los procesos, por un sistema de monitorización, el cual controla la carga de cada CPU y la cantidad de procesos en él.

El middleware también debe poder migrar procesos entre servidores con distintas finalidades:

- Balancear la carga: si un servidor está muy cargado de procesos y otro está ocioso, pueden transferirse procesos a este último para liberar de carga al primero y optimizar el funcionamiento;
- Mantenimiento de servidores: si hay procesos corriendo en un servidor que necesita mantenimiento o una actualización, es posible migrar los procesos a otro servidor y proceder a desconectar del cluster al primero;
- Priorización de trabajos: en caso de tener varios procesos corriendo en el cluster, pero uno de ellos de mayor importancia que los demás, puede migrarse este proceso a los servidores que posean más o mejores recursos para acelerar su procesamiento.

## **2.2 Procesamiento Paralelo**

La velocidad de cálculo siempre ha sido el factor más considerado de los procesadores al momento de seleccionar uno para determinada tarea. Al pasar de los años se han mejorado las tecnologías de fabricación de los mismos y han mejorado su capacidad de cálculo exponencialmente. A pesar de esto, en los últimos años se ha comenzado a explotar la utilización de más de un procesador a la vez, esto ofrece una gran ventaja en cuanto a costos. Sin embargo, su principal beneficio, la escalabilidad (crecer hacia arquitecturas de mayor capacidad), puede ser difícil de alcanzar aún. Esto se debe a que conforme se añaden procesadores, las disputas por los recursos compartidos se intensifican.

Algunos diseños diferentes de procesamiento paralelo enfrentan este problema fundamental:

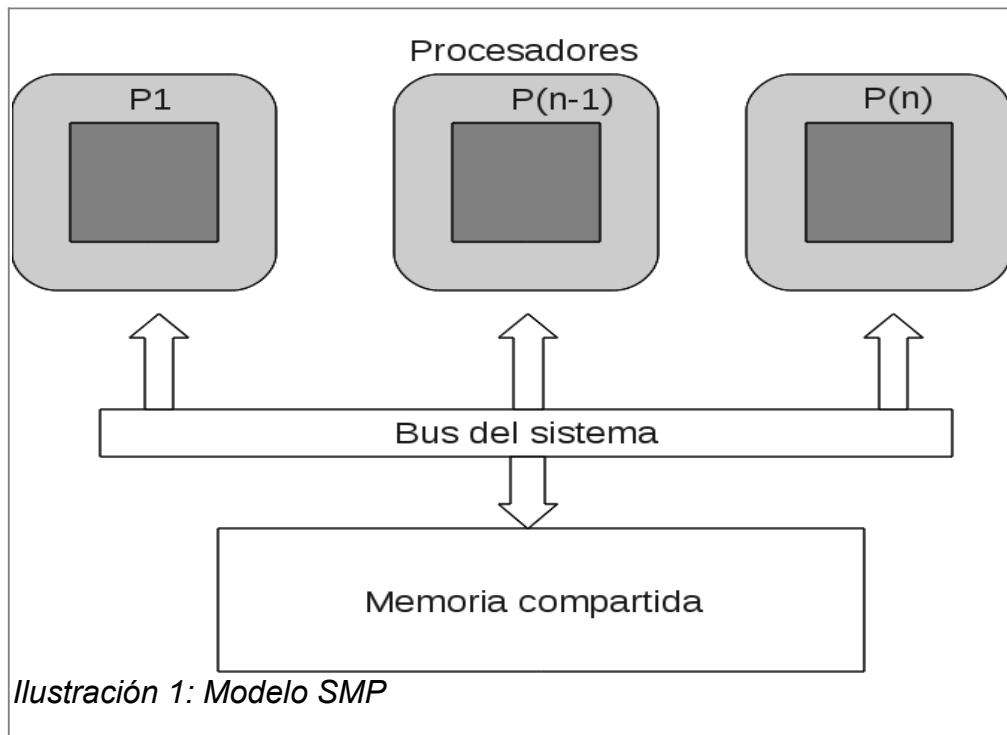
- Multiprocesamiento simétrico
- Procesamiento de múltiples hilos

- Procesamiento masivamente paralelo
- Procesamiento paralelo escalable

Cada diseño tiene sus propias ventajas y desventajas.

### 2.2.1 Multiprocesamiento Simétrico (SMP)

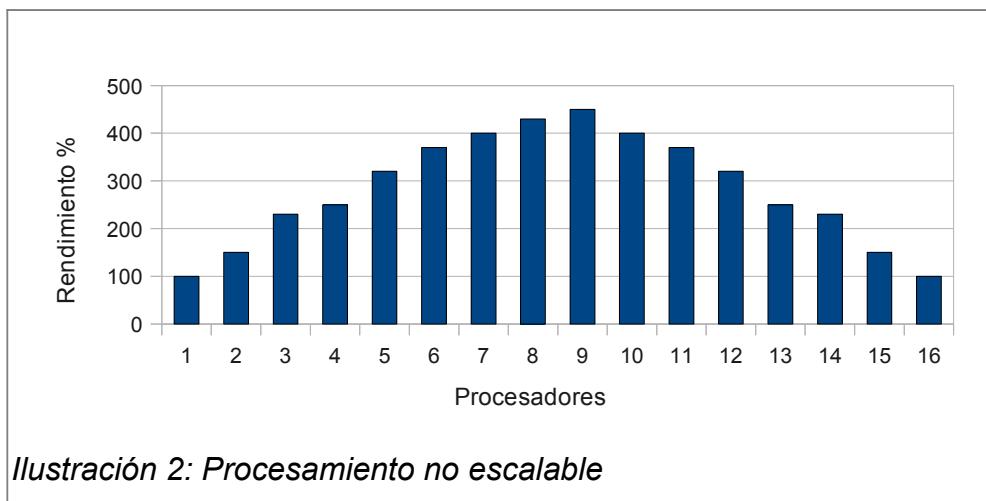
El Multiprocesamiento simétrico (symmetric multiprocessing/SMP) tiene un diseño simple pero aún así efectivo. En SMP, múltiples procesadores comparten la memoria RAM y el bus del sistema. Este diseño es también conocido como estrechamente acoplado (tightly coupled), o compartiendo todo (shared everything).



Debido a que SMP comparte globalmente la memoria RAM, tiene solamente un espacio de memoria, lo que simplifica tanto el sistema físico como la programación de aplicaciones. Este espacio de memoria único permite que un Sistema Operativo con Multiconexión (Multithreaded Operating System) distribuya las tareas entre varios procesadores, o permite que una aplicación obtenga la memoria que necesita para una simulación compleja. La memoria globalmente compartida también vuelve fácil la sincronización de los datos.

SMP es uno de los diseños de procesamiento paralelo más maduro. Apareció en los supercomputadores Cray X-MP y en sistemas similares hace década y media (en 1983).

Sin embargo, esta memoria global contribuye el problema más grande de SMP: conforme se añaden procesadores, el tráfico en el bus de memoria se satura. Al añadir memoria caché a cada procesador se puede reducir algo del tráfico en el bus, pero el bus generalmente se convierte en un cuello de botella al manejarse alrededor de ocho o más procesadores. SMP es considerada una tecnología no escalable.



### 2.2.1.1 **Multiprocesamiento Simétrico de Hilos (SMT)**

En la actualidad, la mayoría de los procesadores poseen soporte para procesar múltiples hilos (Multithread) en un único CPU. En el multiprocesamiento simétrico de hilos (Symmetric MultiThreading SMT), un solo procesador aparece ser dos o más procesadores virtuales. Estos CPU's virtuales comparten los recursos del núcleo del procesador, lo que incluye el motor de ejecución y el cache del procesador.

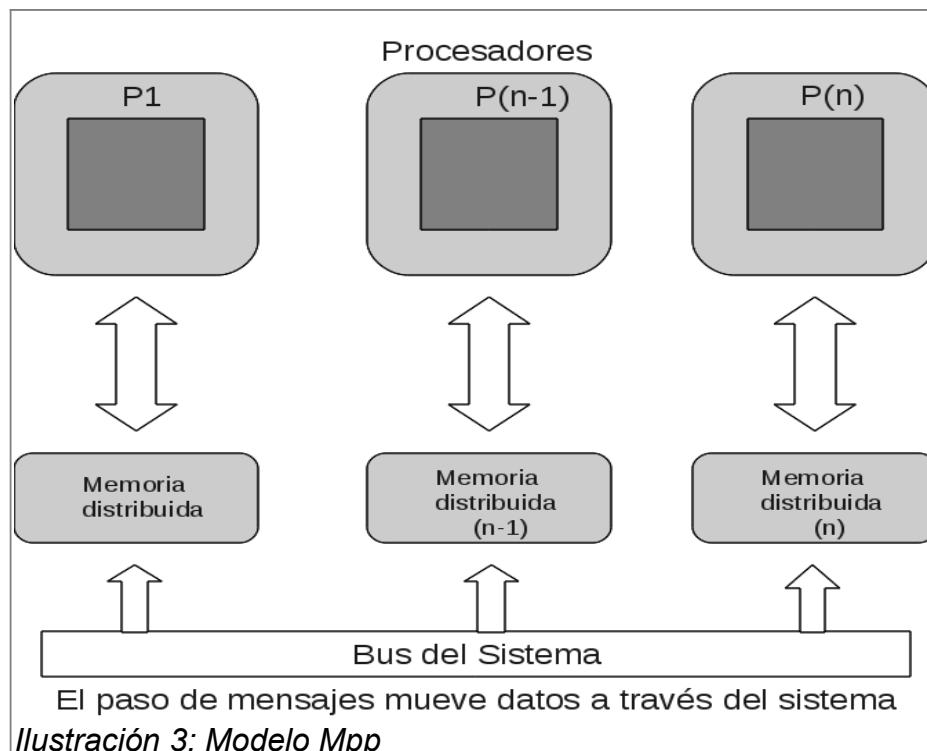
El multiprocesamiento simétrico de hilos permite a dos o más tareas ser ejecutadas simultáneamente en el procesador, lo que resulta en un incremento en el rendimiento del procesador. Pero esto tiene implicaciones de sincronización.

Además se debe tener en cuenta que la existencia de sistemas con dos (o más) CPU's físicos, los cuales pueden tener soporte para dos hilos (o más) cada uno, completando un total de cuatro CPU's

virtuales. El rendimiento de este tipo de sistemas será mejor si el sincronizador está consciente de que las cuatro CPU's virtuales no son lo mismo. Si sólo hay dos tareas en el sistema, el sincronizador debería ubicarlas ambas en dos CPU's diferentes. También, cuando una tarea que está siendo ejecutada en un CPU virtual esta lista para ejecutarse, debiese ser ubicada en el mismo CPU físico ya que la cache ya está en uso.

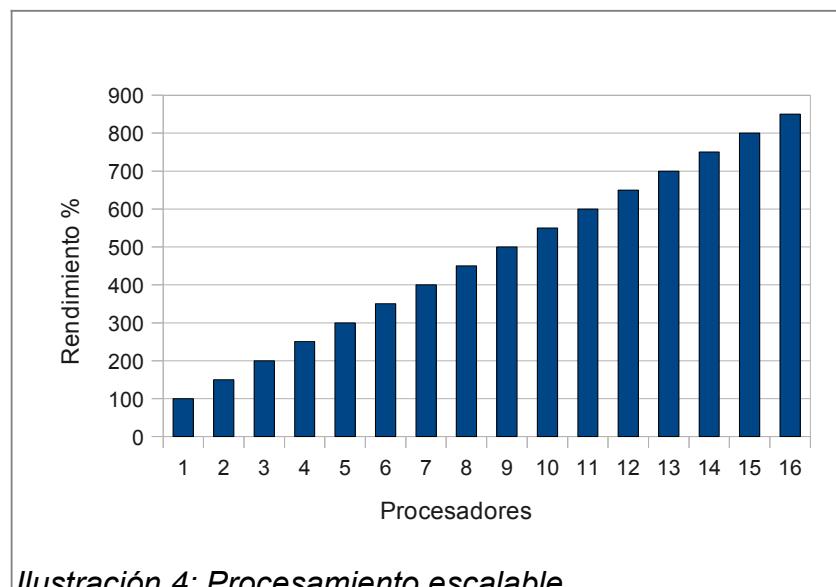
## 2.2.2 Procesamiento Masivamente Paralelo (MPP)

El Procesamiento masivamente paralelo (Massively Parallel Processing/MPP) es otro diseño de procesamiento paralelo. Para evitar los cuellos de botella en el bus de memoria, MPP no utiliza memoria compartida. En su lugar, distribuye la memoria RAM entre los procesadores de modo que se asemeja a una red (cada procesador con su memoria distribuida asociada es similar a un computador dentro de una red de procesamiento distribuido). Debido a la distribución dispersa de los recursos RAM, esta arquitectura es también conocida como dispersamente acoplada (loosely coupled), o compartiendo nada (shared nothing).

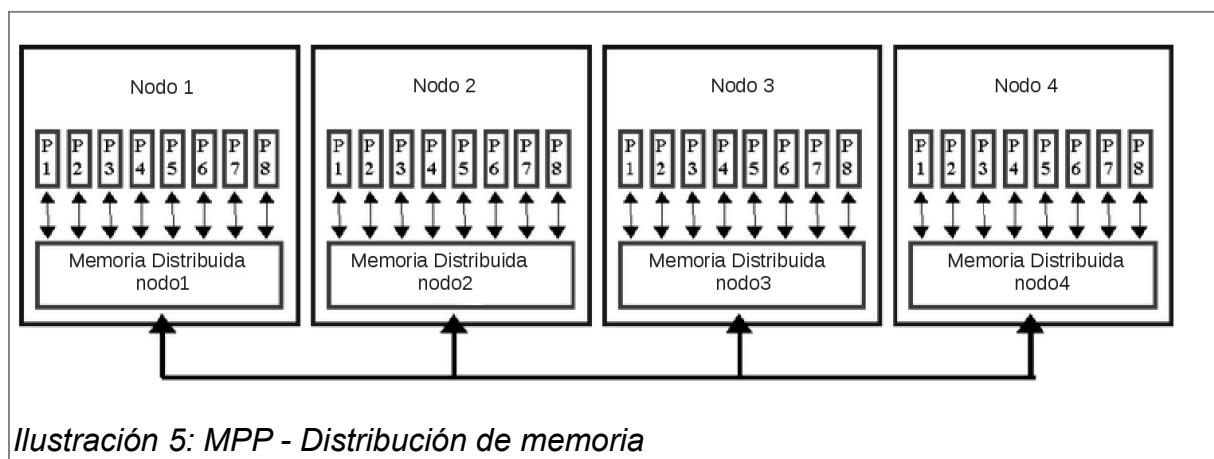


Para tener acceso a la memoria fuera de su propia RAM, los procesadores utilizan un esquema de paso de mensajes análogo a los paquetes de datos en redes. Este sistema reduce el tráfico del bus,

debido a que cada sección de memoria observa únicamente aquellos accesos que le están destinados, en lugar de observar todos los accesos, como ocurre en un sistema SMP. Únicamente cuando un procesador no dispone de la memoria RAM suficiente, utiliza la memoria RAM sobrante de los otros procesadores. Esto permite sistemas MPP de gran tamaño con cientos y aún miles de procesadores. MPP es una tecnología escalable.



El RS/6000 Scalable Powerparallel System de IBM (SP2) es un ejemplo de sistema MPP, que presenta una ligera variante respecto al esquema genérico anteriormente planteado. Los procesadores del RS/6000 se agrupan en nodos de 8 procesadores, los que utilizan una única memoria compartida (tecnología SMP). A su vez estos nodos se agrupan entre sí utilizando memoria distribuida para cada nodo (tecnología MPP). De este modo se consigue un diseño más económico y con mayor capacidad de crecimiento.

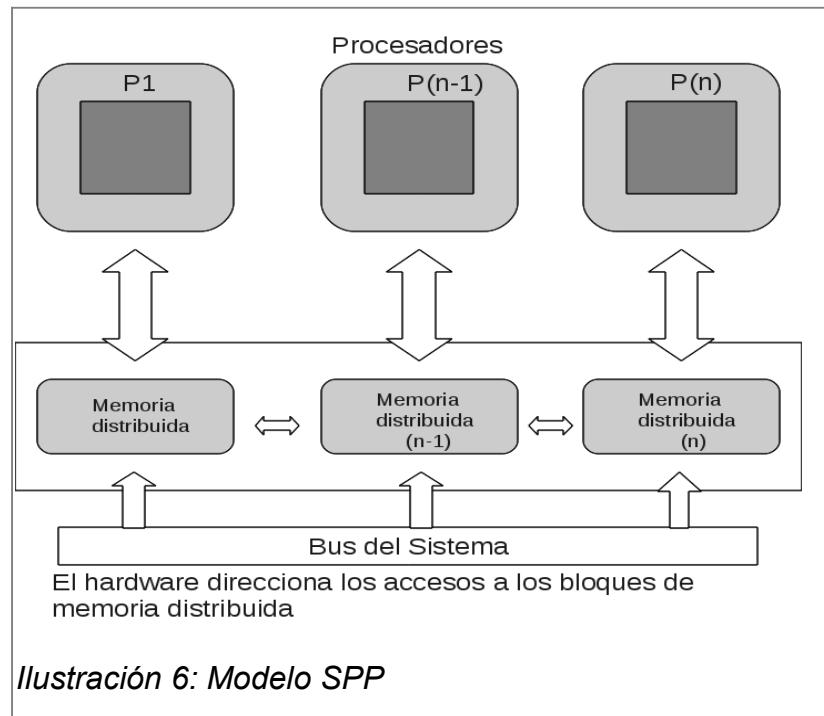


La parte negativa de MPP es que la programación se vuelve difícil, debido a que la memoria se rompe en pequeños espacios separados. Sin la existencia de un espacio de memoria globalmente compartido, correr (y escribir) una aplicación que requiere una gran cantidad de RAM (comparada con la memoria local), puede ser difícil. La sincronización de datos entre tareas ampliamente distribuidas también se vuelve difícil, particularmente si un mensaje debe pasar por muchas fases hasta alcanzar la memoria del procesador destino.

Escribir una aplicación MPP también requiere estar al tanto de la organización de la memoria manejada por el programa. Donde sea necesario, se requieren insertar comandos de paso de mensajes dentro del código del programa. Además de complicar el diseño del programa, tales comandos pueden crear dependencias de hardware en las aplicaciones. Sin embargo, la mayor parte de proveedores de hardware mantienen la portabilidad de las aplicaciones adoptando, sea un mecanismo de dominio público para paso de mensajes conocido como Máquina virtual paralela (Parallel Virtual Machine/PVM), o un estándar en fase de desarrollo llamado Interfaz de Paso de Mensajes (Message Passing Interface/MPI), para implementar el mecanismo de paso de mensajes.

### 2.2.3 Procesamiento Paralelo Escalable (SPP)

¿Cómo superar las dificultades de SMP y MPP? La última arquitectura paralela, el Procesamiento paralelo escalable (Scalable Parallel Processing/SPP), es un híbrido de SMP y MPP, que utiliza una memoria jerárquica de dos niveles para alcanzar la escalabilidad. La primera capa de memoria consiste de un nodo que es esencialmente un sistema SMP completo, con múltiples procesadores y su memoria globalmente compartida.



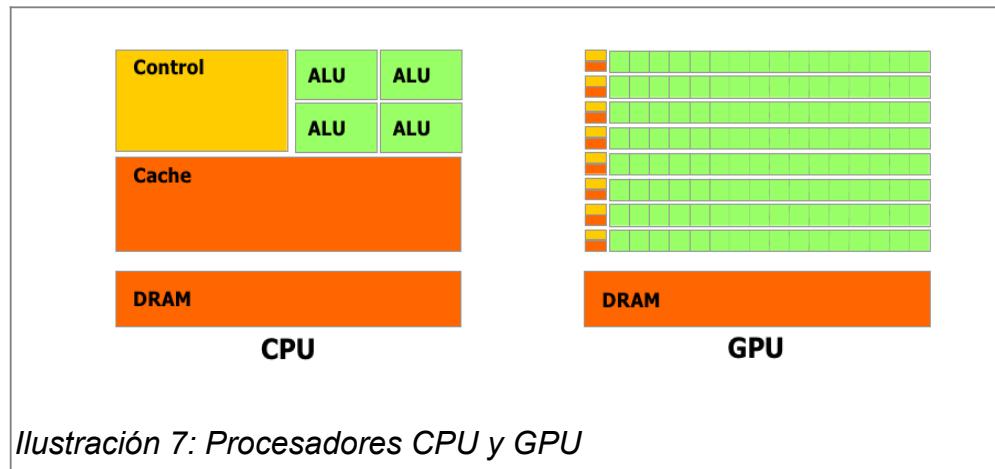
Se construyen sistemas SPP grandes interconectando dos o más nodos a través de la segunda capa de memoria, de modo que esta capa aparece lógicamente, ante los nodos, como una memoria global compartida.

La memoria de dos niveles reduce el tráfico de bus debido a que solamente ocurren actualizaciones para mantener coherencia de memoria. Por tanto, SPP ofrece facilidad de programación del modelo SMP, a la vez que provee una escalabilidad similar a la de un diseño MPP.

## 2.3 Procesamiento Paralelo en GPU

GPU computing, o GPGPU (GPU de propósito general), es el uso de la GPU (unidad de procesamiento gráfico) para realizar operaciones de cálculo científico o técnico de propósito general.

El modelo empleado para esta tecnología se basa en el uso combinado de una CPU y una GPU en un sistema de co-procesamiento heterogéneo. La parte secuencial de la aplicación se ejecuta en la CPU y las partes de mayor carga computacional se aceleran en la GPU. Para el usuario, la aplicación simplemente se ejecuta más rápido porque utiliza la gran capacidad de la GPU para multiplicar el rendimiento.



Como se indica en la ilustración 7, el GPU dedica más transistores del chip a procesadores y menos a memoria. A lo largo de los años, la GPU ha evolucionado hasta alcanzar teraflops de rendimiento en las operaciones de cálculo en coma flotante. Los GPU constan de cientos de núcleos de procesamiento que operan de forma conjunta para manejar los datos de la aplicación a mayor velocidad.

Asignar una función a la GPU implica reescribir ésta para aprovechar el paralelismo del procesador gráfico y agregar palabras clave de "C" para transferir los datos hacia y desde la GPU. El programador se encarga de lanzar decenas de miles de procesos (threads) de forma simultánea. El hardware de la GPU maneja estos procesos y programa su ejecución.

*"Las GPUs han evolucionado hasta un punto en que muchas de las aplicaciones industriales actuales se ejecutan en ellas con niveles de rendimiento muy superiores a los que ofrecerían si se ejecutasesen en sistemas multinúcleo. Las arquitecturas informáticas del futuro serán sistemas híbridos con GPUs compuestas por núcleos de procesamiento paralelo que trabajarán en colaboración con las CPUs multinúcleo".*

*Prof. Jack Dongarra*

*Director del Innovative Computing Laboratory*

Universidad de Tennessee

### **2.3.1 Historia de la GPU**

Los chips de gráficos empezaron siendo canales de procesamiento de gráficos con funciones fijas. Con el paso de los años, estos chips se fueron haciendo más programables, lo que permitió a NVIDIA introducir la primera GPU o unidad de procesamiento gráfico del mercado. Entre los años 1999 y 2000, científicos e investigadores de disciplinas, como el diagnóstico por imagen o el electromagnetismo, empezaron a usar las GPUs para ejecutar aplicaciones de cálculo de propósito general y descubrieron que el enorme rendimiento de la GPU en operaciones de coma flotante producía un extraordinario aumento de la velocidad de ejecución en una gran variedad de aplicaciones científicas. Fue el nacimiento de un nuevo concepto denominado GPGPU o GPU de propósito general.

El problema era que este tipo de procesador tenía que programarse utilizando lenguajes de programación de gráficos como OpenGL y Cg. Los desarrolladores tenían que dar a sus aplicaciones científicas la apariencia de aplicaciones gráficas transformándolas en problemas que dibujasen triángulos y polígonos. Esto limitaba la posibilidad de aprovechar el tremendo rendimiento de las GPUs para usos científicos.

Las empresas fabricantes de hardware gráfico se dieron cuenta de este problema y decidieron invertir en modificar la GPU a fin de hacerla totalmente programable para aplicaciones científicas y añadir soporte para lenguajes de alto nivel como C, C++ y Fortran. Algunas como Intel y Nvidia crearon nuevas arquitecturas como Larrabee y CUDA respectivamente para aprovechar esta tecnología, AMD adaptó su arquitectura original para el uso de OpenCL y otros lenguajes.

En la actualidad existe soporte para programar la GPU con C, C++, Fortran, OpenCL y DirectCompute. Los desarrolladores disponen de una serie de herramientas de desarrollo de software, junto con diferentes librerías y componentes de Middleware. Estos recursos permiten programar la GPU utilizando C u otros lenguajes de programación con una cantidad mínima de palabras clave o extensiones.

### **2.3.2 Funcionamiento**

Las GPU actuales pueden tener 128, 256 ó 512 núcleos (frente a los 8 núcleos de los actuales procesadores más avanzados) o incluso más de 800 en algunos modelos de AMD. Es decir, podemos tener cientos de hilos corriendo concurrentemente en GPU. Si estamos acostumbrados a utilizar

threads, vemos cómo tenemos que programar el hilo y crearlos, y si tenemos 512 núcleos deberíamos crear los 512 hilos; el problema y la solución vienen, en la programación GPU, en que todos los núcleos ejecutarán el mismo código, y cada hilo tendrá una serie de identificadores que nos indicarán qué procesador está funcionando. Por tanto podemos lanzar 512 tareas simultáneas y cada una será lanzada por un núcleo diferente. Si, por ejemplo queremos sumar un vector de 512 elementos con otro, en CPU realizamos una operación cada vez (hasta 512 operaciones), en GPU lanzaremos las 512 a la vez, y terminarán en el tiempo de una operación (más o menos y a grosso modo)

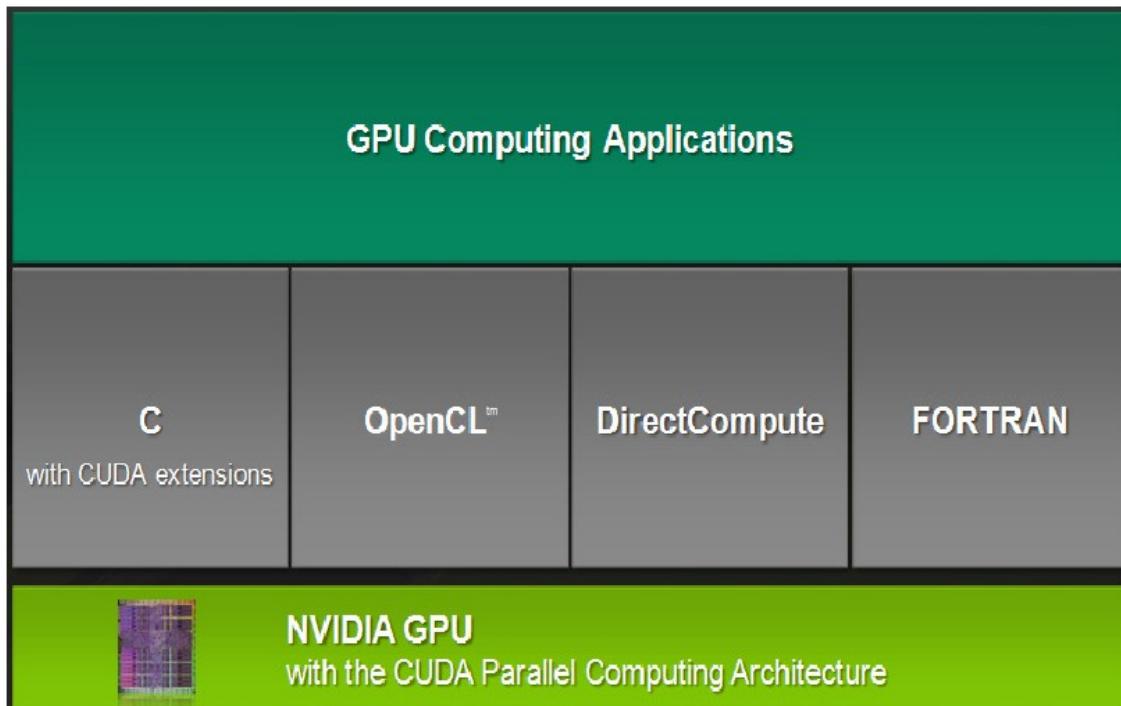
### **2.3.3      Velocidad de Acceso**

Las tarjetas gráficas, por lo general, tienen un tipo de memoria RAM, que es más actual que la memoria de los procesadores (mientras en CPU vamos por DDR3, las gráficas ya usan DDR5). La principal razón es que esta memoria está soldada en la placa y precisamente muy cerca de la GPU, además, el controlador de memoria está integrado en GPU, lo que facilita las transacciones. Por esto y mucho más, una lectura o escritura en la memoria de la tarjeta gráfica consumirá pocos ciclos en GPU, mientras que los accesos a memoria en CPU consumen muchos ciclos de reloj. Con esto, tenemos una lectura/escritura en memoria mucho más rápida, también posee una memoria compartida, mucho más pequeña (de varios Kilobytes) que permite una interacción más rápida. Es algo así como la caché de la CPU, pero en este caso es una memoria que sí podemos controlar (leer y escribir en esa memoria lo que queremos y cuando queremos, mientras que la caché de la CPU es automática).

### **2.3.4      CUDA**

La tecnología creada por la empresa de chips gráficos Nvidia, CUDA (Computer Unified Device Architecture) se creó inicialmente para ser utilizada exclusivamente en tarjetas gráficas desarrolladas por la propia empresa. Esta no se limita a un estándar para programación, ya que es una plataforma completa creada exclusivamente para poder ejecutar procesos que normalmente se ejecutan en un CPU en un GPU.

CUDA crea varias capas de abstracción entre el programador y la GPU, las cuales se representan en la Ilustración 8.



*Ilustración 8: CUDA - Diseñado para varios lenguajes y APIs*

El dispositivo compatible con CUDA puede ser accedido por distintos lenguajes de programación, frameworks y API's tales como CUDA (esta vez como API de programación) o alguna implementación de OpenCL. El dispositivo también se puede utilizar mediante tecnologías tradicionales como C, C++, OpenGL, entre otras.

Actualmente se está integrando la tecnología CUDA en otro tipo de dispositivos, los procesadores ARM. Debido a la liberación de CUDA como framework y API de programación es de esperar que sean desarrolladas implementaciones para otras arquitecturas.

### 2.3.5      **OpenCL**

OpenCL, es un estándar creado por Kronos Group y se caracteriza por ser el primer estándar de programación paralela multiplataforma libre de impuestos o restricciones que existe en el mercado (hasta la liberación de CUDA hace unos meses). Desde su inicio fue pensado para funcionar implementado sobre variados dispositivos y arquitecturas que estuviesen fabricadas respetando el estándar impuesto por el Kronos Group, y debido a la participación de variadas empresas en este grupo de estandarización, se masificó su uso muy rápidamente, llegando en octubre del 2011 a ser la

segunda herramienta más utilizada en computo HPC después de Intel Threading Blocks y superando a OpenMP.

Como estándar, define una API bajo el mismo nombre: OpenCL, la cual permite escribir programas paralelos sobre distintos dispositivos. Esta API define un lenguaje para escribir kernels, los cuales son funciones capaces de ser ejecutadas en distintos dispositivos. La API completa es bastante parecida a CUDA y tienen parámetros similares.

Una de las características más destacables de OpenCL es que las aplicaciones que son desarrolladas utilizándolo pueden ser compiladas y ejecutadas en cualquier implementación de OpenCL sin necesitar ningún cambio en su código, además una aplicación puede utilizar distintos tipos de procesador independiente de la tecnología que este posea mientras soporte OpenCL.

Un claro ejemplo de esta versatilidad se da por el hecho de que es posible compilar y ejecutar una aplicación OpenCL que haga uso de CPU y GPU, utilizando la implementación de un procesador Intel y una tarjeta gráfica AMD. Luego, se puede realizar sin problema el mismo procedimiento con la implementación de un procesador AMD y una tarjeta gráfica Nvidia.

### 3 Descripción del Problema

Lo más importante en el área de la computación gráfica son los modelos que podemos diseñar con los computadores, creando, a partir de primitivas simples objetos complejos y detallados los que a su vez nos permiten crear grandes escenarios. Los modelos en tres dimensiones están compuestos de primitivas como cubos, planos, esferas, cilindros y conos entre otras, las que nos permiten crear formas (Shapes) comunes en nuestro entorno.

El proceso de creación de formas se puede realizar uniendo varias mallas básicas (Mesh) o modificar estas de manera que adopten nuevas formas. Cualquiera sea la técnica utilizada, las mallas siempre se descomponen en tres componentes básicos: vértices, aristas y caras. En algunas aplicaciones se unen estos componentes dándoles distintos nombres y funcionalidades en conjunto.

Durante el proceso de modelado pueden llegar a crearse formas muy detalladas compuestas por miles (o millones) de vértices, los que a su vez conforman más caras y aristas. Mientras más componentes tenga nuestro modelo, tendrá más detalles y será más agradable a la vista, aunque también juega un papel importante la habilidad y experiencia de la persona que crea el modelo.

Para la creación de un escenario 3D debemos ubicar todos los modelos creados en una misma instancia de una aplicación, muchas veces esto se realiza con la misma aplicación que fue utilizada para crear los objetos por separado, dentro de las aplicaciones más conocidas se encuentran 3D Studio, Google Sketch Up y Blender.

Generalmente, los modelos 3D son formas simplificadas de elementos, creados para ser mucho más complejos de lo que se hizo manualmente. Estos modelos están destinados a pasar por un proceso en el cual serán procesados por un computador y este, mediante alguna configuración establecida, creará una imagen, la cual podrá ser visible en un monitor plano (2D). Este proceso recibe el nombre de renderizado.

Para renderizar un modelo, es necesario que utilicemos una aplicación específica para esta tarea, la cual se encargará de tomar la información de todos los componentes de modelo para traducirlos en una imagen. Estas aplicaciones tienen el nombre de motor de renderizado, y muchas veces ni siquiera pertenecen a una aplicación de modelado en específico. Los motores de renderizado aplican a la imagen final todos los detalles que el modelo no muestra durante su creación, tales como

texturizado, pintado, efectos gráficos y movimiento en el caso de la creación de animaciones. En el *Laboratorio de Experimentación Tridimensional* de la Universidad del Bío-Bío se crearán modelos urbanos con la mayor cantidad de detalles posibles y serán creadas animaciones de los mismos modelos para simular problemas de tránsito reales y buscar soluciones a los mismos.

El renderizado de este tipo de modelos necesita un gran poder de procesamiento, por lo que se cuenta con los equipos necesarios para la configuración de un cluster de alto rendimiento y además con tarjetas gráficas con la capacidad de procesar mediante la utilización de su GPU aplicaciones creadas para este tipo de procesador.

En la actualidad, Blender no utiliza el procesador gráfico (GPU) para mejorar sus tiempos de renderizado, por lo que es necesario crear una aplicación que obligue a Blender a ocupar la totalidad de los recursos disponibles.

## 4 Origen del Proyecto

Durante el año 2010 la Universidad del Bío-Bío se hizo partícipe de el proyecto Alfa III GAVIOTA (Grupos Académicos para la Visualización Orientada por Tecnologías Apropiadas), el cual tiene la finalidad de permitir equipar a las universidades socias, con equipamiento de última generación para la realización de aplicaciones de realidad virtual y realidad aumentada.

Dentro del marco de este proyecto se requiere la utilización de una herramienta de procesamiento poderosa para renderizar los modelos urbanos en 3 dimensiones. Es por ello que para nuestra solución de cómputo se decidió optar por el software libre, por su adaptabilidad a la situación presentada y su facilidad para el manejo de soluciones de procesamiento paralelo implementando un cluster de alto rendimiento.

En este proyecto nos limitaremos solo a implementar y optimizar el cluster dejando una plataforma base estable para la posterior creación de modelos, interfaces que no formarán parte de este proyecto.

### 4.1 Objetivos

#### *Objetivos Generales*

1. Crear una solución de computo mediante la utilización de CPU y GPU con el fin de reducir los tiempos de renderizado de modelos urbanos de alta definición en tres dimensiones, sin, en ningún momento olvidar la función que deben cumplir los equipos al estar disponibles para modelado 3D y programación.

#### *Objetivos Específicos*

1. Investigar de los protocolos y lenguajes adecuados a utilizar para realizar la parallelización entre CPU y GPU.
2. Generar una estrategia para la distribución de procesos uniforme entre los nodos.
3. Estudiar el tiempo de renderizado de forma de encontrar el rendimiento óptimo (Benchmark).
4. Integrar de forma conjunta el cómputo utilizando CPU y GPU.

## **4.2 Metodología**

Basaremos nuestro trabajo en una metodología en base a pruebas constantes. Una vez que se termine una etapa, dentro de nuestra metodología en base a pruebas, deseamos centrarnos en pequeños objetivos en el menor tiempo, esto nos permitirá inspeccionar nuestro trabajo más rápido y fácilmente así como medir la calidad de este.

Al final de cada etapa de las contempladas en la planificación se pueden ver y medir los resultados. Así nos auto-organizaremos como equipo de trabajo. Habrán tareas donde cada uno será el encargado y otras donde todo el equipo actuara como responsable, a pesar de esto se priorizaran aquellas actividades que sea necesario realizar en equipo.

Se llevará un registro de cada prueba en donde se analizará el mejor desempeño al realizar las modificaciones planeadas para cada etapa.

Como requisito de nuestra metodología de trabajo será necesario estar presencialmente en el laboratorio un mínimo de 2 días a la semana durante la duración del proyecto. Además será necesario reunirnos al menos 1 día a la semana para revisar los avances, identificar problemas, soluciones y realizar lluvia de ideas de ser necesario.

## **5 Solución del Problema**

### **5.1 Resumen del Problema**

En el Grupo de *Experimentación Tridimensional* de la Universidad del Bío-Bío se crearan modelos urbanos con la mayor cantidad de detalles posibles y serán creadas animaciones de los mismos modelos para simular problemas de transito reales y buscar soluciones a los mismos.

El renderizado de este tipo de modelos necesita un gran poder de procesamiento, por lo que se cuenta con los equipos necesarios para la configuración de un cluster de alto rendimiento y además con tarjetas gráficas con la capacidad de procesar mediante la utilización de su GPU aplicaciones creadas para este tipo de procesador.

En la actualidad, algunos software no utilizan el procesador gráfico (GPU) para mejorar sus tiempos de renderizado, por lo que es necesario crear una implementación que obligue a este a ocupar la totalidad de los recursos disponibles.

### **5.2 Factibilidad de la Solución**

En el estudio de nuestro proyecto, analizaremos el tipo de solución que planteamos frente a otras que podrían ser aplicadas, sin embargo nos enfocaremos a una en especial. Para ello es necesario recurrir a un pequeño estudio de factibilidad y conocer tanto, las ventajas como desventajas que nos presenta nuestra solución, esto nos ayudara a decidir que tan asequible y rentable será.

#### **5.2.1 Estudio de Mercado**

Tendrá como finalidad analizar si existe una demanda real para la implementación de la solución de cómputo que justifique la puesta en marcha el proyecto.

Demandas del Producto

Para la creación de modelos en tres dimensiones en el área de urbanismo dentro de la universidad del Bío-Bío, es necesario ahorrar tiempo en el renderizado, debido a que es un factor crucial para aprovechar los recursos humanos existentes en trabajo de modelado y creación de software y no dar

cabida a espacios ociosos en los que se esté esperando al resultado final, mientras se pierde la disponibilidad de los equipos para la realización de otras tareas.

El espectro de usuarios que utilizarán esta solución corresponden a alumnos que participaron activamente en la propuesta de soluciones para la puesta en marcha del proyecto, sin embargo no se limita a solo esto, ya que la configuración utilizada debe ser aplicable a agentes externos a la universidad y se planea una implementación futura inicialmente al Departamento de Transito de la Municipalidad de Concepción y posteriormente a distintas municipalidades.

### Oferta del Producto

Existen actualmente en el mercado soluciones prefabricadas de alto rendimiento y uso eficiente de recursos (las cuales pueden ser escalables) adquiriendo hardware físicamente o contratando un servicio de procesamiento distribuido remoto. Dentro de este marco de productos existen opciones de pago como Azserver, RenderPro y Cray, entre otras. Este tipo de soluciones a pesar de ser convenientes para funciones específicas, no nos brindan la adaptabilidad necesaria para la realización de múltiples tareas y configuraciones inherente al proyecto.

Nuestra solución consiste en la implementación de un cluster de alto rendimiento con procesamiento paralelo utilizando GPU y CPU, esto implica usar una configuración de cluster diseñada por nosotros, la cual conlleva la utilización de software de código abierto debido a su adaptabilidad, bajo costo y libertad de uso, por lo que la Universidad del Bío-Bío aprovechará los fondos disponibles en la adquisición de hardware, potenciando el uso de nuevas tecnologías de punta e impulsando el desarrollo regional, además fomentará en la Universidad del Bío-Bío futuras investigaciones.

La implementación de un cluster que cumpla los requisitos para explotar completamente los recursos disponibles y otorgar libertad de uso, es lo crucial de nuestra solución, lo cual sera beneficioso para brindar la adaptabilidad necesaria en modificaciones futuras.

A modo de conclusión de las opciones vistas podemos decir, que ocupando nuestra solución seremos capaces de impulsar a la Universidad del Bío-Bío como base del desarrollo científico y arquitectónico de la región a un costo menor que otras opciones.

## **5.2.2      Estudio Técnico**

Tiene por objetivo proveer información para cuantificar el monto de la inversión del proyecto y operaciones pertinentes del área.

La solución de cómputo que se estudiará tiene por principal uso la creación y posterior renderizado de modelos en tres dimensiones, para ello es necesaria la implementación de algún tipo de Middleware que nos simplifique el uso y configuración de la plataforma de trabajo, al mismo tiempo aprovechar al máximo los recursos de hardware elegidos específicamente para esta tarea.

El cluster está compuesto por dos tipos de equipos, el servidor o Head Node y los nodos o Render Nodes.

El servidor estará dedicado a trabajar con procesos que no sea posible o eficiente enviar como carga de trabajo a los nodos, como es el caso de los procesos no paralelizables. Además al servidor le corresponderá el control de colas y asignación de prioridades en los trabajos a realizar, también constara de 2 pantallas para presentaciones de los modelos finalizados en tres dimensiones. En vista de un posible desarrollo de software servirá aplicaciones de desarrollo tales como control de versiones o administración y planificación de proyectos.

Los Render Nodes serán los equipos destinados a trabajar en conjunto al Head Node a los cuales se les enviaran los procesos de renderizado para procesarlos a imágenes de alta definición.

Estos equipos compartirán un directorio común de trabajo en un NAS (Network Attached Storage) en el cual estarán todos los archivos de trabajo.

Dada la ventaja de utilizar software libre para la implementación del cluster, priorizaremos el uso de licencias que nos brinden la mayor adaptabilidad posible sin un costo monetario adicional, obviamente respetando las condiciones de uso de la licencia.

La realización de este proyecto dependerá de la adquisición del hardware por parte de la universidad del Bío-Bío y comenzara la implementación de acuerdo al siguiente plan de trabajo:

## Plan de Trabajo

La actividad se desarrollará de la siguiente forma:

*-Investigación, preparación y planificación de configuraciones previas:*

Un análisis teórico el cual se iniciará intensivamente durante Marzo y Abril y terminará cuando esté disponible el hardware esperado.

*-Configuración básica de los equipos:*

Instalación de los SO y del software analizado en al etapa anterior.

*-1ra Etapa de pruebas:*

Se realizará apenas esté configurado el cluster de la forma más básica

*-Configuración adicional y programación de GPU:*

Se configurará y programara Blender para que procese el renderizado por GPU

*-2da etapa de pruebas:*

Se harán pruebas de rendimiento para medir las mejoras

Luego se volverán a repetir las pruebas hasta obtener una configuración óptima

Cada etapa contendrá sub-etapas que serán definidas a partir del comienzo del proyecto.

## Carta Gantt

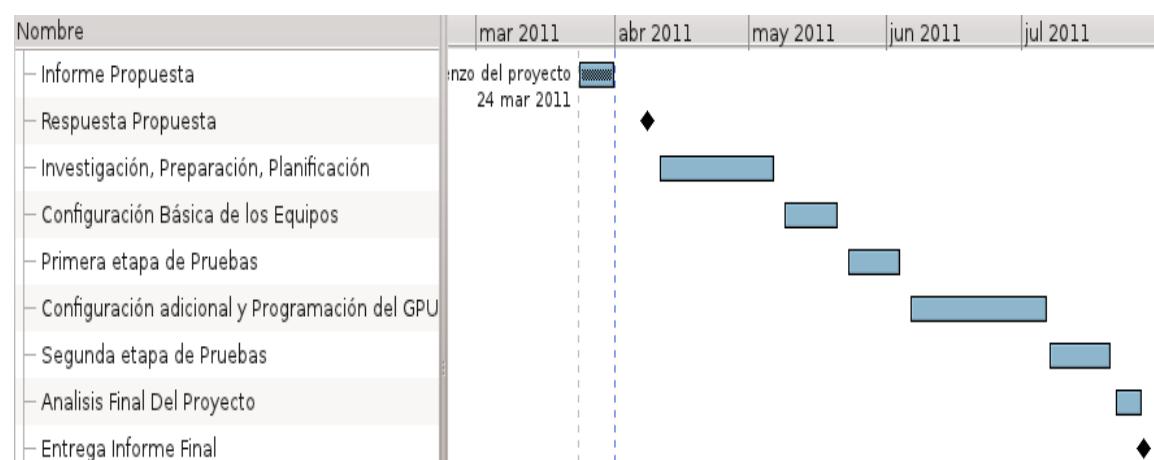


Tabla 1: Carta Gantt

Al tratarse de equipos configurados en forma personalizada el costo de mantenimiento y existencia de repuestos no se elevara más allá del costo de solucionar la misma situación en equipos normales a diferencia de equipos con formatos de fabricación propios de su marca, para los cuales puede ser difícil la adquisición de repuestos. Los equipos cuentan con formato estándar para realizar actualizaciones.

Dada las ventajas de la solución presentada, es claro que se debe optar por la solución personalizable, la cual brinda libertad de decisión y acción sobre la ejecución del proyecto.

### **5.2.3 Estudio Financiero**

Dentro del marco de la realización de nuestro proyecto también debemos considerar el hecho de la inversión realizada para la compra de equipos, sin embargo puede ser necesaria la adquisición de licencias para usar software determinados y específicos para la solución. Aun así en la solución que planteamos deseamos minimizar los costos financieros, es por ello que un aspecto importante es la justificación de porque usar software libre.

¿Qué ventajas nos otorga el uso de software libre en la implementación del cluster?

- Su obtención es gratuita, además la personalización y la adaptación será gracias al trabajo realizado en nuestra investigación.
- Las plataformas GNU/GPL son más seguras y estables. Hay un equipo amplio de técnicos desarrollando en la comunidad constantemente, mejorando el producto con actualizaciones y módulos totalmente gratuitos. Además esta misma característica evita que se hagan gastos en mantención del software.
- La casi inexistencia de virus, malware, backdoor, troyanos, etc... existentes en este universo open source es una razón evidente.
- La independencia del proveedor. No depende de la oferta del proveedor del software y de los precios que este imponga.
- El software libre es un vehículo de transmisión de conocimiento, lo que conlleva a la difusión del proyecto lo que puede traer inversionistas a futuro.

### **5.2.3.1      *Horas hombre***

El presente proyecto no genero costos a la Universidad del Bío-Bío respecto a la horas hombre trabajadas en la configuración del cluster, pero estas fueron calculadas para referencias futuras.

Horas hombre							
Herramienta/Servicio	Nombre	Precio	Moneda	Unidad	Cantidad	Total	
Configuración de Cluster	Ingeniero Informático	1,2	UF	Hora Hombre	160	192	
Agregar un nodo	Ingeniero Informático	1,2	UF	Hora Hombre	8	9,6	

*Tabla 2: Estudio Financiero - Horas Hombre*

Además, se indica que deberá cubrirse un costo en el caso supuesto de agregar un nodo adicional, lo que implica la inversión en un Ingeniero Informático capacitado en MOSIX o computación paralela.

### **5.2.3.2      *Licencias de Software***

Uno de los factores que incentivo el uso de software libre gratuito fue tomada por que el software libre pagado y el software privativo, generalmente, solo tiene diferencias de poco peso para una configuración tan particular como la de un cluster. A excepción del servicio técnico el cual fue omitido por el hecho de tratarse de un proyecto de tesis que contaba con dos personas para realizar este tipo de actividades. A pesar de eso, se evaluaron los costos de opciones pagadas frente a las opciones gratuitas utilizadas.

Herramienta/Servicio	Nombre	Precio	Moneda	Unidad	Cantidad	Total
Software Administrador de Cluster	PVS Pro	20,48	USD	Núcleo	40	819,2
Software Administrador de Cluster	MOSIX*	0	-	-	40	0
Sistema Operativo	SLE Server	349	USD	Nodo	5	1745
Sistema Operativo	Debian	0	-	-	5	0

*Tabla 3: Estudio Financiero - Licencias de Software*

\*Este precio de MOSIX es cero por tratarse de una institución educacional. En caso de querer replicar esta configuración fuera del área educacional se recomienda estudiar alternativas como OpenMosix.

El costo total en licencia de software podría haber superado USD\$2.000.

## **5.3 Tipos de Configuraciones de Cluster**

### **5.3.1 Tipo de Enfoque para el Procesamiento Paralelo**

En los últimos años se ha comenzado a explotar la utilización de más de un procesador a la vez, esto ofrece una gran ventaja en cuanto a costos. Sin embargo, su principal beneficio, la escalabilidad (crecer hacia arquitecturas de mayor capacidad), puede ser difícil de alcanzar. Esto se debe a que conforme se añaden procesadores, las disputas por los recursos compartidos se intensifican.

Algunos diseños diferentes de procesamiento paralelo enfrentan este problema fundamental:

#### **5.3.1.1 Multiprocesamiento Simétrico (SMP)**

Ventajas: Tiene un diseño simple pero aún así efectivo. Múltiples procesadores comparten la memoria RAM y el bus del sistema, tiene solamente un espacio de memoria, lo que simplifica tanto el sistema físico como la programación de aplicaciones.

Desventajas: Conforme se añaden procesadores, el tráfico en el bus de memoria se satura, generalmente el bus se convierte en un cuello de botella al manejarse alrededor de ocho o más procesadores, es considerada una tecnología no escalable.

#### **5.3.1.2 Procesamiento Masivamente Paralelo (MPP)**

Ventajas: Evitar los cuellos de botella en el bus de memoria, MPP no utiliza memoria compartida. En su lugar, distribuye la memoria RAM entre los procesadores de modo que se semeja a una red, los procesadores utilizan un esquema de paso de mensajes análogo a los paquetes de datos en redes, permite sistemas de gran tamaño con cientos y aún miles de procesadores. MPP es una tecnología escalable.

Desventaja: Son difíciles de programar porque las aplicaciones se deben dividir de tal manera que todos los segmentos que se ejecutan se puedan comunicar unos con otros, esto se debe a que cada CPU tiene su propia memoria.

### **5.3.1.3      *Procesamiento Paralelo Escalable (SPP)***

Ventajas: Utiliza una memoria jerárquica de dos niveles para alcanzar la escalabilidad. La primera capa de memoria consiste de un nodo que es esencialmente un sistema completo, con múltiples procesadores y su memoria globalmente compartida. La memoria de dos niveles reduce el tráfico de bus debido a que solamente ocurren actualizaciones para mantener coherencia de memoria. Por tanto, SPP ofrece facilidad de programación del modelo SMP, a la vez que provee una escalabilidad similar a la de un diseño MPP.

### **5.3.2      Optimizar Rendimiento, Distribución de Trabajos**

Para lograr este punto, se ve la necesidad de implementar un sistema basado en Procesamiento paralelo escalable, este al ser una mezcla de multiprocesamiento simétrico y procesamiento masivo paralelo, nos da la ventaja de optimizar los recursos compartidos en la implementación del cluster. Los nodos tienen una configuración homogénea entre ellos, es por esto que la utilización de este método nos garantiza un rendimiento óptimo y parejo de las tareas a procesar. También nos dará las bases para la ejecución de balanceo de carga sobre el sistema y no caída de un nodo en caso de saturación por extrema capacidad de cálculo.

Alternativas de Solución: SW Libre vs SW Privativo

En nuestra investigación sobre que es un clúster y cómo funcionan, nos hemos encontrado con algunas soluciones aplicables al sistema que queremos optimizar. Una de ellas es la de usar un lenguaje de programación paralela.

### **5.3.3      Lenguajes de Programación Paralela**

Existen varios lenguajes de programación paralela, sobresaliendo de estos MPI (Message Passing Interface) y PVM (Parallel Virtual Machine), por ser uno de los estándares más aceptados.

MPI consiste de una biblioteca estándar para programación paralela en el modelo de intercambio de mensajes. En este estándar se han incluido los aspectos más relevantes de otras bibliotecas de programación paralela.

Entre las ventajas de MPI se encuentra la disponibilidad de varios modos de comunicación, los cuales permiten al programador el uso de buffers para el envío rápido de mensajes cortos, la sincronización de procesos o el traslape de procesos de cómputo con procesos de comunicación. Esto último reduce el tiempo de ejecución de un programa paralelo, pero tiene la desventaja de que el programador debe ser más cuidadoso para evitar la corrupción de mensajes. Dos de las principales distribuciones libres de MPI son: LAM/MPI y MPICH.

PVM se comenzó a desarrollar en el verano de 1989 por el Oak Ridge National Laboratory, y posteriormente junto con la Universidad de Tennessee en los EUA. Es una biblioteca de envío de mensajes, totalmente libre, capaz de trabajar en redes homogéneas y heterogéneas, y que hace uso de los recursos existentes en algún centro de trabajo para poder construir una máquina paralela de bajo costo, obteniendo su mejor rendimiento en "horas muertas". Maneja transparentemente el ruteo de todos los mensajes, conversión de datos y calendarización de tareas a través de una red de arquitecturas incompatibles. Está diseñado para conjuntar recursos de cómputo y proveer a los usuarios de una plataforma paralela para correr sus aplicaciones, independientemente del número de computadoras distintas que utilicen y donde éstas se encuentren localizadas. El modelo computacional de PVM es simple y además muy general. El usuario escribe su aplicación como una colección de tareas cooperativas. Las tareas acceden los recursos de PVM a través de una biblioteca de rutinas. Estas rutinas permiten la inicialización y terminación de tareas a través de la red, así como la comunicación y sincronización entre tareas.

Los constructores de comunicación incluyen aquellos para envío y recepción de estructuras de datos así como primitivas de alto nivel, tales como emisión, barreras de sincronización y sumas globales.

Estos lenguajes son muy complejos en desarrollo, punto a considerar cuando la aplicación a usar debe ser paralelizada, lo que nos daría un gran problema temporal y uso de recursos.

Para solucionar esto, se investigó el uso de otros métodos más eficientes y que nos de mayores prestaciones. Nos encontramos con soluciones integrales que hacen uso de librerías como las mencionadas anteriormente y las potencian, haciendo incluso posible el uso de toda una batería de aplicaciones para mediciones y estadísticas, tanto así como monitores de recursos, configuración de equipos más eficiente y óptima entre otros. A esto se le denomina Middleware y es el encargado de interactuar entre el usuario y el cluster en sí. Provee de las siguientes características.

- Una interfaz única de acceso al sistema, denominada SSI (Single System Image), la cual genera la sensación al usuario de que utiliza un único computador muy potente.
- Herramientas para la optimización y mantenimiento del sistema: migración de procesos, checkpoint-restart (congelar uno o varios procesos, mudarlos de servidor y continuar su funcionamiento en el nuevo host), balanceo de carga, tolerancia a fallos, etc.
- Escalabilidad: debe poder detectar automáticamente nuevos servidores conectados al cluster para proceder a su utilización.

También cabe mencionar que permite el uso de la paralelización en aplicaciones no paralelizadas mediante lenguajes tradicionales de paralelización para cluster como MPI. Esto es sin duda una gran solución y avance en el desafío de la optimización y utilización de recursos a alto nivel, haciendo esto transparente para el usuario frente al uso de la aplicación.

Aplicaciones como [MOSIX](#), [OpenMOSIX](#), [Cóndor](#), [OpenSSI](#), son Middleware's. En si son clusters y se encargan de recibir los trabajos entrantes y los redistribuye de manera que el proceso se ejecute más rápido, el sistema no sufre sobrecargas en un servidor. Para lograr esto se realizan políticas (automáticamente o por un administrador) que le indican dónde y cómo debe distribuir los procesos, por un sistema de monitorización, el cual controla la carga de cada CPU y la cantidad de procesos en él.

El Middleware también debe poder migrar procesos entre servidores con distintas finalidades:

- Balancear la carga
- Mantenimiento de servidores
- Priorización de trabajos

### **5.3.4      Middleware para la implementación del Cluster**

Anteriormente se nombraron algunos como MOSIX, OpenMosix, Condor, OpenSSI. Se estudiará cual de ellos nos dará las mejores estimaciones para lograr una implementación eficiente al problema.

## Descripción de los Middleware Estudiados

### 5.3.4.1 CONDOR

Condor es un producto de Condor Research Project de la Universidad de Wisconsin, Madison y desarrollado por el Departamento de Ciencias de la computación hace ya más de 10 años. Condor es un software de código abierto. Cientos de organizaciones en la industria, el gobierno de EEUU y las universidades utilizan Condor para establecer ambientes de cómputo que superan el rango de cientos de estaciones de trabajo.

Condor es un sistema de administración especializado para monitorizar y satisfacer necesidades computacionales en trabajos de cómputo intensivos. Este sistema provee un mecanismo de manejo de colas, políticas de planificación, esquema de prioridades, monitores de recursos, y administración de los mismos.

Los usuarios realizan peticiones a Condor, que luego son colocadas en una cola, en donde mediante un proceso de selección se establece en dónde y cuándo se ejecutarán.

Entre las funcionalidades más importantes de Condor, se puede mencionar:

- ✓ Entrega distribuida
- ✓ Prioridades de usuario
- ✓ Prioridades de tareas
- ✓ Dependencia de tareas
- ✓ Soporte de tareas simultáneas
- ✓ Puntos de verificación (checkpoints) y migración de tareas
- ✓ Suspensión y reanudación de tareas
- ✓ Autenticación y autorización
- ✓ Plataformas heterogéneas

### **5.3.4.2 MOSIX**

Es una extensión del kernel de Linux para clusters y multi-clusters orientado para la computación de alto rendimiento (HPC) en las plataformas Linux X86, incluyendo clusters, multi-clusters, cluster con GPU y la nube. MOSIX soporta los procesos interactivos y concurrentes además de trabajos por lotes. Incorpora la detección automática de recursos y la distribución dinámica de la carga de trabajo, encontrada comúnmente en computadores personales con varios procesadores.

MOSIX está implementado como una capa de software que permite a las aplicaciones ejecutarse en los nodos remotos como si se ejecutaran de forma local. Los usuarios pueden iniciar aplicaciones (de tipo secuencial, paralela y GPU) en un nodo, mientras MOSIX buscara los recursos automáticamente y los ejecutara de forma transparente en otros nodos. No existe la necesidad de modificar o enlazar las aplicaciones con algunas biblioteca, copiar archivos, o iniciar sesión remotamente en otros nodos o incluso asignar procesos a distintos nodos de una GPU, todo es hecho automáticamente. La distribución de la aplicación en los nodos es supervisada por un conjunto de algoritmos exhaustivos que monitorean el estado de los recursos y mejoran el rendimiento mediante una asignación dinámica de los recursos, por ejemplo balanceo de carga.

Una cualidad única de MOSIX es que trabaja a nivel de proceso, a diferencia de otros sistemas operativos que operan a nivel de trabajos. Esto significa que el sistema se adapta y redistribuye la carga de trabajo cuando el numero de procesos o demanda de un trabajo cambia (usando fork y exit). Esto es especialmente útil en trabajos paralelos.

La última versión de MOSIX puede manejar clusters, Multi-cluster y cluster con GPU. El manejo flexible permite a los administradores de distintos clusters compartir sus recursos computacionales, mientras conservan la autonomía de desconectar sus clusters en cualquier momento, sin interrumpir procesos que ya están siendo ejecutados. Un multi-cluster MOSIX se puede extender indefinidamente mientras haya confianza entre los administradores del cluster. Esto debe incluir las garantías de que las aplicaciones huésped no serán alteradas mientras se ejecutan en un cluster remoto y que un computador hostil no puede ser conectado a la red local. En la actualidad estos son los requerimientos estándar de cualquier cluster y multi-cluster dentro de una organización.

MOSIX puede ser ejecutado de forma nativa o como una máquina virtual. En forma nativa se obtiene un mejor rendimiento pero requiere la modificación de un kernel basado en Linux, a diferencia de una maquina virtual que puede ejecutarse sobre un sistema operativo que soporte virtualización sin necesidad de modificaciones, incluyendo Windows, Linux y OS-X

MOSIX es adecuado para ejecutar aplicaciones de alto rendimiento con poca o moderada cantidad de entrada y salida de datos. Es particularmente adecuado para:

- La utilización eficiente de sistemas con muchos recursos – Mediante la detección automática de recursos y balanceo de carga.
- Ejecutar aplicaciones con tiempos de ejecución o requerimientos de recursos impredecibles.
- Ejecutar procesos largos – los cuales son enviados automáticamente a nodos en clusters remotos y movidos de vuelta cuando estos nodos son desconectados
- Combinación de nodos de distintas velocidades - mediante la migración de procesos entre los nodos en función de su respectiva velocidad, la carga actual y la memoria disponible.

Al igual que Condor, MOSIX posee las mismas características y muchas otras, las principales son:

- ✓ Proporciona los aspectos de una única imagen de sistema.
- ✓ Los usuarios pueden ingresar en cualquier nodo.
- ✓ No hay necesidad de modificar o vincular las aplicaciones corriendo en el cluster con bibliotecas especializadas.
- ✓ No hay necesidad de copiar los archivos a los nodos remotos.
- ✓ Manejo automático de recursos y carga de trabajo.
- ✓ Migración de proceso.
- ✓ Balanceo de carga.
- ✓ La migración de procesos del nodo más lento al más rápido.
- ✓ Elección de ejecutar el proceso en uno o mas nodos, usando un rango de nodos.
- ✓ Uso de plataformas heterogéneas, soporta arquitecturas 32 o 64 bits.
- ✓ Utilización de nodos independiente de sus capacidades y arquitecturas.
- ✓ Sockets migrables para la comunicación directa entre los procesos de migración.
- ✓ Ejecución de procesos en un ambiente cerrado (sandbox).
- ✓ Trabajos por lotes.
- ✓ Herramientas: instalación automática y scripts de configuración, monitores en línea.

- ✓ Entrega distribuida.
- ✓ Prioridades de usuario.
- ✓ Prioridades de tareas.
- ✓ Dependencia de tareas.
- ✓ Soporte de tareas simultáneas.
- ✓ Puntos de verificación (checkpoints) y recuperación.
- ✓ Suspensión y reanudación de tareas.
- ✓ Autenticación y autorización.

#### **5.3.4.3      *OpenMOSIX***

OpenMOSIX es la versión libre de MOSIX, se trata de un sistema de cluster para Linux que permite a varias máquinas actuar como un único sistema multiprocesador (denominado en inglés SSI). Esto permite que no tengamos que reprogramar nuestras aplicaciones para que aprovechen el cluster. Los procesos no saben en qué nodo del cluster se ejecutan, y es el propio OpenMOSIX el responsable de "engañosos", y redirigir las llamadas al sistema al nodo del cluster en el que se lanzó el proceso. OpenMOSIX implementa un algoritmo balanceador que permite repartir de forma óptima la carga, si está el cluster bien calibrado.

Se compone de un parche al kernel, responsable de las migraciones transparentes de procesos, y unas herramientas de área de usuario, necesarias para calibrar y administrar el cluster.

OpenMOSIX puede migrar cualquier proceso mientras que no haga uso de los segmentos de memoria compartida. Según la calibración, migrará procesos más ligeros, o más pesados.

Actualmente el desarrollo de OpenMOSIX está detenido. Sigue funcionando bien para los kernels 2.4 y para el kernel 2.6 aún no funciona completamente. Sin embargo, por su estabilidad y robustez aún hay gente que lo emplea, y sigue instalándolo. El único problema real es con las máquinas que tienen hardware no soportado por el kernel 2.4.

Tiene algunas de las cualidades de MOSIX, pero su desarrollo es lento lo que implica una gran desventaja frente a MOSIX.

#### **5.3.4.4 OpenSSI**

- ✓ OpenSSI es una implementación para clusters de código abierto , provee de un ambiente SSI (Single System Image), Permite que un conjunto de equipos sea tratado como un gran sistema, haciendo que las aplicaciones que se ejecutan en cualquier acceso a la máquina y utilizar todos los recursos del cluster.
- ✓ OpenSSI está basado en Linux y fue lanzado como un proyecto de código abierto por Compaq en 2001. Es la etapa final de un largo proceso de desarrollo, que se remonta a LOCUS, desarrollado en la década de 1980.
- ✓ OpenSSI está diseñado para ser utilizado como cluster de un alto rendimiento y alta disponibilidad, con OpenSSI es posible crear un clúster sin punto único de fallo, por ejemplo el sistema de archivos se puede reflejar entre dos nodos, de modo que si un nodo se bloquea el proceso de acceso al archivo pasa al otro nodo. Como alternativa, el cluster puede ser diseñado de tal manera que cada nodo tiene acceso directo al sistema de archivos.

Algunas de sus características son:

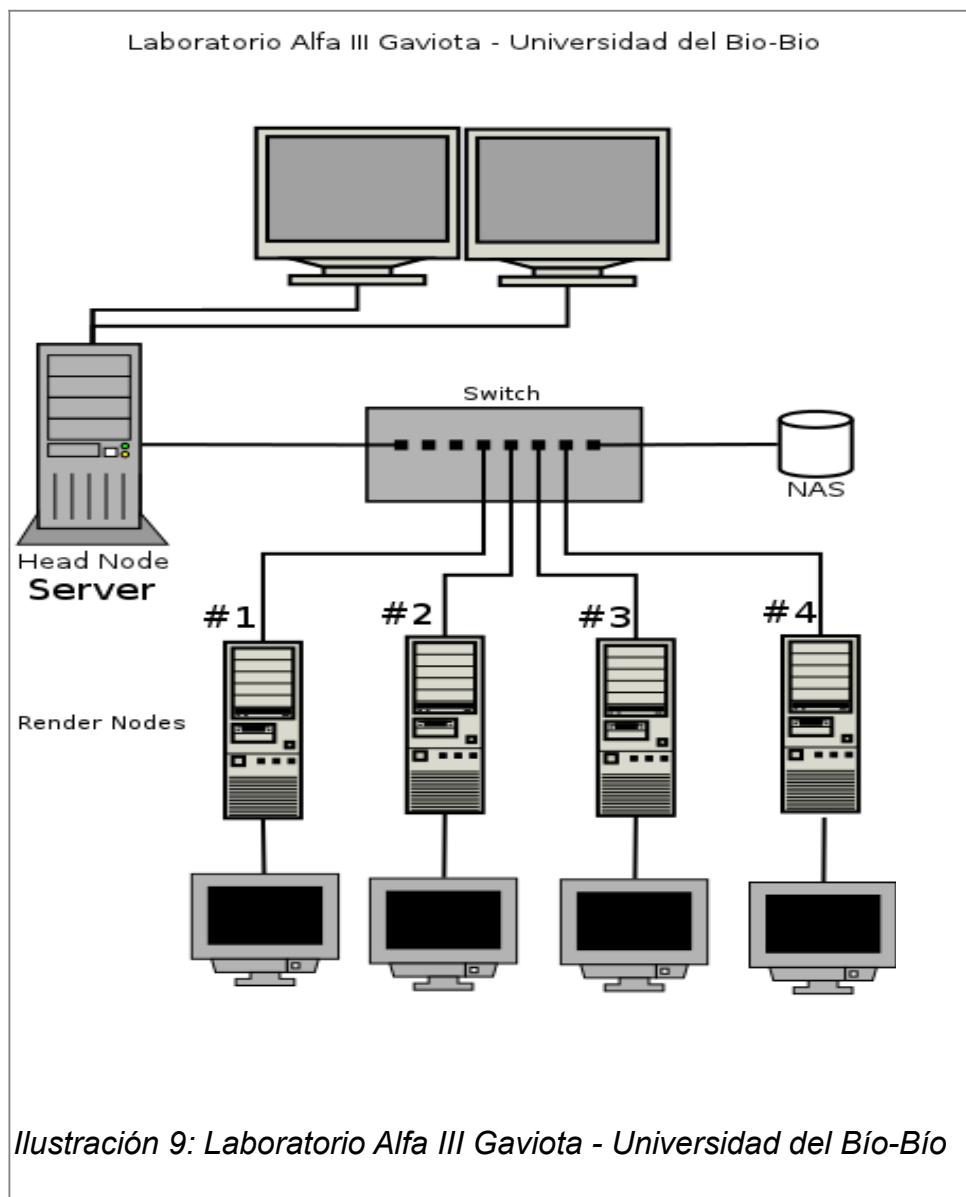
- ✓ Espacio único de proceso.
- ✓ Migración.
- ✓ Raíz único.
- ✓ CFS (Cluster File System)SAN sistemas de archivo cluster.
- ✓ SAN (Storage Area Network).
- ✓ NFS(Network File System).
- ✓ Unidad de estrada/salida únicas.
- ✓ Mecanismo de interprocesos único.
- ✓ Manejo automático IP del cluster.

## 6 Configuración del Cluster

### 6.1 Características Técnicas del Hardware

El cluster tiene una configuración física que intenta explotar al máximo los recursos disponibles los cuales fueron escogidos específicamente para ser usados en este proyecto (Alfa III – Gaviota).

Al ser equipos con doble finalidad también de detalla hardware adicional que está disponible aunque no necesariamente tiene relación con el cluster, si no que con fines adicionales del proyecto los cuales son creación de escenarios 3D mediante modelamiento en Blender y presentación de los render generados en monitor 3D. La arquitectura del laboratorio completo se encuentra distribuida como indica la Ilustración 9.



- Head Node (Servidor): Este equipo debe ser el mejor nodo dentro del cluster, el cual además de poseer mayor capacidad de procesamiento debe poseer mejor soporte para manejo de monitores por su tarjeta de vídeo. Será el encargado de manejar dos pantallas de presentación además de un monitor de control de los trabajos.
- Render Nodes (Estaciones de trabajo): Cada una de las estaciones de trabajo debe manejar su propios periféricos, los cuales son: teclado, mouse y monitor fullHD además de ejecutar la aplicación de modelado y servir de nodo de renderizado del cluster para asignarle trabajos en caso de tener recursos disponibles.
- NAS (Network Attached Storage): Directorio compartido de trabajo de los nodos
- SWITCH: Dispositivo con soporte Gigabit Ethernet capas de soportar la gran cantidad de procesos que estarán siendo enviados entre todos los nodos y el NAS.

### **6.1.1 Red del Laboratorio:**

- Router CISCO SG 30010 con 10 puertos Gigabit Ethernet será el encargado del enrutar el trafico de la red
- Cableado está realizado con cable par trenzado unifilar categoría 6
- El almacenamiento en red está situado en un dispositivo NAS (Network Attached Storage) con soporte Gigabit Ethernet y 2TB de almacenamiento.

## **6.1.2      Hardware del Head Node**

Principales características del nodo principal:

Cantidad de equipos iguales: 1 (*Servidor*)

Procesador: 2 Intel Xeon E5620 2.4Ghz, 12Mb de caché

Cantidad de Núcleos: 8 en cada procesador, con un total de 16 cores

Memoria RAM: 4 Módulos de 4GB cada uno, con un total de 16GB

Tarjeta de Video: Tarjeta Video NVIDIA QUADRO FERMI 600 LP 1GB, BLK

Tarjeta Video NVIDIA QUADRO NVS300 512MB, PCIE16X

Tarjeta de red: 2 Integrada Gigabit Ethernet

## **6.1.3      Hardware de los Render Nodes**

Principales características del los otros nodos:

Cantidad de equipos iguales: 3 (*Nodos 1,2 y 3*)

Procesador: AMD Phenom II 1090T x6 3.2GHZ, 9Mb de caché

Cantidad de Núcleos: 6

Memoria RAM: 8GB

Tarjeta de Video: NVIDIA QUADRO FERMI 2000/1GB

Tarjeta de red: 2 Integrada Gigabit Ethernet

Cantidad de equipos iguales: 1 (*Nodo 4*)

Procesador: AMD Phenom II 1090T x6 3.2GHZ, 9Mb de caché

Cantidad de Núcleos: 6

Memoria RAM: 8GB

Tarjeta de Video: NVIDIA QUADRO FERMI 600 LP 1GB

Tarjeta Video NVIDIA QUADRO NVS300 512MB, PCIE16X

Tarjeta de red: 2 Integrada Gigabit Ethernet

## **6.2 Características del Software**

### **6.2.1 Sistema Operativo (Debian)**

#### Sistema Operativo

La elección del sistema operativo implica que debemos acotar nuestra solución a la idea de tener un sistema cluster eficiente, libre y robusto, capaz de aguantar la sobrecarga de procesos, ser fácil en su manejo y además de bajo costo. El software libre brinda esta posibilidad, además de ser eficiente y colaborativo permite al usuario hacer uso legal sin escatimar en pago de licencias.

Para entender mejor que es el software libre comentaremos que es y algunas de sus ventajas.

¿Qué es el Software Libre?

El software libre es aquel que puede ser distribuido, modificado, copiado y usado; por lo tanto, debe venir acompañado del código fuente para hacer efectivas las libertades que lo caracterizan. Dentro del software libre hay, a su vez, matices que es necesario tener en cuenta. Por ejemplo, el software de dominio público significa que no está protegido por el copyright, por lo tanto, podrían generarse versiones no libres del mismo, en cambio el software libre protegido con copyleft impide a los distribuidores incluir algún tipo de restricción a las libertades propias del software así concebido, es decir, garantiza que las modificaciones seguirán siendo software libre. También es conveniente no confundir el software libre con el software gratuito, éste no cuesta nada, hecho que no lo convierte en software libre, porque no es una cuestión de precio, sino de libertad.

¿Que ventajas tiene usar Software Libre?

A continuación enumeraremos algunas de las ventajas del Software Libre.

- Es económico
- Libertad de uso y redistribución
- Independencia tecnológica
- Fomento de la libre competencia al basarse en servicios y no licencias
- Soporte y compatibilidad a largo plazo

- Formatos estándar
- Corrección más rápida y eficiente de fallos
- Métodos simples y unificados de gestión de software
- Sistema en expansión

Sin embargo también existen algunos inconvenientes, pero son los menos y que en nuestro caso no se aplica.

- Algunas aplicaciones (bajo Linux) pueden llegar a ser algo complicadas de instalar (existe la suficiente compatibilidad para el software requerido dentro del cluster).
- Inexistencia de garantía por parte del autor (en nuestro caso tenemos la posibilidad de utilizar nuestros conocimientos en caso de algún fallo).
- Poca estabilidad y flexibilidad en el campo de multimedia y juegos (no se aplica, puesto que el cluster no está destinado para esta acción).
- Menor compatibilidad con el hardware (en nuestro caso tenemos una ventaja frente a ello, el hardware utilizado posee la suficiente compatibilidad para hacer eficiente su uso).

## Elección del Sistema Operativo para el Cluster

Dentro de la amplia gama de posibilidades para optar por un sistema operativo anfitrión, nos inclinaremos hacia la rama de GNU/Linux, particularmente el sistema operativo Debian GNU/Linux. Debian es una comunidad conformada por desarrolladores y usuarios, que mantiene un sistema operativo GNU basado en software libre. El sistema se encuentra precompilado y empaquetado en un formato .deb para múltiples arquitecturas de computador y para varios núcleos.

Nació como una apuesta por separar en sus versiones el software libre del software no libre. El modelo de desarrollo del proyecto es ajeno a motivos empresariales o comerciales, siendo llevado adelante por los propios usuarios, aunque cuenta con el apoyo de varias empresas en forma de infraestructuras. Debian no vende directamente su software, lo pone a disposición de cualquiera en Internet, aunque sí permite a personas o empresas distribuirlo comercialmente mientras se respete su licencia

Algunas de las razones de por qué usar Debian GNU/Linux son:

- Está mantenida por sus usuarios.
- Gran capacidad de Soporte Técnico.
- Comunidad amplia de usuarios.
- Sistema de empaquetamiento de software útil, rápido y sencillo.
- Instalación Sencilla.
- Amplia cantidad de software disponible.
- Paquetes de Software bien integrados.
- Código fuente disponible.
- Actualizaciones periódicas.
- Sistema de seguimiento de errores.
- Estabilidad.
- Rápido y ligero en memoria.
- Estándar de controladores para hardware.
- Software de seguridad.

La versión elegida para instalar dentro del cluster será la ultima estable del sistema operativo correspondiente a Debian 6.0.3 Squeeze.

### **6.2.2      Middleware (MOSIX)**

MOSIX como Middleware a utilizar en el Cluster

¿Qué es?

Anteriormente ya comentamos que es MOSIX, sin duda al tratarse de una extensión para el kernel de Linux permite que la administración sea de forma más eficiente, ya que será ejecutada a nivel de núcleo del sistema. La principal ventaja de MOSIX es que hace parecer una granja de nodos como un computador muy grande de mucha capacidad, esto es gracias a que utiliza una interfaz de acceso, denominada SSI (Single System Image), la cual genera la sensación al usuario de que utiliza un único computador. MOSIX, además permite ejecutar aplicaciones "normales" (no paralelizadas) en un cluster. También se encarga de la "migración de procesos" y balanceo de carga como principales

características, lo que permite migrar el proceso de nodo en nodo en caso de mucha sobrecarga, este será movido hacia otro de más recursos.

A diferencia de otros tipos de Clusters, MOSIX puede ser usado en computadores de distintas características incluyendo su arquitectura, 32 o 64 bits, haciendo compatible dentro de la granja de nodos a cualquiera de estos.

Otra singular característica que permite MOSIX, es la opción de utilizar los nodos como cualquier equipo de trabajo, dando libertad de acción al uso que se desee de la maquina. Esto es gracias al hecho de ser una extensión del kernel de Linux, es posible utilizarlo como un modulo que se carga cuando se necesite. Por ejemplo; una solución de alto computo y ahorrativa de recursos seria la utilización de un laboratorio compartido para estudiantes, que desde ciertas horas, es posible utilizarlo como cluster y en otras como estaciones de trabajo.

La solución que debemos plantear tiene como requisito la utilización de software para modelado y animaciones en 3 dimensiones. Estos software por lo general no son aplicaciones paralelizadas y en algunos casos son imposibles de modificar dicha cualidad en su código ya que son de código cerrado y protegidas con una cierta licencia de uso.

### ¿Cómo Funciona?

En MOSIX, a diferencia de otros clusters, no es necesario modificar las aplicaciones ni tampoco utilizar librerías especiales. De hecho, tampoco es necesario asignar "a mano" los procesos a los diferentes nodos que componen el cluster.

La idea es que después de la creación de un nuevo proceso (fork), MOSIX intenta asignarlo al mejor nodo disponible en ese momento. MOSIX estará constantemente monitorizando los procesos, y si fuera necesario, migrará un proceso entre los nodos para maximizar el rendimiento promedio.

MOSIX realiza todo esto automáticamente, bajo el concepto de "*fork and forget*" al igual que en un sistema SMP. Esto significa que sólo algunas aplicaciones se beneficiarán de un cluster MOSIX, básicamente:

- Procesos que requieren de mucha CPU, aplicaciones científicas, de ingeniería, etc.
- Procesos paralelos, especialmente los que tienen tiempos de ejecución impredecibles.

- Clusters con nodos de diferentes velocidades y/o distintas cantidades de memoria.
- Entornos multi-usuario y de tiempo compartido.
- Sistemas escalables.

MOSIX funciona silenciosamente. Sus operaciones son transparentes para las aplicaciones. Los usuarios no necesitan saber dónde se están ejecutando los procesos, tampoco necesitan preocuparse de lo que están haciendo otros usuarios.

Como MOSIX está implementado en el kernel de Linux, sus operaciones son totalmente transparentes para las aplicaciones. Esto permite definir distintos tipos de clusters, incluso un cluster con diferentes CPU's o velocidades LAN.

Otra característica de MOSIX, es que sus algoritmos son descentralizados - esto significa que cada nodo puede ser el maestro de los procesos creados localmente, y un servidor de los procesos remotos que migraron desde otros nodos. Esto permite agregar o remover nodos desde el cluster en cualquier momento.

### **6.2.3 Blender y sus motores de renderizado**

Para la realización del proyecto Alfa III Gaviota se eligió priorizar el uso de software libre por su bajo costo (generalmente gratuito) y su adaptabilidad a las necesidades específicas del proyecto en cuestión, ya que, el tener el código fuente completo a disposición permite la modificación del mismo o la creación de un fork con adaptaciones a lo que sea que demande el proyecto. Claro está, sin dejar de considerar el trabajo que conlleva realizar tales modificaciones.

A diferencia de otras aplicaciones de modelado 3D, Blender no se limita solo al modelado, ya que contiene una gran cantidad de herramientas relacionadas. Podemos decir que Blender es un paquete 3D al que podemos darle, entre otros, los siguientes componentes o aplicaciones:

- Herramienta de modelado 3D con múltiples vistas configurables.
- Texturizado y pintado de modelos.
- Iluminación.
- Animación de modelos.

- Motor de renderizado, bajo el mismo nombre de “Blender”, además permite utilizar otros motores para los mismos modelos. Entre ellos Cycles, Povray, NetRender, Lux, Vray, entre otros.
- Creación de juegos con comportamientos programables.
- Motor de juegos, diferente al motor de renderizado normal.
- Scripting, permite la ejecución de scripts en python utilizando la biblioteca bpy la que nos da la opción de utilizar funciones de Blender.
- Compositor que permite mezclar y editar clips de video.

Además, Blender es multiplataforma y nos permite usar todas sus funcionalidades en Windows (XP, Vista, 7), Mac OS X, Linux y FreeBSD. En la ilustración 10 podemos ver la interfaz gráfica de blender para la creación de modelos 3D



Al momento de renderizar una imagen Blender hace uso de su motor de renderizado, este motor es el que se encarga de transformar modelos en base a vértices, aristas y caras en contenido presentable en cualquier monitor en un formato conocido de imagen o video. Blender incluye, sin limitarse a ellos los siguientes motores:

#### **6.2.3.1      *Blender***

El motor por defecto de Blender, puede ser ejecutado en modo gráfico desde la interfaz de blender o utilizando una terminal y trabajar con él en segundo plano. Para esto debe estar previamente seleccionado como motor en el archivo .blend que queremos renderizar.

#### **6.2.3.2      *Cycles***

Motor de renderizado, el cual hace uso de GPU y CPU mediante el uso de la tecnología CUDA de Nvidia y además con soporte experimental de OpenCL. Permite opciones de selección de la tecnología a utilizar en un panel.

Además de la ejecución en modo gráfico desde la interfaz de blender puede ser lanzado utilizando una terminal y trabajar con el en segundo plano. Para esto debe estar previamente seleccionado como motor en el archivo .blend que queremos renderizar y establecida la tecnología a utilizar.

#### **6.2.3.3      *NetRender***

Motor de renderizado que se encarga de repartir el trabajo en una red local, funciona activando al menos un nodo como servidor, el cual puede tener asociados clientes. Un tercer rol es asignado al equipo que envía la carga, el cual es el cliente.

Tiene la desventaja que debemos abrir blender en cada uno de los equipos para su uso, y iniciar el servicio de server/slave, según corresponda en cada uno de los equipos mediante su interfaz gráfica, además, el cliente nunca renderiza lo que hace que se desperdicie el poder de cálculo de un equipo.

#### **6.2.3.4      *Blender Game Engine***

Motor de renderizado (o rasterizado al tratarse de trabajo en tiempo real) el cual nos brinda una representación gráfica aceptable para ejecutar en tiempo real la programación establecida para un escenario. Este motor interactúa directamente con la API de python disponible en Blender, así como con el Blender Game Logic. Es la elección óptima para juegos y realizar simulaciones en tiempo real coherentes físicamente, para luego, recordar los movimientos realizados y renderizar a frames y vídeo de alta calidad con algún otro motor.

## 7 Implementación de la Solución

### 7.1 *Instalación de Debian*

Para emplear la solución, haremos uso del laboratorio antes mencionado, este consta de 4 nodos de alta capacidad y un servidor servidor central que se encargara de la sincronización entre nodos y este.

Para comenzar la implementación debemos disponer de una imagen de instalación del sistema operativo Debian GNU/Linux obtenida desde <http://www.debian.org/CD/netinst/>, utilizaremos la imagen de instalación por red. La elección de esta versión y no la completa es simplemente porque al tratarse de instalar mediante red solo usaremos los paquetes necesarios y así liberar al sistema de un uso ineficiente de memoria y espacio.

Anexo a lo anterior, se requiere el uso de una conexión a internet en todos los equipos que componen el cluster, para ello previamente se debe configurar un dispositivo conmutador (Router) disponible e imprescindible para la creación del cluster. El dispositivo se debe configurar con direcciones IP secuenciales y fijas. Esto es previo requisito a instalación de MOSIX.

Una vez que la imagen de instalación sea correctamente descargada y copiada aun medio (CD/DVD/USB), se dará comienzo a la instalación en los nodos y servidor. Nos encontraremos con varias etapas que se nos guiaran en la instalación de Debian GNU/Linux. Procederemos a explicar en qué consisten cada una de ellas.

## Preparación de la Instalación

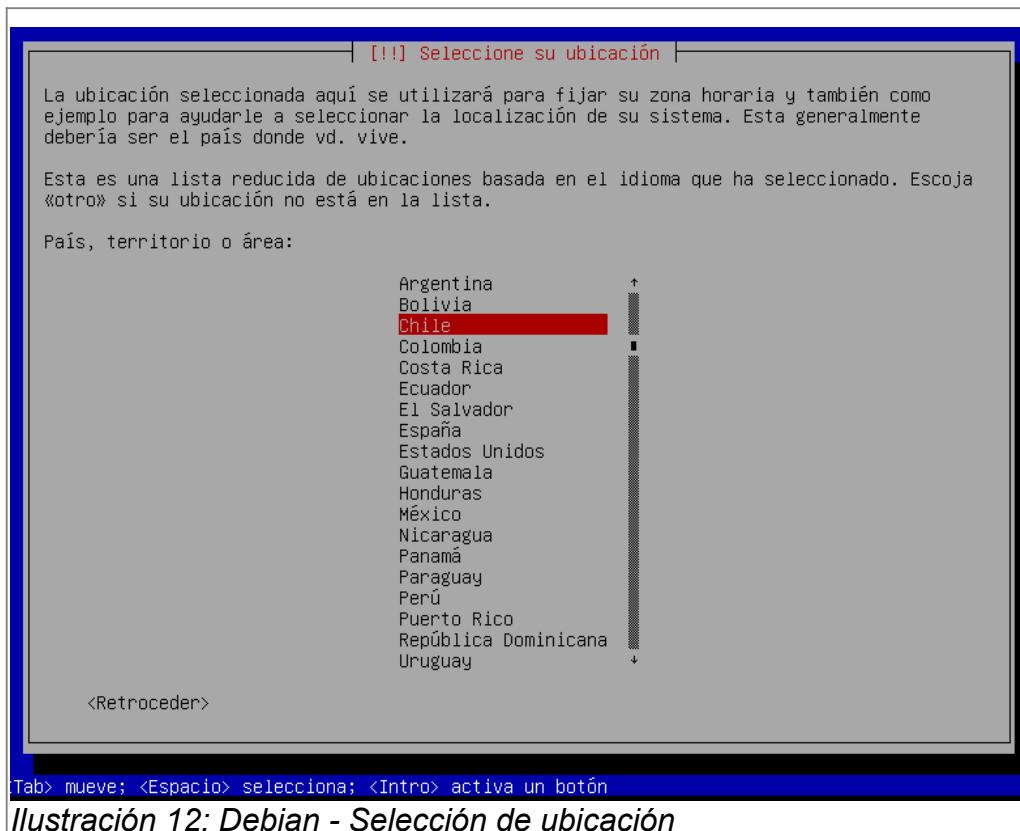
### • Selección de Lenguaje



Ilustración 11: Debian - Selección de idioma

Es la primera etapa que nos encontraremos al ejecutar la instalación de Debian GNU/Linux. Se debe escoger el idioma de la instalación dentro de una amplia variadas.

- Selección de la Ubicación

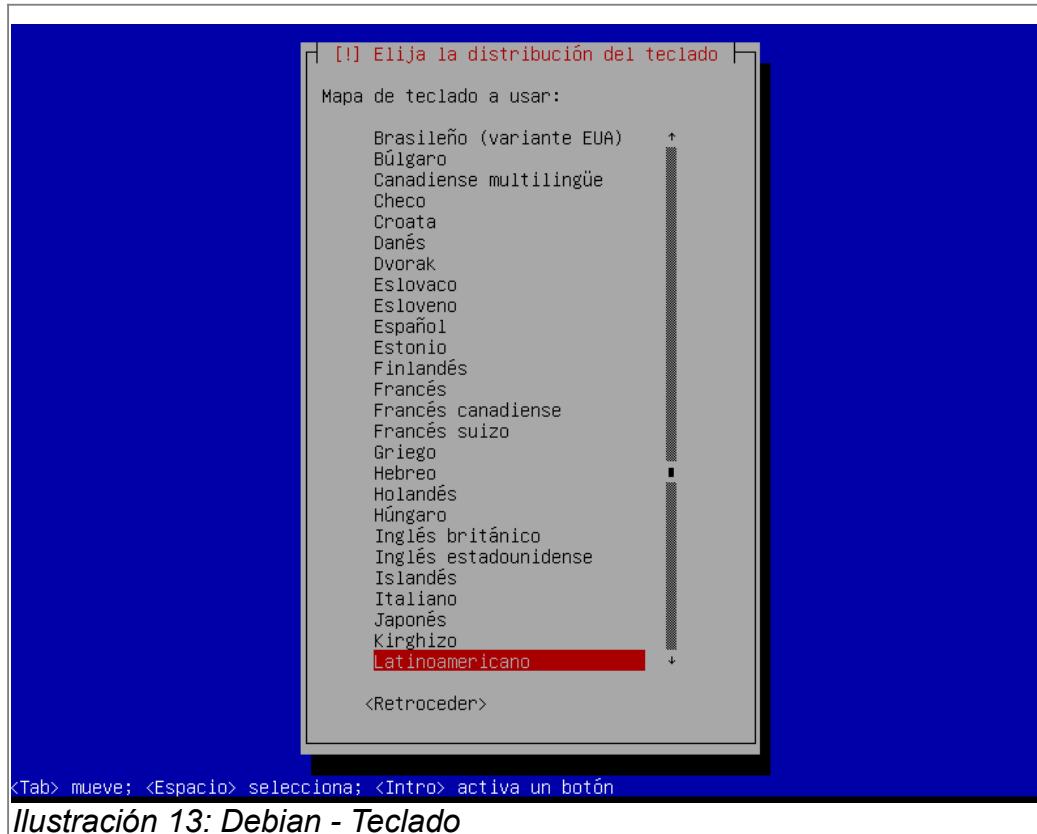


Tab > mueve; Espacio > selecciona; Intro > activa un botón

Ilustración 12: *Debian - Selección de ubicación*

Esta etapa nos indica que debemos seleccionar el país de donde estamos instalado la distribución. Es necesario para fijar la zona horaria del sistema.

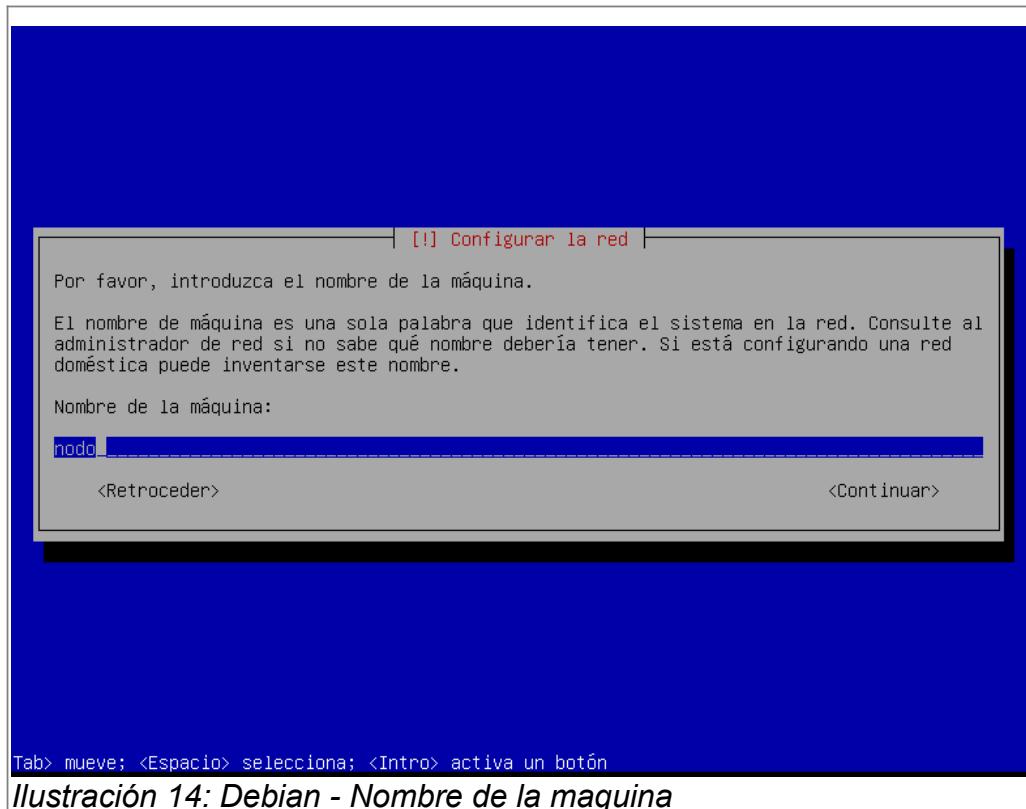
- Seleccionar Distribución de Teclado



Para una correcta instalación del sistema se debe indicar cuál es nuestra distribución de teclado que poseemos en cada uno de equipos del cluster. La no configuración de este paso podría traer algunos inconvenientes menores en el uso del dispositivo de entrada, esencialmente el uso de símbolos.

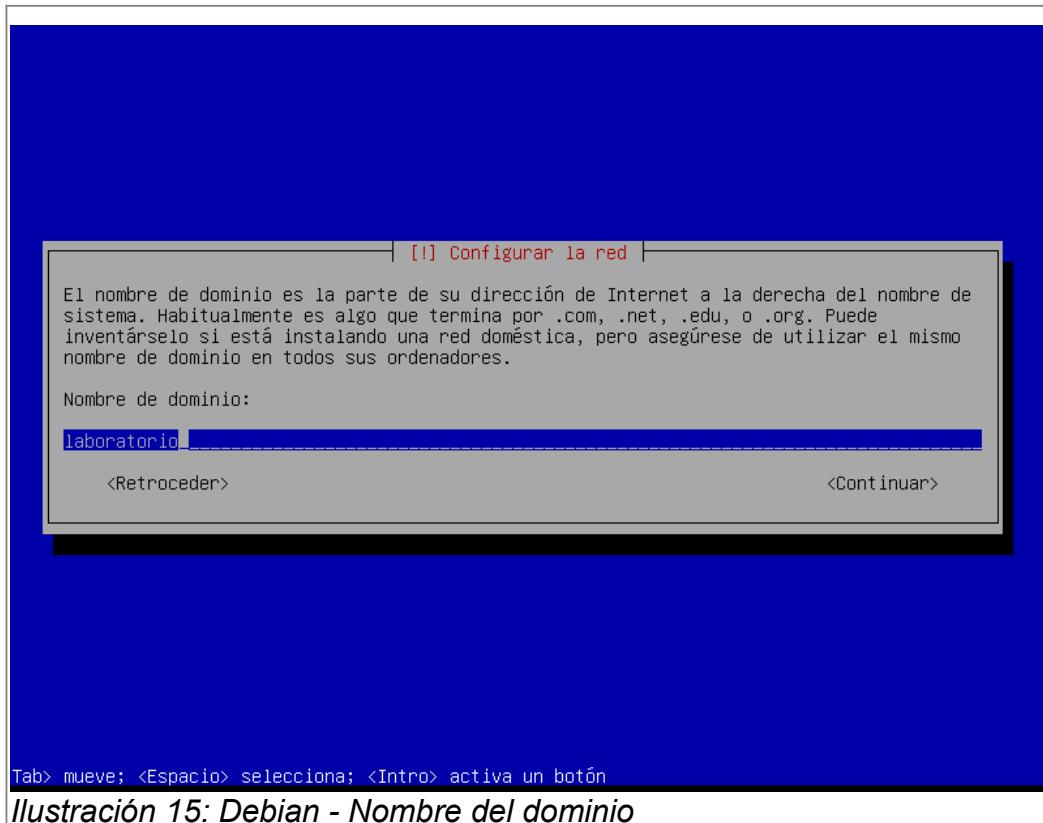
## Configuración para el dominio de la red

### •Nombre de Equipo



Instancia en donde se le debe asignar un nombre al equipo. Este corresponderá a nodo más un número o servidor.

- Dominio de Red



Se debe indicar el nombre del domino de la red. En otras palabras corresponde a el nombre de la unión de computadores. Generalmente lo toma desde la configuración previa en el dispositivo conmutador.

## Configuración de Usuarios y Contraseñas

- nombre de Usuario

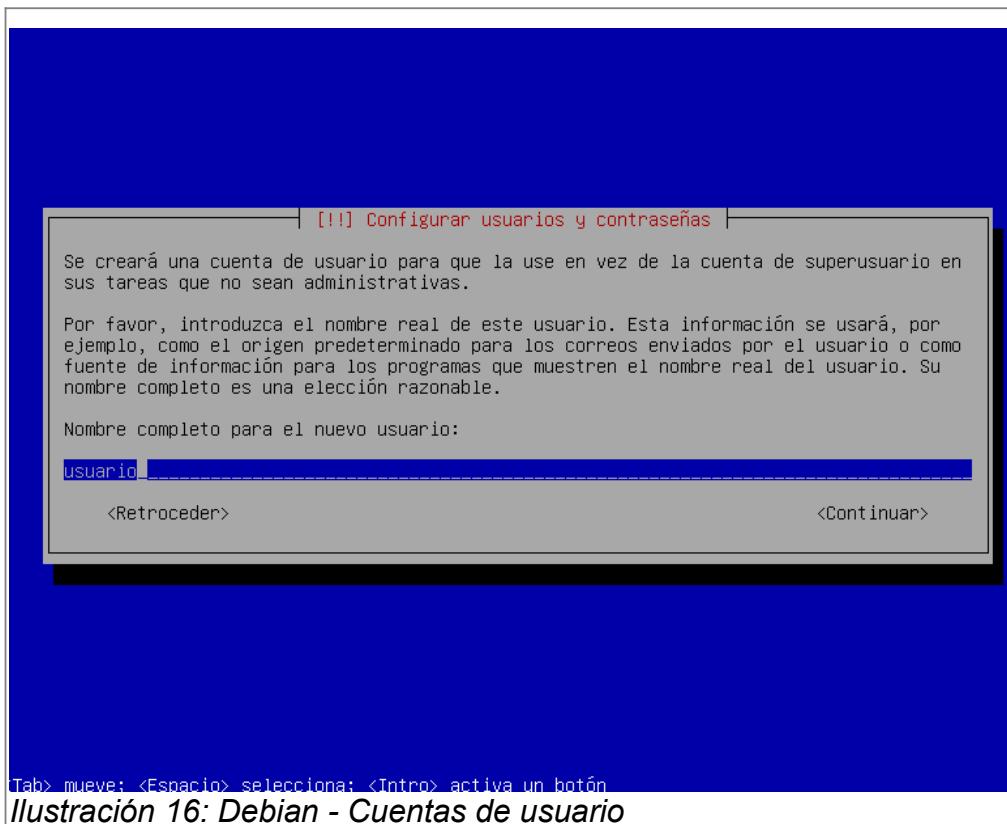
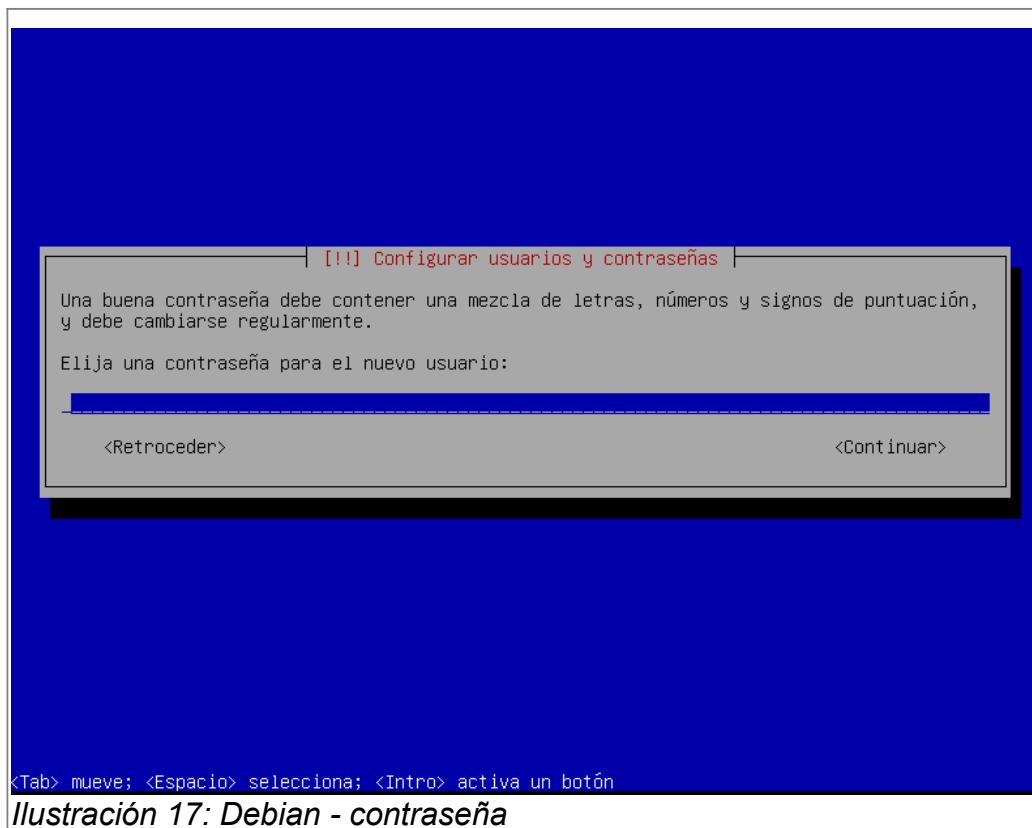


Ilustración 16: Debian - Cuentas de usuario

Se debe indicar algún nombre de usuario, el cual tendrá acceso al sistema. No es un usuario con control total del sistema instalado, sino uno con limitaciones sobre la administración. Posteriormente se le pueden agregar los privilegios necesarios para el control total, esto lo puede hacer el usuario root o superusuario. El usuario root se crea en cada instalación automáticamente.

- Contraseña de Usuario



*Ilustración 17: Debian - contraseña*

Se debe indicar una contraseña que puede estar compuesta por números, letras o puntos.

- Configuración de Zona Horaria



Se requiere para configurar hora y fecha útil para el sistema de registros.

- Particionamiento de Discos

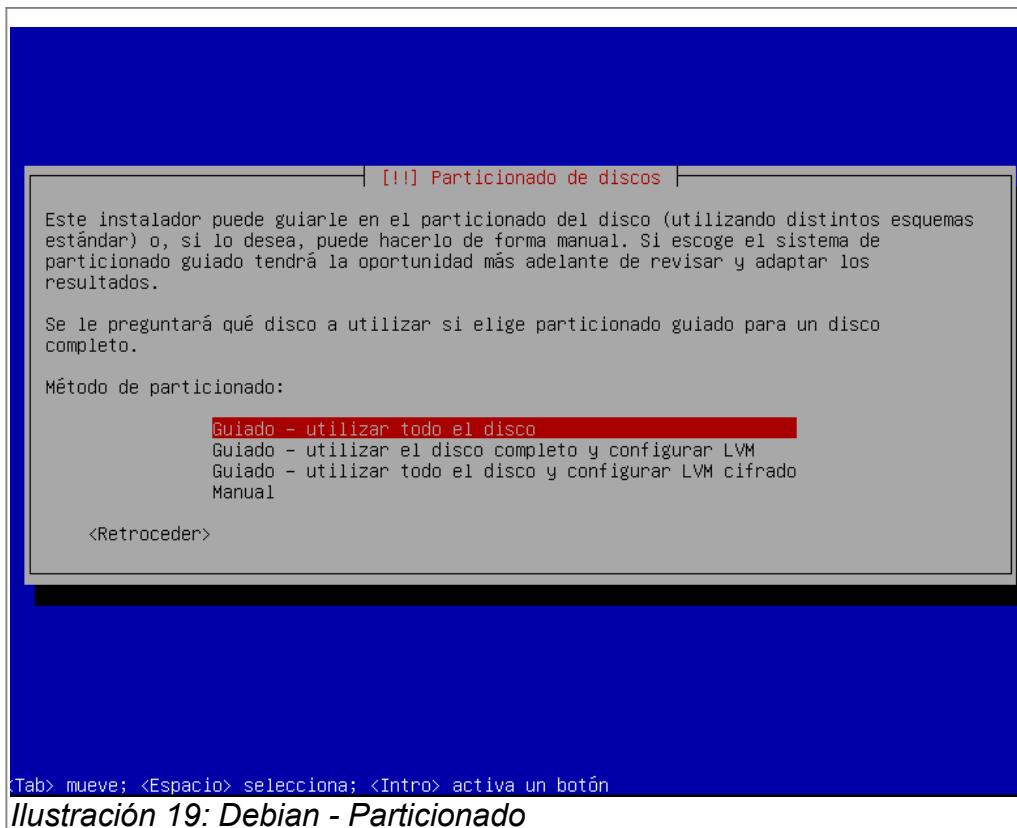


Ilustración 19: *Debian - Particionado*

El sistema se instala en el disco duro, lo que requiere la creación de una partición y sistema de archivos. Se puede usar la forma automática asistida o emplear un método manual de configuración.

## Comienza la Instalación del Sistema

Una vez ejecutada los preparativos, se procede a la instalación de los paquetes básicos.



*Ilustración 20: Debian - Instalación de paquetes*

## Configuración del Gestor de Paquetes

- Para realizar la instalación por red, se debe indicar cual servidor espejo usaremos. En nuestro caso, alguno de los servidores espejos alojados en chile.

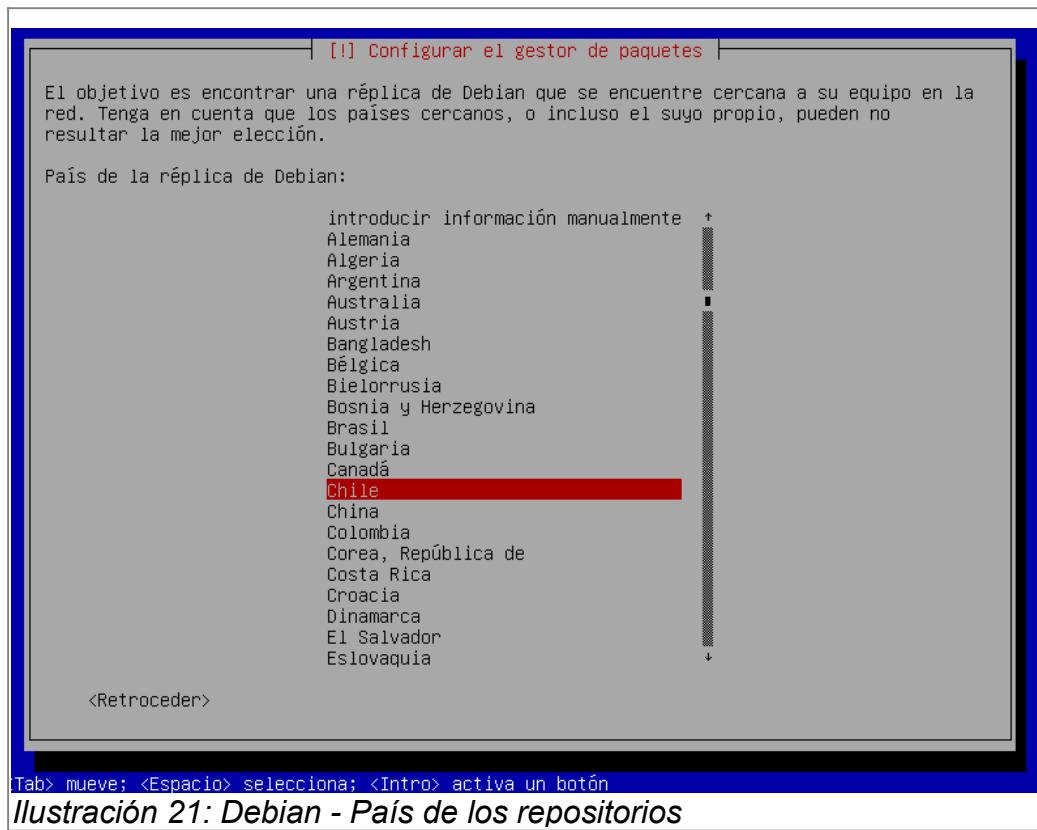


Ilustración 21: Debian - País de los repositorios

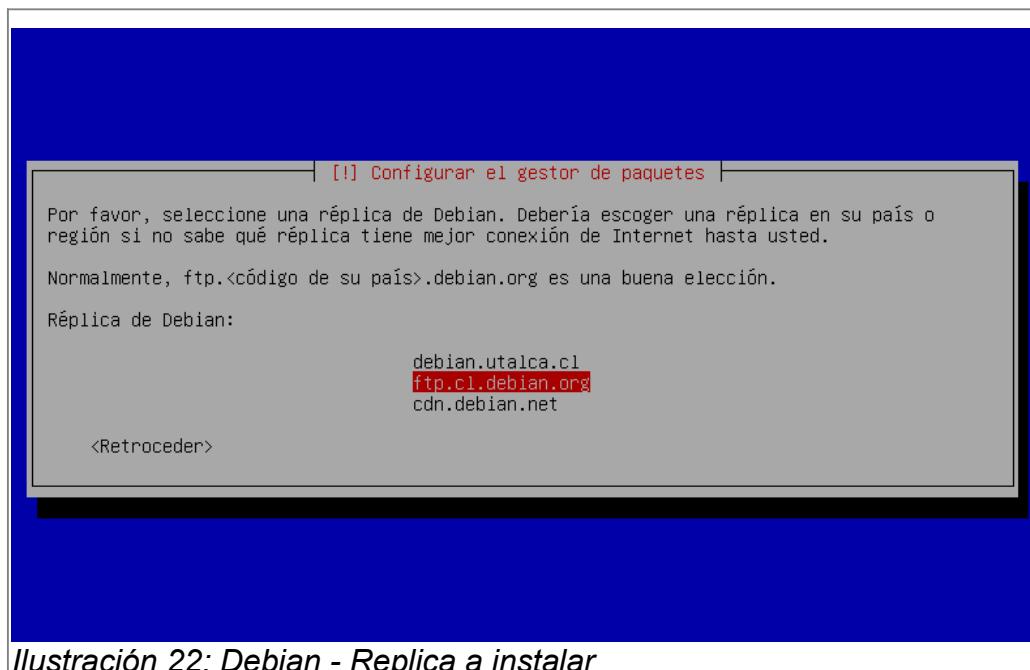


Ilustración 22: Debian - Replica a instalar

- Selección de Programas



En esta sección se decide qué conjunto de paquetes se deben instalar. Se elijen opciones para:

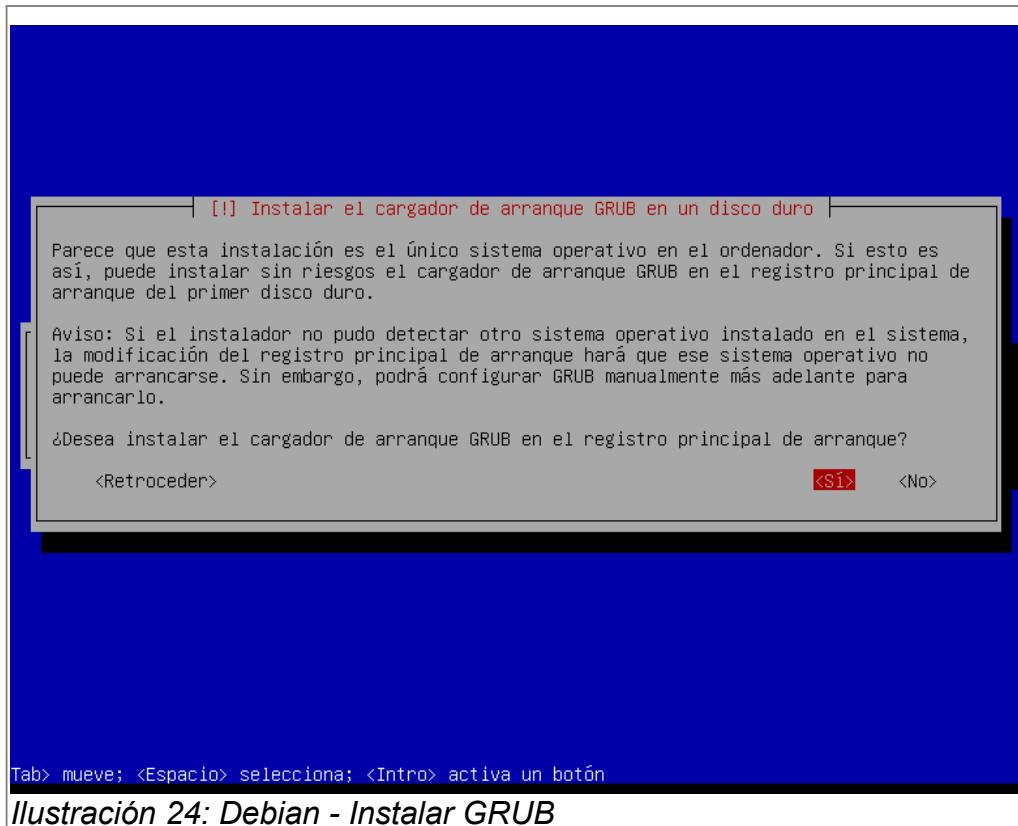
Utilidades de Sistema; Se compone de paquetes de software necesarios para su posterior uso como: aplicaciones para monitorizar el sistema, bibliotecas principales de desarrollo, aplicaciones de red, gestión de usuarios, entre otros.

Entorno de Escritorio; Se compone de aplicaciones para el uso de cada equipo como estación de trabajo, incluye bibliotecas gráficas, aplicaciones multimedia, gestor de archivos, entre otros.

Servidor de SSH; Servidor para la comunicación por medio de terminales anexas a todos los equipos del cluster, permite la administración remota.

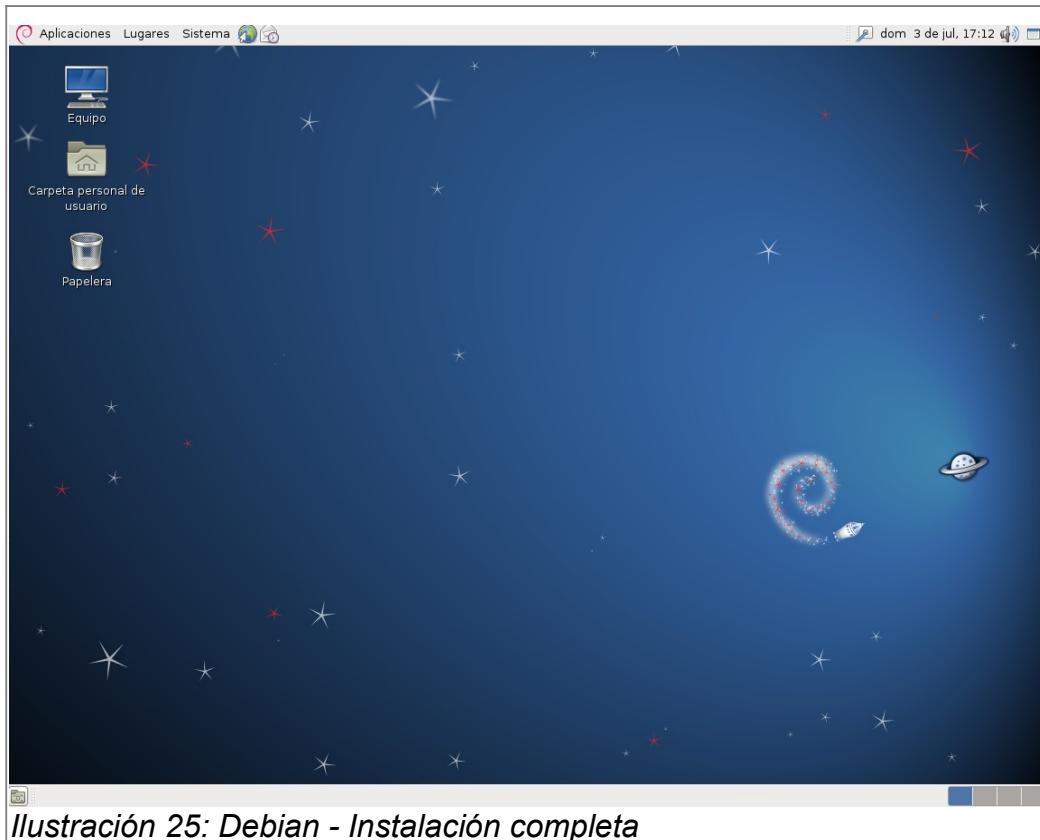
En caso de necesitar aplicaciones adicionales a la selección de paquetes, se instalarán posteriormente mediante el comando “aptitude install nombre\_aplicación”

- Cargador de Arranque del Sistema



Realizada la instalación de los paquetes seleccionados se mostrará la última etapa correspondiente a la elección de donde se quiere instalar el administrador de arranque del sistema.

Aquí finaliza la instalación del sistema operativo Debian GNU/Linux. En su primer arranque la vista del escritorio de trabajo será la siguiente.



## **7.2 Instalación y Configuración del núcleo MOSIX**

MOSIX al ser una extensión del kernel del Linux se debe incorporar a su núcleo. Esto implica recompilar el kernel actualmente instalado en los equipos del cluster, para ello debemos obtener una copia de MOSIX y posteriormente ejecutar su instalación. La instalación debe ser aplicada con permisos de “superusuario” y constara de tres pasos fundamentales, Descarga de Paquetes, Configuración del Núcleo y Compilación e Instalación.

### **7.2.1 Requerimientos para la ejecución del Cluster MOSIX**

- Todos los nodos deben utilizar Linux (cualquier combinación o distribución de esta es válida).
- Todos los nodos participantes deben estar conectados a una red que admita los protocolos de red TCP/IP y UDP/IP donde cada nodo debe tener una dirección IP única en el rango 0.1.0.0 a 255.255.254.255 que sea accesible a los demás nodos.
- Los puertos en TCP/IP, UDP/IP deben ser 249/524 y 249/250 respectivamente, estos deben estar disponibles para MOSIX y no ser utilizados por otras aplicaciones o cortafuegos.
- La arquitectura de todos los nodos pueden ser i386 (32 bits) o x86\_64 (64-bit). Los procesos que se inician en un nodo de 32 bits puede migrar a un nodo de 64 bits, pero no al revés.
- En los nodos multiprocesador (SMP), todos los procesadores deben ser de la misma velocidad.
- Los administradores de sistemas deben ser capaces de otorgar la seguridad correspondiente en el cluster y permitir la intercomunicación sin problema de los nodos.

## 7.2.2 Descarga de Paquetes

Obtendremos una copia de MOSIX desde el sitio web [http://www.mosix.org/txt\\_acadnp.html](http://www.mosix.org/txt_acadnp.html), para ello descargamos la última versión estable correspondiente a MOSIX-2.32.0.2 para el kernel 3.0.

MOSIX tiene un método de instalación automático para compilar un kernel mediante un script (mosix.install.sh), sin embargo, el método a utilizar para recomplilar sera el “debian way”, es decir a la manera que Debian reconfigura y administra la compilación de su núcleo. La razón de porque usar este método y no cualquier otro viene dado de la siguiente explicación:

¿Porqué compilar un kernel a la manera Debian?

El hecho de compilarlo a la manera Debian nos proporciona ciertas ventajas que si fuera compilarlo en su forma tradicional. Cuando se utiliza la forma “debian way” este se empaqueta automáticamente en un archivo que tiene extensión .deb, es decir, se crea un paquete. El simple hecho de que se genere un paquete con nuestro kernel compilado nos proporciona las siguientes ventajas:

- Cuando tengamos que eliminar el kernel debido a que probablemente ya tengamos uno nuevo, lo único que tenemos que utilizar es la herramienta dpkg de la siguiente forma: dpkg -r TU\_KERNEL. Donde tu\_kernel es el nombre que se le dio al paquete.
- Al poder tener el kernel en un paquete se nos hace muy fácil guardarlo en un medio USB/CD/DVD, o mejor aún, si tenemos un servidor web podríamos subirlo para que otros lo descarguen y utilicen el kernel modificado. Comúnmente el objetivo de esto es poder instalar ese mismo kernel en varias máquinas.

Por ejemplo, si se tiene muchos nodos en un cluster y se procede a actualizar el kernel a todos, sería muy absurdo compilar el kernel en para cada una de las máquinas, pero sería inteligente y astuto utilizar herramientas de Debian para poder crear un kernel genérico, el cual nos proporcionará un paquete con nuestras configuraciones o preferencias.

Para que la recomplilación del kernel se ejecute correctamente se deben instalar ciertas dependencias necesarias, estas son:

- ✓ module-assistant
- ✓ module-init-tools
- ✓ gcc
- ✓ make
- ✓ g++
- ✓ kernel-package
- ✓ initramfs-tools
- ✓ dpkg-dev
- ✓ zlib1g-dev
- ✓ libncurses5
- ✓ libncurses5-dev

Se ejecutara el siguiente comando con privilegios de superusuario

*“aptitude install module-assistant module-init-tools gcc make g++ kernel-package initramfs-tools dpkg-dev zlib1g-dev libncurses5 libncurses5-dev”*

El sistema se sincronizará automáticamente e instalará los paquetes solicitados.

### 7.2.3 Configuración del Núcleo

A continuación se descomprime el archivo bajado desde [www.mosix.org](http://www.mosix.org) y se procede a su instalación. Los comandos deben ser ejecutados con privilegios de superusuario en el siguiente orden:

1. Descomprimir el fichero:

```
tar -xvf MOSIX-2.32.0.2.for_Linux_kernel-3.0.tbz
```

2. Entrar al directorio de MOSIX:

```
cd mosix-2.32.0.2
```

3. Obtener una copia original del kernel Linux-3.0:

```
wget http://www.kernel.org/pub/linux/kernel/v3.0/linux-3.0.tar.bz2
```

4. Descomprimir el kernel:

```
tar xjf linux-3.0.tar.bz2
```

5. Entrar al directorio del kernel:

```
cd linux-3.0
```

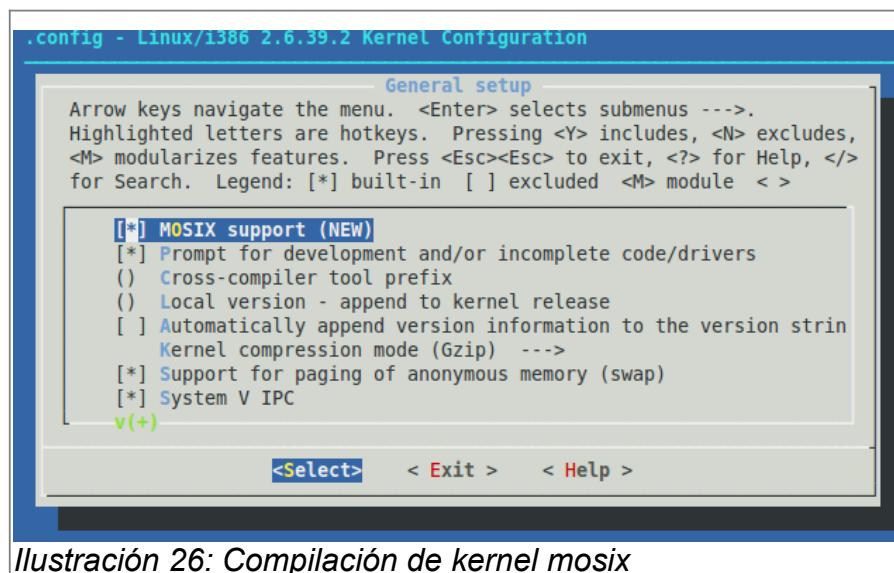
6. Aplicar el parche MOSIX al kernel:

```
patch -p1 < ../other/patch-3.0
```

7. Configurar el kernel:

```
make menuconfig
```

Con la última instrucción aparecerá la pantalla de configuración de nuestro nuevo kernel. Debemos asegurarnos de marcar la opción MOSIX support (NEW) dentro de General Setup.



## 7.2.4 Compilación e Instalación

Hasta ahora hemos logrado la parte manual de la instalación. En este instante estamos preparados para la compilación de nuestro nuevo kernel, la cual será llevada a cabo al modo “debian way”.

Ejecutamos el siguiente comando en modo superusuario:

```
“make-kpkg --initrd kernel_image kernel_headers”
```

*Este comando equivale a: make dep, make clean, make bzImage y make modules*

La opción –initrd crea una imagen initrd en el paquete que se guardará en /boot cuando instalemos el

kernel (recordar que solo estamos compilando y empaquetando, no instalando).

El comando anterior creará dos paquetes con extensión .deb en el directorio superior. Un paquete será el kernel y el otro los kernel-headers.

La opción de kernel\_headers es opcional, pero en el caso del cluster sera útil debido a que muchos programas y módulos necesitan tener los headers del kernel que se está usando para poder ser instalados, ejemplo: el driver de Nvidia, o ATI en el caso de las tarjetas gráficas.

NOTA: El tiempo de compilación es proporcional a la configuración que se haya hecho y al poder de cómputo de cada máquina.

## Instalación

Para instalar nuestro nuevo kernel solo bastará ejecutar el comando de instalación para distribuciones basadas en .deb.

Para una correcta ejecución debe ser ejecutado bajo privilegios de superusuario.

## Instalar un Paquete.

El programa dpkg es la base del sistema de gestión de paquetes de Debian GNU/Linux. Fue creado por Ian Jackson en 1993. Se utiliza para instalar, quitar, y proporcionar información sobre los paquetes .deb.

Su ejecución es sencilla. Ejemplo: instalar paquete.

***"dpkg -i nombre\_paquete"***

la opción -i corresponde a "install". También es posible aplicar otras modificaciones del paquete por ejemplo:

•-r, remover el paquete.

•-purge, elimina el paquete y todas sus configuraciones.

•-v, muestra la información del paquete.

•-configure o -reconfigure, configura un paquete que está desempaquetado pero no configurado o necesita reconfigurar.

## 7.3 Configuración MOSIX como Middleware de Administración del Cluster

Ahora nuestro nuevo kernel ya está instalado en el sistema, solo resta la instalación de MOSIX como Middleware. Anteriormente se comentó que MOSIX tiene un método de instalación automática mediante un script (`mosix.install.sh`) y cuando es ejecutado con privilegios de super usuario entrega tres posibilidades de instalación como vemos en la siguiente imagen:

NOTA: para ejecutar el script se debe ejecutar el comando “`./mosix.install`” con privilegios de super usuario.

```
Please choose how to prepare the kernel:  
1. Fetch the Linux kernel (2.6.39.2) sources from the internet.  
2. The Linux kernel (2.6.39.2) sources are already on my disk.  
3. I already installed the kernel, or will install it separately.  
  
Note for openSUSE users:  
You do not need to compile a kernel - a ready MOSIX-kernel RPM  
for openSUSE is available at http://www.mosix.org/txt\_eval.html:  
it is best to select '3' now, then download and install it.  
  
Option [1]: 3
```

Ilustración 27: `mosix.install`

La primera de ellas es la instalación completamente automática, descarga el kernel lo parcha y luego lo instala. Finaliza con la instalación del Middleware.

La segunda opción corresponde a una variación de la primera, pero se le debe indicar el directorio de donde se tiene la copia del kernel de Linux.

Por ultimo, la tercera opción, es la que nos interesa, esta nos permite solo instalar nuestro kernel por separado y Middleware.

Luego de instalar completamente MOSIX, es necesaria su configuración. El término de esto es la base para la posterior solución utilizando GPU's.

Para configurar un set de nodos solo basta utilizar la herramienta “`mosconf`”. Un administrador de MOSIX se ejecutará y mediante un menú se puede agregar, modificar o eliminar las políticas del cluster. El script “`mosconf`” permite llevar a cabo varias configuraciones paso a paso. Usualmente es

llamado la primera vez que se ejecuta "mosix.install". Esta acción será necesaria en cada uno de los equipos. También es posible hacer lo anterior mediante el método manual, lo que significa modificar los archivos principales de MOSIX, **mosip**, **mosix.map**, **userview.map**.

### 7.3.1 Descripción de Archivos de MOSIX

Están ubicados en el directorio /etc/mosix. Al momento de la instalación de MOSIX, los archivos se instalan en el directorio con su contenido vacío.

Si se prefiere editar los archivos manualmente, el formato de todos los archivos de configuración se encuentra en la sección "CONFIGURACION" del manual de mosix (man mosix) y al menos deben estar presentes los siguiente archivos:

/etc/mosix/mosix.map (nodos que están en el cluster)  
/etc/mosix/secret (configuración de seguridad y autentificación entre nodos)  
/etc/mosix/mosip (nuestra IP - si no esta clara utilizar comando "ifconfig")  
/etc/mosix/userview.map (para usar números de nodo en vez de IPs)

A continuación de detalla el contenido de estos archivos.

*/etc/mosix/mosip*  
*contiene solo la ip del nodo, esta ip identifica al nodo.*

*Ej:*

192.168.1.101

*/etc/mosix/mosix.map*  
*contiene la lista de nodos que componen el cluster.*

*Ej:*

192.168.1.101 1 p  
192.168.1.102 1 p  
192.168.1.103 1 p  
192.168.1.104 1 p  
192.168.1.105 1 p

/etc/mosix/userview.map

contiene la lista de nodos que componen el cluster pero utilizando identificaciones distintas.

Ej:

1 192.168.1.101 1

2 192.168.1.102 1

3 192.168.1.103 1

4 192.168.1.104 1

5 192.168.1.105 1

/etc/mosix/secret

contiene la contraseña que hace uso mosix para la comunicación entre sus procesos y migraciones. Ej:

passwd.mosix

estos archivos son esenciales para el trabajo normal del cluster. Una vez configurados correctamente y reiniciado el servicio podemos hacer uso de la herramienta “*mon*” para ver en consola cada nodo y su carga.

Si planea tener una grupo de multi-cluster, también debe configurar:

/etc/mosix/partners/\* (compañeros del grupo)

Si planea usar la característica de lotes, también debe configurar:

/etc/mosix/ecsecret (autentificación para lotes de trabajo del cliente - chmod 600!)

/etc/mosix/essecret (autentificación para lotes de trabajo del servidor - chmod 600!)

Los siguientes archivos de configuración son opcionales, pero también se usan normalmente:

/etc/mosix/queue.conf (políticas de encolamiento)

/etc/mosix/private.conf (espacio de almacenamiento para archivos temporales privados)

/etc/mosix/retainpri (atraso de procesos con menor prioridad)

Los siguientes archivos de configuración no se suelen utilizar:

/etc/mosix/freeze.conf (política de congelamiento)

/etc/mosix/maxguests (numero limite de huéspedes desde el grupo de clusters)  
/etc/mosix/tunes/{more\_stuff} (biblioteca de las medidas de la topología)  
/etc/mosix/newtune (tu topología)  
/etc/mosix/speed (para forzar una velocidad del procesador distinta de la estándar)  
/etc/mosix/myfeatures (características topológicas de este nodo)

#### Iniciando MOSIX:

Para iniciar MOSIX debe ejecutar "mosd" (es lo que hace "/etc/init.d/mosix start")Están ubicados en el directorio /etc/mosix. Al momento de la instalación de MOSIX, los archivos se instalan en el directorio con su contenido vacío.

Si se prefiere editar los archivos manualmente, el formato de todos los archivos de configuración se encuentra en la sección "CONFIGURACION" del manual de mosix (man mosix) y al menos deben estar presentes los siguientes archivos:

/etc/mosix/mosix.map (nodos que están en el cluster)  
/etc/mosix/secret (configuración de seguridad y autentificación entre nodos)  
/etc/mosix/mosip (nuestra IP - si no está clara utilizar comando "ifconfig")  
/etc/mosix/userview.map (para usar números de nodo en vez de IPs)

A continuación de detalla el contenido de estos archivos.

#### */etc/mosix/mosip*

*Contiene solo la ip del nodo, esta ip identifica al nodo.*

*Ej:*

**192.168.1.101**

#### */etc/mosix/mosix.map*

*Contiene la lista de nodos que componen el cluster.*

Ej:

```
192.168.1.101 1 p  
192.168.1.102 1 p  
192.168.1.103 1 p  
192.168.1.104 1 p  
192.168.1.105 1 p
```

/etc/mosix/userview.map

Contiene la lista de nodos que componen el cluster pero utilizando identificaciones distintas.

Ej:

```
1 192.168.1.101 1  
2 192.168.1.102 1  
3 192.168.1.103 1  
4 192.168.1.104 1  
5 192.168.1.105 1
```

/etc/mosix/secret

Contiene la contraseña que hace uso mosix para la comunicación entre sus procesos y migraciones.Ej:

**passwd.mosix**

estos archivos son esenciales para el trabajo normal del cluster. Una vez configurados correctamente y reiniciado el servicio podemos hacer uso de la herramienta “*mon*” para ver en consola cada nodo y su carga.

Si planea tener un grupo de multi-cluster, también debe configurar:

/etc/mosix/partners/\* (compañeros del grupo)

Si planea usar la característica de lotes, también debe configurar:

/etc/mosix/ecsecret (autentificación para lotes de trabajo del cliente - chmod 600!)

/etc/mosix/essecret (autentificación para lotes de trabajo del servidor - chmod 600!)

Los siguientes archivos de configuración son opcionales, pero también se usan normalmente:

/etc/mosix/queue.conf (políticas de encolamiento)  
/etc/mosix/private.conf (espacio de almacenamiento para archivos temporales privados)  
/etc/mosix/retainpri (atraso de procesos con menor prioridad)

Los siguientes archivos de configuración no se suelen utilizar:

/etc/mosix/freeze.conf (política de congelamiento)  
/etc/mosix/maxguests (numero limite de huéspedes desde el grupo de clusters)  
/etc/mosix/tunes/{more\_stuff} (biblioteca de las medidas de la topología)  
/etc/mosix/newtune (tu topología)  
/etc/mosix/speed (para forzar una velocidad del procesador distinta de la estándar)  
/etc/mosix/myfeatures (características topológicas de este nodo)

Iniciando MOSIX:

Para iniciar MOSIX debe ejecutar "mosd" (es lo que hace "/etc/init.d/mosix start")

### 7.3.2 Administración básica de MOSIX

Para la administración básica de nuestro cluster MOSIX podemos hacer uso de las siguientes aplicaciones que provee MOSIX.

mon / mmon – Ver que esta sucediendo

-mon y mmon son dos monitores para ver el estado de los recursos en cada uno de los cluster y en todos los cluster a la vez.

mosrun – Ejecutando procesos en MOSIX

-Para ejecutar una aplicación en MOSIX la orden debe comenzar con mosrun.

–Este tipo de programas pueden migrar a otros nodos

–Ejemplo: > mosrun myprog 1 2 3 (ejecuta myprog con los argumentos 1 2 3)

-Los programas que no son iniciados con mosrun se ejecutan de forma nativa en Linux y no pueden migrar a otros nodos.

## mosps – Ver los procesos de MOSIX

mosps (como ps) provee información acerca de los procesos MOSIX (y posee varias de las cualidades de ps)

## Mosq – Ver y controlar la cola

permite ver y controlar los procesos corriendo en el cluster

## migrate – Controlar los procesos de MOSIX

Permite controlar la migración de procesos MOSIX  
para más información ver apéndice **Herramientas de MOSIX**.

## **7.4 Instalación del soporte CUDA y OpenCL**

En nuestro caso particular de configuración de cluster, debemos tener los equipos habilitados tanto para desarrollo como ejecución de aplicaciones, ya que estos funcionan como estaciones de trabajo además de clusters, por lo que necesitamos instalar los drivers de desarrollo de Nvidia que son los que cubren todas nuestras necesidades.

De preferencia se debe instalar la última versión de los drivers para el modelo de tarjeta gráfica correspondiente, esto es debido a que generalmente, el driver mas reciente tiene soporte cada vez mejor para las tecnologías de procesamiento con GPU.

### Paso I: Desactivar Nouveau

Nouveau es un proyecto, cuya meta es entregar drivers libres y gratuitos para las tarjetas gráficas Nvidia, desafortunadamente no soporta la mayor parte de las tecnologías que si podemos encontrar habilitadas al utilizar los drivers propietarios de Nvidia, por lo que en este caso serán desactivados y reemplazados.

Primero editaremos el siguiente archivo:

```
vim /etc/modprobe.d/blacklist.conf
```

Agregaremos las siguientes líneas:

```
blacklist nouveau  
options nouveau modeset=0
```

Estas deshabilitan el driver en debian.

Luego creamos el siguiente archivo:

```
vim /etc/modprobe.d/nvidia-installer-disable-nouveau.conf
```

Agregaremos las mismas lineas:

```
blacklist nouveau  
options nouveau modeset=0
```

Este proceso debe ser realizado en los 2 archivos indicados, ya que la segunda es la forma requerida por el instalador del driver de Nvidia.

Finalmente reiniciamos el equipo e iniciamos sin entorno grafico X o lo detenemos mediante:

```
/etc/init.d/gdm3 stop (reemplazar gdm3 con kdm, gdm, xdm segun corresponda)
```

Paso 2: Instalar devdriver

Antes de iniciar la instalación del driver de desarrollo, el primer paso es iniciar sesión como usuario root:

```
sudo su -l
```

Nos movemos al directorio donde está ubicado el driver, en nuestro caso:

```
cd ~/Descargas/opencl/
```

Ejecutamos el instalador del driver lo hacemos mediante sh

```
sh ./devdriver_4.1_linux_64_285.05.15.run
```

Una vez terminada la instalación correctamente reiniciamos  
reboot

Paso 3: Instalar Cuda Toolkit

Iniciar sesión como usuario root:

```
sudo su -l
```

Nos ubicaremos en el directorio donde descargamos el instalador de cudatoolkit\_(version)\_(arch)\_(distro).run, en nuestro caso lo hacemos mediante el comando cd de la siguiente forma:

```
cd ~/Descargas/opencl/
```

Para ejecutar el instalador lo hacemos mediante sh

```
sh ./cudatoolkit_4.1.15_linux_64_ubuntu_11.run
```

El instalador luego de verificar la integridad del archivo y extraer su contenido nos preguntara la ruta que deseamos utilizar, utilizaremos /usr/local/cuda la cual es la ruta predeterminada presionando [ENTER]

En caso de existir una versión anterior la reemplazamos [ENTER]

El instalador nos indicara que incluyamos esta implementación de cuda en LD\_LIBRARY\_PATH pero NO debemos hacerlo aun, solo esperamos que el instalador nos diga que debemos reiniciar y lo hacemos.

Paso 4: Instalar GpuComputingSDK

Iniciar sesión como usuario root:

```
sudo su -l
```

Moverse al directorio donde está ubicado el instalador gpucomputingsdk\_4.1.15\_linux.run así:

```
cd ~/Descargas/opencl/
```

Ejecutaremos el instalador con sh

```
sh ./gpucomputingsdk_4.1.15_linux.run
```

El instalador luego de verificar la integridad del archivo y extraer su contenido nos preguntara la ruta que deseamos utilizar, utilizaremos ~/NVIDIA\_GPU\_Computing\_SDK la cual es la ruta predeterminada presionando [ENTER]

Nos pide la ubicación de cuda, ya que utilizamos la ubicación predeterminada (/usr/local/cuda) damos [ENTER]

Hasta aquí hemos completado correctamente la instalación, solo debemos agregar la ruta del directorio CUDA a LD\_LIBRARY\_PATH, lo hacemos así:

```
export LD_LIBRARY_PATH=/usr/local/cuda/lib64
```

## 7.5 *Instalación de MOSIX-VCL*

El estándar OpenCL permite a ciertas aplicaciones programadas para propósitos GPGPU poder acelerar el cálculo de estas utilizando la GPU como medio de procesar datos. Sin embargo el número de tales dispositivos de aceleración se ve limitado por la capacidad de conexión que tenga un nodo, por lo general 1-4 dispositivos por equipo.

MOSIX-VCL es una plataforma OpenCL, que extiende el acceso a la GPU (y otros dispositivos OpenCL) más allá de los del nodo local.

Los usuarios de MOSIX-VCL pueden ejecutar sus programas en conjunto de otros nodos, ampliando la capacidad de cálculo como si se tratase de un cluster MOSIX. La ventaja de

MOSIX-VCL reside en que no es estrictamente necesario poseer un cluster para aprovechar el poder de procesamiento de las GPU, esto hace que cada maquina o nodo se comporte como un sistema independiente y eventualmente si es que se desea, es posible fusionarlo con otros nodos y potenciar un cluster para propósitos generales.

### **7.5.1 Requerimientos MOSIX-VCL**

Para tener un correcto funcionamiento de MOSIX-VCL se debe tener presente lo siguiente:

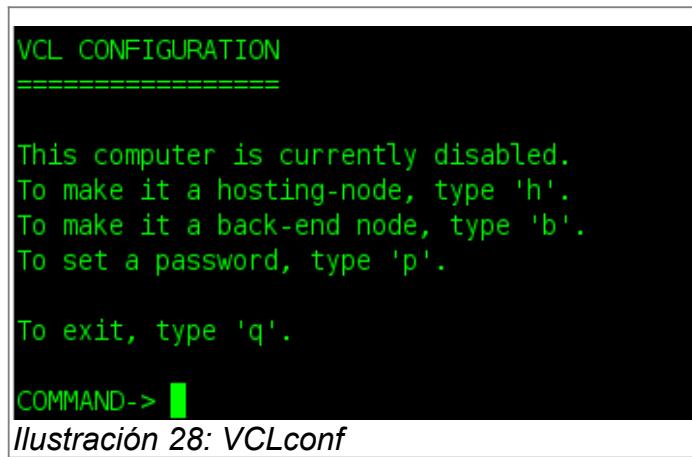
- Todos los equipos o nodos participantes deben ejecutar Linux con la arquitectura de x86\_64(64 bits).
- Todos los nodos participantes deben estar conectados a una red que admita los protocolos de red TCP/IP.
- EL puerto TCP / IP utilizado sera el 255 y debe ser reservado para VCL (no utilizado por otras aplicaciones o bloqueado por un cortafuegos).
- Los nodos Back-End deben tener instalado la versión OpenCL 1.1, sin embargo no todos los nodos están obligados a tener el mismo hardware o versión de OpenCL.

## 7.5.2 Instalando MOSIX-VCL

Obtendremos una copia de MOSIX-VCL desde el sitio web [http://www.mosix.org/txt\\_vcl.html](http://www.mosix.org/txt_vcl.html), para ello descargamos la última versión estable correspondiente a VCL.1.13

MOSIX-VCL tiene un método de instalación automático en donde mediante el script de instalación llamado “**vcl.install**”.

Una vez ejecutado el script podemos hacer uso de la herramienta de configuración “**vclconf**” y configurarnos el nodo ya sea como Back-End o Hosting.



```
VCL CONFIGURATION
=====
This computer is currently disabled.
To make it a hosting-node, type 'h'.
To make it a back-end node, type 'b'.
To set a password, type 'p'.
To exit, type 'q'.
COMMAND->
```

Ilustración 28: VCLconf

NOTA: para ejecutar el script se debe ejecutar el comando “**./vcl.install**” con privilegios de super usuario.

Por último para correr una aplicación con MOSIX-VCL se hace de manera muy similar a MOSIX. Para ello hacemos uso del comando vclrun y pasamos como parámetro la aplicación.

Ej:

vclrun [opcion] {myprog} [argumentos] (ejecuta myprog con los argumentos )

### **7.5.3 Ejecución de MOSIX-VCL en sistema Debian**

La versión de MOSIX-VCL solo dispone de soporte para distribuciones basadas en paquetes rpm, no así como la implementación que hemos desarrollado utilizando Debian. Este fue un problema que se debió solucionar y que tiene relación con las llamadas a sistema startproc, killproc y checkproc.

Como se comentó anteriormente estas llamadas solo funcionan en sistemas rpm por lo cual se investigó y modifíco el script de inicio de VCL que se encarga de levantar la aplicación y mantener corriendo el soporte para las aplicaciones GPGPU de la siguiente manera.

Para las llamadas se hace uso del demonio propio de sistemas basados en paquetes apt. start-stop-daemon, este demonio se encarga de hacer las funciones que desarrollada starptoc y killproc.

Ejemplo de uso: Matar un Proceso

Original: killproc /sbin/broker.old

Modificado: /sbin/start-stop-daemon --stop --exec /sbin/broker

Ejemplo de uso: Iniciar un Proceso

Original: startproc /sbin/opencld

Modificado: /sbin/start-stop-daemon --start --exec /sbin/opencld

Con esto conseguimos realizar la misma funcionalidad de starproc y killproc.

También para reemplazar a checkproc se debió hacer uso de las init-functions de Debian.

Se modifíco el inicio del script con las siguientes líneas:

```
PATH=/sbin:/usr/sbin:/usr/local/sbin:/bin:/usr/bin:/usr/local/bin
```

```
#. /etc/rc.status
```

```
. /lib/lsb/init-functions
```

```
[ -x /etc/init.d/functions ] && . /etc/init.d/functions
```

posteriormente remplazando checkproc por status\_of\_proc

Original: checkproc /sbin/opencld.old

Modificado: status\_of\_proc /sbin/opencld

De esta manera mantenemos la funcionalidad del script de inicio de VCL haciendo compatible con cualquier distribución basada en Debian.

Para más información de la modificación del script de inicio revisar el apéndice “**Modificaciones VCL**”

## **7.6 Configuración adicional de los nodos**

Para mantener un standard que permita que las aplicaciones funcionen sin ningún problema se creó un script para la automatización de un nodo extra en el cluster. Este tiene por función modificar la configuración del nodoX lo que permite dejar todo correctamente instalado.

Algunas de las modificaciones al nodoX son:

- Agregar el nodo a la lista de usuarios sudo.
- Agregar usuario blender a los nodos (necesario para el renderizado por GPU).
- Instalar paquetes de programas.
- Añadir directorios compartidos (necesario para correr ciertos procesos MOSIX).
- Modificación de interfaz de red (necesario para que MOSIX funcione).
- Replicar configuraciones MOSIX.

Para más información revisar apéndice “**Script de Automatización Nodo**”

## **7.7 Instalación de monitor web - *Ganglia***

Ganglia es un monitor, el cual nos permitirá, mediante un servicio web corriendo en apache, ver la carga y rendimiento de todos los nodos en el cluster

### **7.7.1 Instalación en el Server**

En nuestro caso, el nodo host y el headnode son el mismo, por lo tanto necesitamos instalar los siguientes paquetes:

```
sudo aptitude install rrdtool librrds-perl librrd2-dev gmetad ganglia-monitor web-frontend
```

El paquete web-frontend está disponible hace poco tiempo y dependiendo de los repositorios que se estén utilizando puede ser necesario compilarlo desde el source. En nuestro caso no fue necesario, pero aun así se debe descargar el source para obtener los archivos del servicio web:

```
wget http://downloads.sourceforge.net/ganglia/ganglia-3.0.7.tar.gz?modtime=1204128965&big_mirror=0
```

Descomprimimos el contenido de la descarga:

```
tar xvzf ganglia*.tar.gz
```

Creamos el directorio para copiar todo el contenido web:

```
mkdir /var/www/ganglia
```

Finalmente, copiamos el contenido desde la carpeta “web” de ganglia web-frontend:

```
cp web/* /var/www/ganglia
```

## 7.7.2 Configuración de Apache2

Editamos el archivo /etc/apache2/sites-enabled/000-default para direccionar el servicio de apache a ganglia:

```
vim /etc/apache2/sites-enabled/000-default
```

y lo dejamos de la siguiente forma:

```
<Directory /var/www/>
    Options FollowSymLinks MultiViews
    AllowOverride None
    Order allow,deny
    allow from all
    RedirectMatch ^/$ /ganglia/
</Directory>
```

Luego reiniciamos el servicio apache2:

```
sudo /etc/init.d/apache2 restart
```

## 7.7.3 Instalación en los Nodos

La instalación en el resto de los nodos es bastante más simple, y se limita a instalar el ganglia monitor:

```
sudo aptitude install ganglia-monitor
```

A pesar de que el monitor ganglia es excelente por la opción que nos da de monitorear remotamente los equipos, en la realización de las pruebas se utilizara la herramienta clásica “htop”, así como los resultados de esta herramienta para analizar en tiempo real lo mas actualizado posible. Ganglia solo se utilizará para monitoreo remoto y por su facilidad para el análisis de los datos consolidado en un solo lugar.

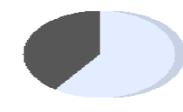
Finalmente Ganglia debe verse como se ve en la Ilustración 29.



Cluster Report for Fri, 20 Jan 2012 09:32:34 -0300

Metric  load\_one  Last  week  Sorted  descending >Grid > unspecified >  -Choose a Node >

## Cluster Load Percentages



## Overview of unspecified

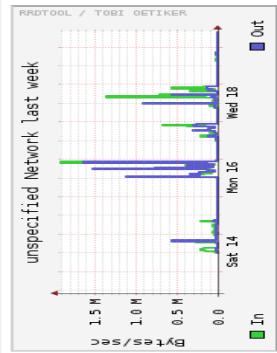
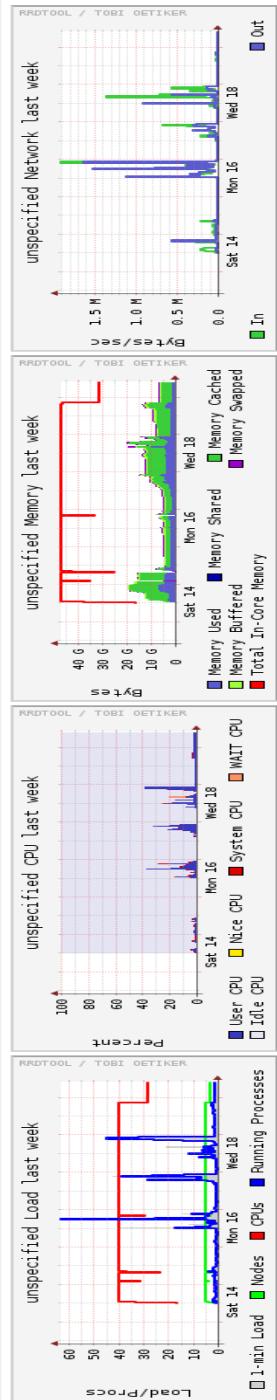
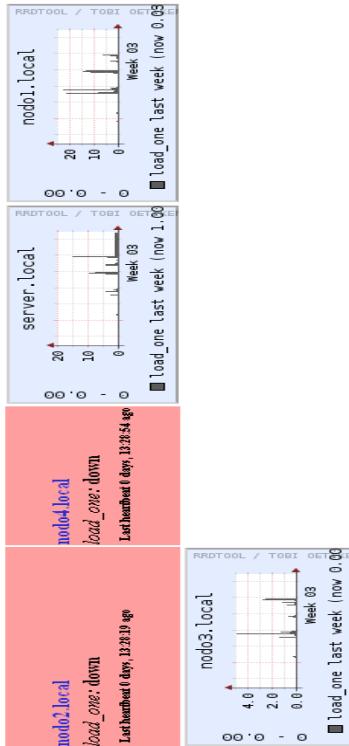
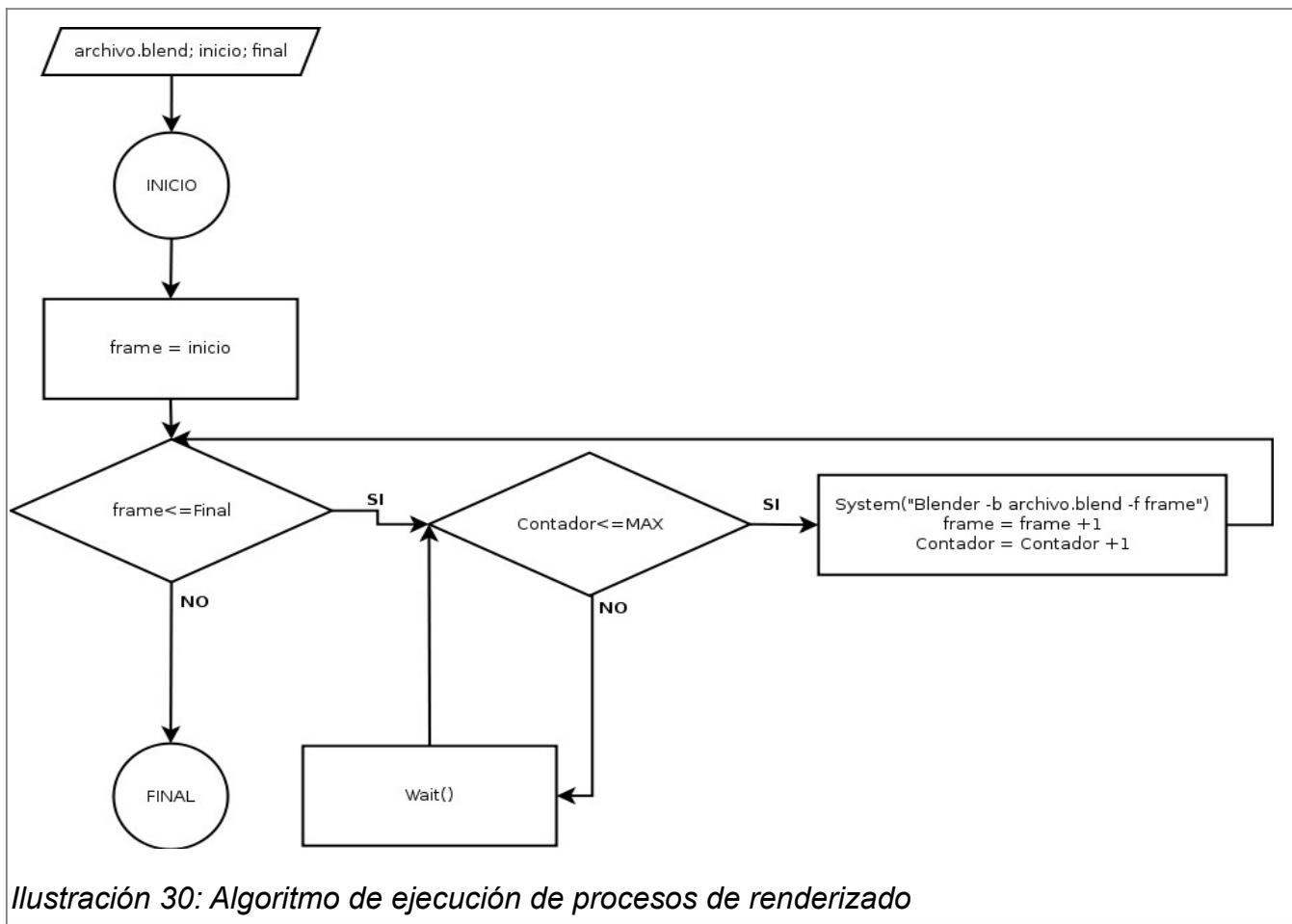
Show Hosts yes  no  unspecified load\_one last week sorted descending | Columns 4 ▶

Ilustración 29: Monitor web Ganglia

## 8 Aplicación de Renderizado (BlenderBG)

Para el funcionamiento óptimo del cluster y la utilización completa de sus recursos, no fue suficiente solo con la instalación de MOSIX, además de las implementaciones de CUDA y OpenCL. Fue necesaria la creación de una aplicación de creación de procesos hijos, los cual fue probada y configurada para su funcionamiento óptimo en el cluster.

El proceso funciona según se muestra en el diagrama de la Ilustración 30.



El funcionamiento general de la misma consiste en recibir como parámetros los frames que indican el comienzo y el fin de la aplicación. Para empezar se le asigna el valor del frame de inicio a la variable frame, esta se irá aumentando cada vez que se renderize un frame exitosamente, por lo que al ser mayor que el numero del frame final la aplicación finaliza.

Cuando el valor de frame aún es menor que el número del frame final, quiere decir que aún quedan

frames por renderizar, por lo que se procede a preguntar si aun no se ha pasado el máximo de memoria disponible. En caso de ser así se renderiza el frame actual, si no, se espera hasta que estén disponibles los recursos.

## 9 Benchmarks

Para conocer y medir en detalle el rendimiento del sistema en situaciones específicas se realizaron benchmarks (comparativas) funcionales del cluster, enfocadas primeramente en el objetivo para el cual fue adquirido y configurado: Ser utilizado como renderfarm para animaciones 3D creadas en Blender.

En todo momento se mantuvo la misma configuración de hardware y software que se ha indicado a lo largo de este documento.

La plataforma de pruebas está dividida de la siguiente forma:

Combinación de equipos	Pruebas Prácticas			
	Render CPU – Blender {2}	Render CPU – Blender BG {2}	Render GPU – Cycles {2}	RenderGPU – Cycles BG {2}
1 Nodo	✓	✓	✓	✓
Head Node	✓	✓	✓	✓
Cluster Completo	{1}	✓	{1}	✓

Tabla 4: Pruebas Prácticas

- ✓ Corresponde a las pruebas finalizadas exitosamente.

{1} Estas pruebas fueron omitidas ya que no es posible paralelizar la interfaz gráfica de Blender utilizando MOSIX, en la práctica no representa ningún problema funcional.

{2} Cada una de estas fue realizada con 3 modelos de tamaños distintos, con la finalidad de notar el cambio de rendimiento dependiendo de la memoria utilizada, por lo que se genera información de la siguiente forma:

Combinación de equipos	Prueba N		
	Modelo 1	Modelo 2	Modelo 3
1 Nodo	✓	✓	✓
Head Node	✓	✓	✓
Cluster Completo	✓	✓	✓

Tabla 5: Pruebas practicas por modelo

Luego de las pruebas indicadas anteriormente se ejecutaron aplicaciones de cálculo y consumo para recrear situaciones ficticias de estrés con finalidad de conocer las capacidades y consumo máximo de las distintas configuraciones así como sus distintas limitaciones.

Combinación de equipos	DistKeyGen	AWK	ForKit
1 Nodo	✓	✓	✓
Head Node	✓	✓	✓
Cluster Completo	✓	✓	✓

Tabla 6: Pruebas de rendimiento

- ✓ Corresponde a las pruebas finalizadas exitosamente.

## **9.1 Pruebas Prácticas**

Todas las pruebas de esta etapa fueron realizados con animaciones creadas en Blender, y siempre se renderizaron los 100 primeros frames de cada una de ellas. El requisito mínimo para considerar la prueba como finalizada exitosamente es renderizar todos los frames y juntarlo en un archivo de vídeo o dejarlos disponibles uno a uno para su posterior unión.

En el caso de el renderizado por GPU, el motor de renderizado cycles esta hecho para, primero, realizar un pre-procesamiento del modelo y hacerlo lo mas detallado posible en cierta cantidad de pasadas con la finalidad de logra mayor detalles que el renderizado por CPU. En este caso, hemos establecido esa cantidad de pasadas en 10, lo que hace que el modelo quede agradable a la vista y con una iluminación de calidad similar a la obtenida por CPU.

Los distintos niveles de detalles y ubicación de la cámara en cada uno de los modelos son los factores a considerar a la hora de analizar los resultados, por lo que se detallan a continuación:

### **Modelo 1**

- Cantidad Total de Vértices: 942
- Tamaño del archivo: 201,1 Kb
- Iluminación: 1 Sun, 1 Hemi
- Objetos: 1 Humano baja calidad

### **Modelo 2**

- Cantidad Total de Vértices: 128075
- Tamaño del archivo: 2,0 Mb
- Iluminación: 1 Sun
- Objetos: 1 Humano baja calidad, 1 vehículo calidad media, 2 árboles alta calidad

### **Modelo 3**

- Cantidad Total de Vértices: 3818746
- Tamaño del archivo: 201,2 Mb
- Iluminación: 1 Sun, 1 Area
- Objetos: 30 árboles alta calidad, 1 vehículo baja calidad, 18 bloques.

Un gráfico resumen de toda esta sección de pruebas se puede ver en la Ilustración 31.

### **9.1.1 Render CPU – Blender**

Esta prueba consiste en renderizar los primeros 100 frames utilizando el motor predeterminado de blender.

El archivo .blender debe estar configurado previamente con el motor blender.

Para ejecutar esta prueba lo hacemos dentro del directorio compartido de la siguiente forma:

Ej: nodo1@nodo1:/media/NAS>blender -b -a modelo.blend -s 1 -e100

Se ordenaron los tiempos de ejecución en la tabla 4.

	Modelo 1	Modelo 2	Modelo 3
Nodo Normal	00:02:06	00:05:39	00:14:44
Head Node	00:00:58	00:03:29	00:13:44
Cluster Completo	N/A	N/A	N/A

*Tabla 7: Render CPU - Blender*

Conclusiones:

- Este motor de renderizado no puede aprovechar los recursos del cluster por defecto, por lo que se debe omitir la prueba de Cluster Completo
- El tiempo de renderizado del HeadNode es menor al de un nodo normal, lo que quiere decir que la aplicación se paraleliza localmente y aprovecha la mayor cantidad de núcleos del procesador del servidor

### **9.1.2 Render CPU – Blender Background**

Esta prueba consiste en renderizar los primeros 100 frames utilizando el motor modificado de blender, el cual llamamos BlenderBG (Background del ingles, Segundo Plano) para que haga mejor uso del cluster y de las capacidades de cada equipo.

El archivo .blender debe estar configurado previamente con el motor blender.

Para ejecutar esta prueba lo hacemos dentro del directorio compartido de la siguiente forma:

Ej: nodo1@nodo1:/media/NAS>blenderBG CPU modelo.blend 1 100

Se ordenaron los tiempos de ejecución en la tabla 5.

	Modelo 1	Modelo 2	Modelo 3
Nodo Normal	00:01:38	00:04:46	00:07:38
Head Node	00:00:36	00:02:18	00:03:43
Cluster Completo	00:00:18	00:00:57	00:02:01

Tabla 8: Render CPU - Blender Background

Conclusiones:

- Este motor de renderizado si puede aprovechar los recursos del cluster.
- El tiempo de renderizado del HeadNode es menor al de un nodo normal, esta vez la diferencia es incluso mas notoria.
- Los tiempos de renderizado se reducen al utilizar el cluster con CPU hasta en un 62%

### 9.1.3 Render GPU – Cycles

Esta prueba consiste en renderizar los primeros 100 frames utilizando el motor de renderizado por GPU Cycles.

El archivo .blender debe estar configurado previamente con el motor cycles.

Para ejecutar esta prueba lo hacemos dentro del directorio compartido de la siguiente forma:

Ej: nodo1@nodo1:/media/NAS>blender -b -a modelo.blend -s 1 -e100

Se ordenaron los tiempos de ejecución en la tabla 6.

	Modelo 1	Modelo 2	Modelo 3
Nodo Normal	00:03:19	00:34:42	00:38:43
Head Node	00:03:24	00:28:00	00:51:33
Cluster Completo	N/A	N/A	N/A

Tabla 9: Render GPU - Cycles

Conclusiones:

- Este motor de renderizado no puede aprovechar los recursos del cluster por defecto, por lo

que se debe omitir la prueba de Cluster Completo

- El tiempo de renderizado del HeadNode no es mejor en todos los casos debido a que el GPU de los nodos normales es superior al del HeadNode, por lo que el render es mas rapido en los modelos que requieren renderizado puro sin mucha iluminación. En el caso del modelo 2 que es el mas detallado a pesar de no tener tantos elementos requiere el uso conjunto de CPU para ser procesado, por que el renderizado es mas rápido en el HeadNode.

#### 9.1.4 Render GPU – Cycles Background

Esta prueba consiste en renderizar los primeros 100 frames utilizando el motor de renderizado por GPU Cycles modificado para que haga uso del cluster y un mejor uso de cada equipo.

El archivo .blender debe estar configurado previamente con el motor cycles.

Para ejecutar esta prueba lo hacemos dentro del directorio compartido de la siguiente forma:

Ej: nodo1@nodo1:/media/NAS>blenderBG GPU modelo.blend 1 100

Se ordenaron los tiempos de ejecución en la siguiente tabla:

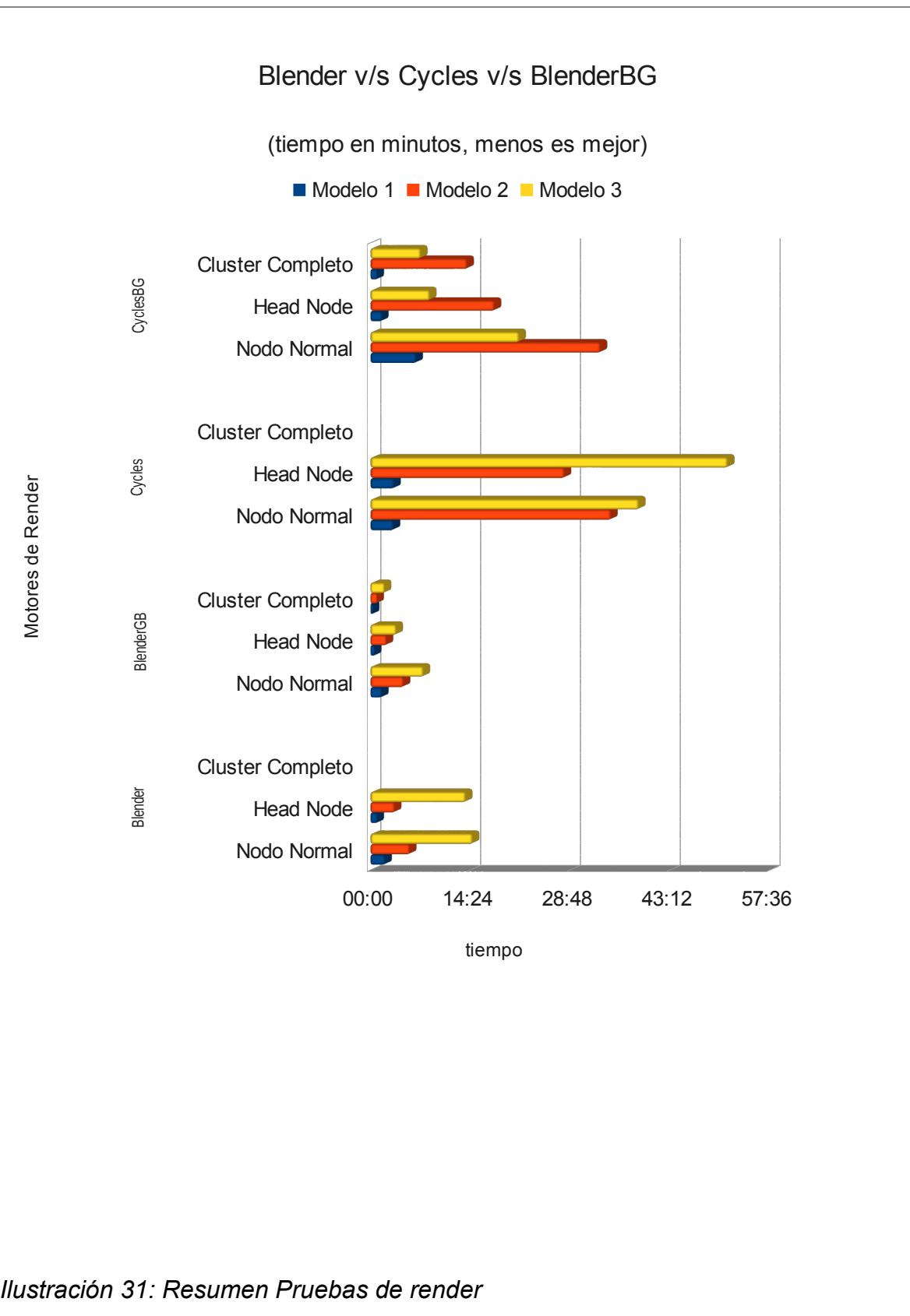
	Modelo 1	Modelo 2	Modelo 3
Nodo Normal	00:06:36	00:33:11	00:21:27
Head Node	00:01:38	00:17:54	00:08:33
Cluster Completo	00:00:58	00:13:57	00:07:15

Tabla 10: Render GPU - Cycles Background

Conclusiones:

- Este motor de renderizado si puede aprovechar los recursos del cluster y sus GPUs.
- El tiempo de renderizado del HeadNode es menor al de un nodo normal, esto es por que al sobre exigir al motor de renderizado por GPU es apoyado por el uso de CPU, que es donde super el HeadNode al resto de los nodos.
- Los tiempos de renderizado se reducen al utilizar el cluster con GPU y CPU hasta en un 305% en comparación a utilizar GPU en un Nodo Normal

## 9.1.5 Resultados



- Se observa un rendimiento superior utilizando el cluster en todos los casos.
- Existe una anomalía en los tiempos de renderizado por GPU al comparar los tiempos de Cycles entre el nodo normal y el Head Node. Esto se debe a que los nodos normales tienen mejor capacidad gráfica. Esto sucede en el modelo con mejor iluminación y detalles más cercanos.
- En todo momento, los equipos tenían funcionalidad plena para tareas comunes como editar texto, y navegar.

### **9.1.6 Conclusiones pruebas prácticas**

- Se debe dar preferencia al renderizado en cluster para ahorrar tiempo de renderizado
- El renderizado por GPU debe utilizarse de preferencia cuando se quiere alcanzar una muy alta calidad, en cualquier otro caso, es mejor realizar el renderizado por CPU que es mucho mas rápido.

## 9.2 Pruebas de rendimiento

Esta etapa es una de las más importantes y la que nos dará la respuesta si efectivamente la configuración está correcta en términos de rendimiento de los equipos.

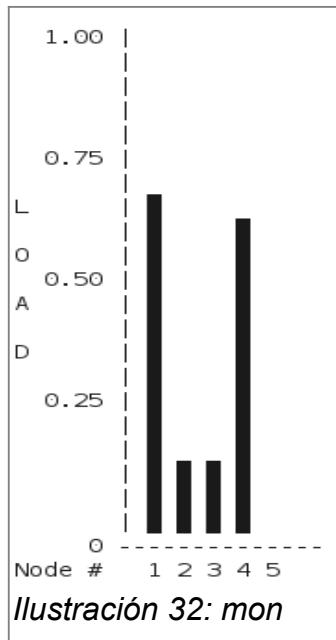
Se realizaron pruebas en relación a la capacidad de cálculo y migración de procesos, siendo esto último uno de los aspectos más importantes en MOSIX. En primer lugar se ejecutó la prueba que MOSIX provee para verificar la estabilidad, tanto así como la migración de procesos en forma automática y transparente al usuario.

```
nodo1@nodo1:~$ mosrun testload &
```

Esta prueba se ejecuta n veces, es decir, se lanza la instrucción n veces en una terminal y en modo segundo plano con el operador “&”, sin el uso del operador no es posible llegar a estresar el nodo y probar la efectividad de MOSIX.

Después de 10 ejecuciones simultáneas logramos percibir el proceso de migración de MOSIX revisando la carga de los nodos con “mon”.

Con esto comprobamos el correcto funcionamiento de MOSIX y se dispone a realizar cada una de las pruebas programadas.



Nota: cada una de las pruebas son ejecutadas en la terminal de comandos.

## 9.2.1 DistKeyGen

Prueba que consiste en generar 80.000 claves RSA con encriptación de 2048bits y es distribuida en 40 procesos fork's simultáneamente.

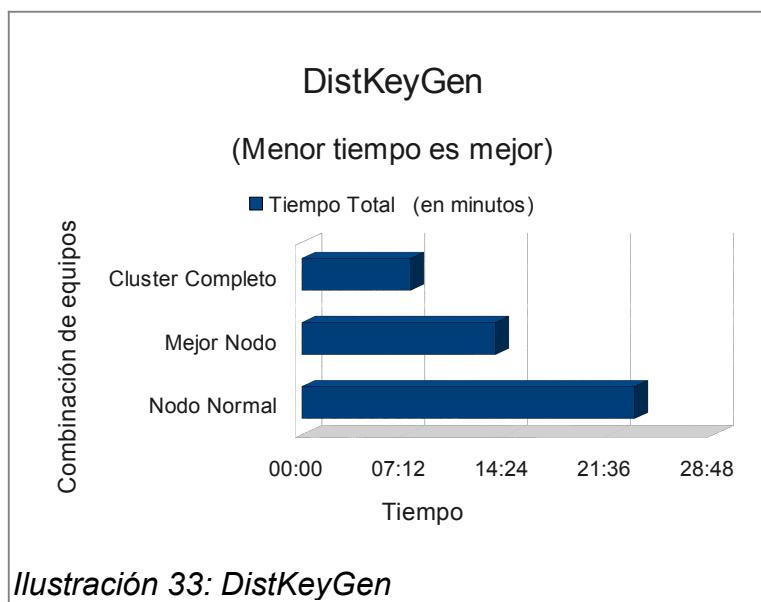
La prueba se ejecuta con:

Ej: `nodo1@nodo1:~$ ./DisKeyGen (en un nodo) o mosrun ./DistKeyGen (en cluster)`

DistKeyGen	Tiempo Total	%
Nodo Normal	23:16	305,47%
Mejor Nodo	13:33	177,90%
Cluster Completo	07:37	100,00%
MIN	00:07:37	

Tabla 11: DistKeyGen

Obtenemos el gráfico de la ilustración 33.



Conclusiones:

- La diferencia en tiempo de ejecución entre el uso de cluster y nodo normal es de 205,47% más costoso para nodo normal, en comparación a la diferencia de 77,9% del cluster y mejor nodo respectivamente.
- Según lo anterior podemos notar que al generar muchos ficheros de pequeño tamaño el uso de recursos es casi lineal (mientras más nodos trabajen no significa que sea muy rápido el proceso de escritura en disco).

## 9.2.2 AWK

Esta prueba consiste en ejecutar un ciclo for anidado calculando  $(100000)^2$  operaciones de cálculo matemático. Esta prueba es ejecutada por un script de inicio que lanza 10 veces la instrucción en segundo plano.

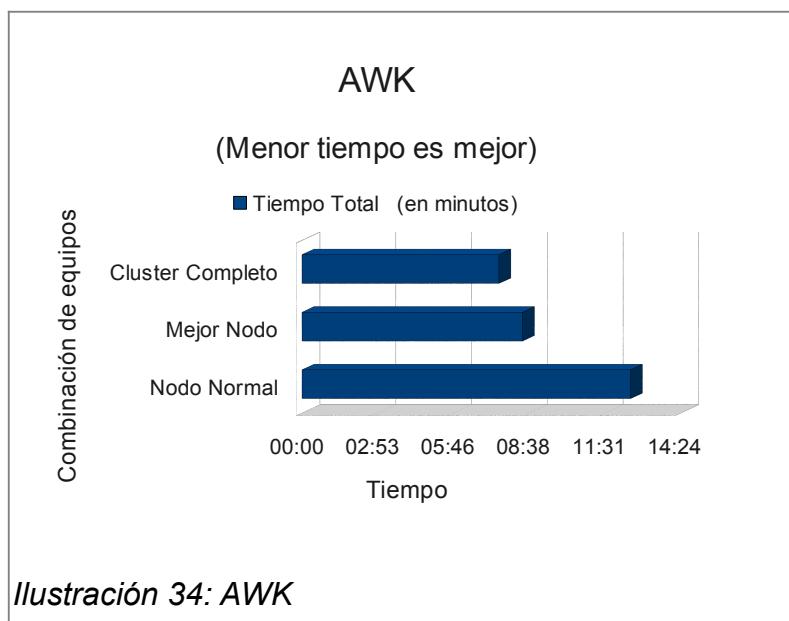
La prueba se ejecuta con:

Ej: `nodo1@nodo1:~$ ./start_awk.sh` (en un nodo) o `./start_awk.sh` (cambiando la orden de lanzamiento anteponiendo "mosrun" para el caso de cluster)

AWK	Tiempo Total (en minutos)	%
Nodo Normal	12:29	166,82%
Mejor Nodo	08:23	112,03%
Cluster Completo	07:29	100,00%
MIN	07:29	

Tabla 12: AWK

Obtenemos el gráfico de la ilustración 34.



Conclusiones:

- La diferencia en tiempo de ejecución entre el uso de cluster y nodo normal es de 66,82% más costoso para nodo normal, en comparación a la diferencia de 12,03% del cluster y mejor nodo respectivamente.
- Se puede notar que el cálculo de los ciclos anidados se ve beneficiado por la cantidad de núcleos que posea cada máquina a la que se le aplica. Es por esta razón en que el mejor nodo (nodo que contiene 16 núcleos) tarda casi lo mismo que al correr la prueba sobre todo el cluster tomando en cuenta que la prueba ejecuta 10 iteraciones de la misma instrucción y ocupa 10 núcleos calculando a la vez.+
- De acuerdo a lo anterior el nodo normal solo posee 6 núcleos para calcular esta prueba, sin importar que estos sean más rápidos que los del mejor nodo. Aun así se aprecia la diferencia de “dividir para conquistar” que utiliza el mejor nodo.

### 9.2.3 Forkit

Esta prueba realiza un simple cálculo matemático de seno(x)+raiz(x) por un ciclo de 1000000 veces. Esta prueba es iniciada muchas veces a la vez ocupando muchos procesadores en el cluster MOSIX.

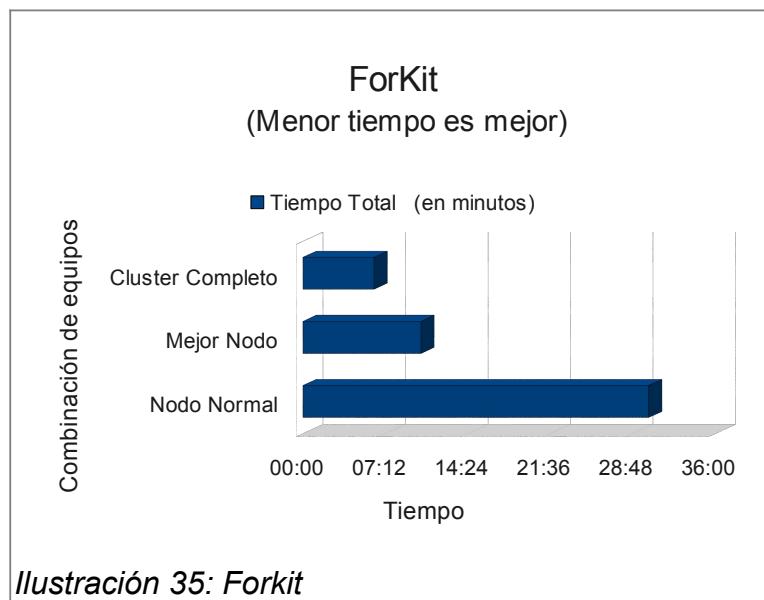
La prueba se ejecuta con:

Ej: *nodo1@nodo1:~\$ ./start\_forkit.sh (en un nodo) o ./start\_forkit.sh (cambiando la orden de lanzamiento anteponiendo “mosrun” para el caso de cluster)*

ForKit	Tiempo Total (en minutos)	%
Nodo Normal	30:10	486,56%
Mejor Nodo	10:18	166,13%
Cluster Completo	06:12	100,00%
MIN	06:12	

*Tabla 13: Forkit*

Obtenemos el gráfico de la ilustración 35:



Conclusiones:

- La diferencia en tiempo de ejecución entre el uso de cluster y nodo normal es de 386,56% más costoso para nodo normal, en comparación a la diferencia de 66,13% del cluster y mejor nodo respectivamente.
- Según el resultado obtenido es posible apreciar una gran diferencia de calculo entre el uso de cluster y nodo normal. Sin embargo en el caso de mejor nodo no es tan significativa ya que como la prueba anterior el poder de más núcleos trabajando a la vez en una sola maquina es aparentemente mejor para este tipo de cálculos matemáticos.

### **9.3 Conclusiones pruebas rendimiento**

Las pruebas anteriores sirvieron para comprobar el funcionamiento correcto del cluster y ver su real poder de cálculo tomando como referencia el tiempo que demora en terminar una instrucción.

MOSIX es eficiente en el balanceo de carga de los nodos, dejando al menos un núcleo de cada nodo libre para realizar otras tareas, por esto es difícil lograr que se congele el cluster a menos que se haga un uso extensivo de la memoria lo que provoque que vuelque parte de memoria a los discos, reduciendo el rendimiento del equipo.

Según las pruebas anteriores observamos que el uso mediante cluster es siempre beneficioso para cualquier tipo de proceso, pero en particular se ven beneficiadas las aplicaciones que hagan uso de paralelismo o procesos pesados para su ejecución. La computación futura apunta hacia eso, el mejor uso de los recursos disponibles y en menor tiempo de ejecución.

Para más información de las pruebas y revisar el código ver apéndice “**Pruebas de Rendimiento**”

## 10 Oportunidades Futuras

El diseño de cluster no solo se orienta al renderizado de animaciones, además, dado las posibilidades de usos es posible usarlo para otros fines en conjunto con los ya planteados en esta investigación. Algunos de los usos académicos en que se puede utilizar son:

- Creación de GRID compuesto por varios cluster de distintas instituciones con el fin de aumentar la capacidad de procesamiento y compartir recursos.

Además, otras áreas en las que puede aplicarse esta investigación fueron omitidas intencionalmente ya que se escapa de los objetivos propuestos, pero deben ser mencionadas para hacer notar la multifuncionalidad de este tipo de cluster para procesamiento paralelo.

- Simulaciones Físicas y Matemáticas.
- Procesamiento de datos mediante cálculo GPGPU.
- Aplicaciones de Bioingeniería.
- Medicina Molecular.
- Estación de Realidad Virtual.
- Balanceador de Carga.
- Blender Game Engine con cluster.

Estas son solo algunas posibilidades de uso a implementar en este cluster.

# 11 Conclusiones

## Conocimiento de una Nueva Tecnología

Antes de realizar el proyecto, personalmente no teníamos conocimientos expertos en relación a clusters, solo conocíamos la teoría. A través de la investigación realizada logramos entender de mejor manera de que tan amplio es el mercado de la computación paralela, sobre todo con nuevos formatos y presentaciones populares como “la nube”. La computación del futuro apunta a ese tipo de mercado, un mercado en donde ya no existirán las computadoras tradicionales con procesadores mononucleo, de echo el mercado actual ya presenta amplia variedad de productos de múltiples núcleos, encapsulando el término “cluster” como algo normal dentro de un computador. Además se destaca la gran variedad de Software dedicado a este mercado, existiendo desde distribuciones completamente equipadas para ser instaladas o una aplicación que potencia el uso de un set de nodos convirtiéndolos en clusters para un fin determinado. También destaca el hecho de que ya no solo es posible hacer uso del poder de procesamiento de datos por medio de los procesadores comunes, sino, ahora se habla de clusters de GPU's o más bien la computación GPGPU. Esta investigación deja abierta la brecha a la adopción de nuevas tecnologías y realización de proyectos futuros utilizando nuestro trabajo como base.

## Implementación de un Cluster tipo MOSIX

El objetivo principal del proyecto era implementar un cluster que permitiera no solo el uso de este como maquina para procesar datos, sino, también como escritorio de trabajo. Una de las grandes ventajas que destacamos de MOSIX es el echo de ser independiente de la distribución Linux y de como sea configurada. El requisito era usar la tecnología adquirida obteniendo un resultado óptimo para el fin que fue diseñada y el proyecto MOSIX cumple con tal requisito. Los nodos se instalaron utilizando Debian como sistema base en conjunto con el escritorio Gnome además de las aplicaciones necesarias para convertir el cluster en una estación de trabajo de animaciones y renderizado de imágenes dentro de un ambiente OpenSource.

El costo de la aplicación es nula en términos económicos. La curva de aprendizaje es bastante rápida debido a la extensa documentación de MOSIX disponible en su sitio web. Aun así no existen muchos proyectos que hagan uso de esta nueva tecnología, la razón es que existen otras que llevan más tiempo distribuidas y son más conocidas.

La productividad de los participantes en el proyecto Alfa Gaviota se vio mejorada, obteniendo así mejores resultado en menores tiempos.

La implementación de este cluster nos trajo algunos beneficios no solo de velocidad y mejor uso de recursos, sino, de una amplia variedad de conocimientos los cuales serán de gran apoyo a futuras investigaciones relacionadas con esta.

## **Pruebas**

Se realizaron algunas pruebas prácticas y de rendimiento sobre el cluster. Todas estas pruebas fueron llevadas a cabo con éxito. Cada prueba realizada es acompañada con su respectivo gráfico para una mejor comprensión.

Sin lugar a dudas el cluster se comporta muy bien disminuyendo en gran cantidad los tiempos de espera entre una y otra ejecución además de encargarse de la migración automática de procesos. En el caso de las pruebas de rendimiento la diferencia es en algunos casos abismante dependiendo de algunos factores tales como: velocidad de transferencia de la red, velocidad de escritura/lectura de disco entre otras. Es aquí donde encontramos el limitante del cluster que no depende de la velocidad del nodo, sino de el esquema de red, tecnología utilizada de cableado, discos usados, etc. Con mejor hardware se pueden obtener mayores resultados pero para efectos de la configuración de los equipos esto no es determinante puesto que cumple perfectamente con su función. En las pruebas prácticas encontramos diferencias notorias entre una configuración u otra. Siendo las pruebas cycles v/s cycleBG y blender v/s blenderBG comparadas entre si logrando resultados mucho mejores en la de utilización de CPU. La explicación a esto radica en que la prueba por GPU es más lenta porque intenta conseguir resultados más realistas de la imagen renderizada. Esta tiene como función apoyar la iluminación de la escena para lograr un mejor resultado. Se destaca principalmente que el uso de cluster siempre es mejor que en cualquiera de las otras variantes.

## **Consideraciones y Experiencias**

Esta investigación se desarrollo en torno a los objetivos planteados por el proyecto Alfa Gaviota. Por todos los medios se trato de cumplir a cabalidad el origen del proyecto en que actualmente esta en curso gracias al apoyo de este cluster.

Para la documentación de esta investigación se consulto páginas oficiales haciendo uso de internet como la mejor herramienta de información sobre estas tecnologías. Comúnmente es difícil encontrar literatura de manera cercana ya que se trata de proyectos de un nivel técnico. En la insaciable búsqueda de una solución nos encontramos con muchas alternativas pero no todas se ajustan a los requerimientos que necesitamos cumplir. El proyecto MOSIX es nuevo en el mundo del clustering. Recientes investigaciones se han desarrollado en torno a el como es el caso de MOSIX-VCL proyecto en el cual se pretende replicar el éxito del clustering pero llevándolo a la computación mas poderosa en términos de procesamiento y que tiene que ver con el mercado de las GPU, este es amplio y cada día es mas cotizado ya que tiene excelentes niveles de retroalimentación por parte de usuarios exigentes que piden más poder de cálculo para sus fines personales.

Con el apoyo de fundaciones como The Khronos Group Inc, el mundo de la supercomputación esta más cerca a los usuarios comunes. OpenCL es el standard de programación para propósitos GPGPU y se convirtió en la segunda herramienta mas utilizada en computo HPC después de Intel Threading Blocks y superando a OpenMP.

En el transcurso de la investigación también se investigó sobre MOSIX-VCL, como medio de procesamiento de alta velocidad para investigaciones futuras que tengan que ver con cálculos científicos, físicos y matemáticos. Sin duda el uso de esta herramienta será útil y beneficiosa para los demás propósitos de fines educativos que destine dar el proyecto Alfa Gaviota o la Universidad del Bío-Bío.

En lo posible se trató de generar un documento instructivo y detallista que sirva como guía para la creación de un cluster, no necesariamente de alto rendimiento. Se pueden crear clusters de alta disponibilidad y balanceo de carga con bastante sencillez con tan solo algunos conocimientos básicos de redes. Esta investigación pretende dar un impulso al mundo del clustering como una tecnología innovadora y que al día de hoy está teniendo más y más auge.

## APENDICE I - Herramientas de MOSIX

### Monitores – Ver que esta sucediendo

-mon y mmon son dos monitores para ver el estado de los recursos en cada uno de los cluster y en todos los cluster a la vez.

-mon – Muestra información básica (formato tty) acerca de los recursos locales del cluster

-tipos de salidas:

- “l” – Carga del CPU (relativa)
- “f” - Número de procesos congelados
- “m” - Memoria (utilizada y libre), espacio en swap (utilizado y libre) - consecutivamente
- “u” - Utilización
- “d/D” - Nodos muertos
- “h” - Ayuda con la lista completa de opciones

-mmon – Incluye todas las funciones de mon y muchas mas

-tipos de salidas (consecutivas):

- “l” - Carga, carga + procesos congelados, procesos congelados
- “m” - Memoria (utilizada y libre), espacio en swap, espacio en el disco
- “g” - Estado del multi-cluster (procesos locales/huésped, prioridad)
- “i” - Leer el ratio de entrada/salida, escribir ratio, total entrada/salida
- “o” - Actividad de la red (entrada, salida, local), ratio RPC (Remote Procedure Call)
- “s” - Velocidad del CPU, estado, Versión de MOSIX, Tiempo de ejecución
- “d/D” - Nodos muertos
- “h” - Ayuda con la lista completa de opciones

Otras características de mmon:

- Puede correr en nodos que no poseen MOSIX, por ejemplo, tu estación de trabajo
- mmon -h bmos-01 (Nodo #1 en el cluster bmos)
  - Muestra el estado de varios cluster (consecutivamente)
- Ejemplo: mmon -c amos-01, bmos-01, cmos-01

- Esquema de colores personalizados usando el archivo ~/.mmon.cfg

## mosrun – Ejecutando procesos en MOSIX

- Para ejecutar una aplicación en MOSIX la orden debe comenzar con mosrun.
  - Este tipo de programas pueden migrar a otros nodos
  - Ejemplo: > mosrun myprog 1 2 3 (ejecuta myprog con los argumentos 1 2 3)
- Los programas que no son iniciados con mosrun se ejecutan de forma nativa en Linux y no pueden migrar a otros nodos
- Un programa que es iniciado por mosrun y todos sus procesos hijos se mantienen bajo MOSIX
- Un proceso MOSIX (que fue iniciado con mosrun) puede ser ejecutado en nodos mosix de distintas arquitecturas.

### Ejemplo de migración de procesos:

- Acceder a cualquier nodo MOSIX en un cluster
  - En una terminal ejecutar mon o mmon
  - En otra terminal iniciar un proceso de uso intensivo de dos CPU, por ejemplo el programa “testload” que esta incluido en MOSIX
- >mosrun testload &
- >mosrun testload &
- Observar en la terminal de mon/mmon como se mueven los procesos en los nodos del cluster
  - Escribir moskillall para detener los procesos en ejecución

## mosrun – Opciones de asignación de nodos

- -r{nombre del host} – ejecuta en este host (nodo)
- -{a.b.c.d} – ejecuta en el nodo con esta IP
- -{n} – ejecuta en el nodo n
- -h – ejecuta en el nodo home
- -b – Intenta elegir el mejor nodo
- -jID1-ID2[,ID3-ID4] – ejecuta en un nodo al azar en el rango ID1-ID2, ID3-ID4, ...
- Ejemplos:
  - > mosrun -rmos1 miprograma (ejecuta en el nodo mos1)

> mosrun –b miprograma (ejecuta en el mejor nodo)

> mosrun –3 miprograma 1 2 3 (ejecuta en el nodo #3, con argumentos 1 2 3)

## mosrun – Donde pueden migrar los procesos

- -G Permite al proceso migrar a los nodos en otros cluster
  - De otra forma, el proceso está limitado al cluster local
  - -G{class} – si class > 0 quiere decir que un proceso tiene permitido migrar a nodos en otro cluster. -G es equivalente a -G1
- -m{megabytes} – especifica la cantidad máxima de memoria necesitada por el programa, para prevenir la migración del proceso a nodos que no tienen suficiente memoria
- Además de la migración, las opciones -G y -m afectan al a la asignación inicial (-b flag) y la cola (visto más adelante)
- Ejemplo:
  - > mosrun –G –m1000 miprograma (permite a miprograma ejecutarse en otro cluster, pero solo en nodos con al menos 1GM libre de memoria)

## mosrun – marcando los procesos

- La opción -J{Job ID} de mosrun permite agrupar para la fácil identificación en distintas instancias de mosrun
  - El “Job ID” es un entero (por omisión 0)
  - Cada usuario puede asignar sus propios “Job ID”
- “Job ID” es heredado por todos los procesos hijos
- “Job ID” puede ser visto por mosq y mosps
- Todos los trabajos de un usuario con el mismo “Job ID” pueden ser matados colectivamente por moskillall y migrados por migrate
- Ejemplos:
  - > mosrun –J20 miprograma (ejecuta miprograma con Job\_ID = 20)
  - > mosps –J20 (muestra sólo mis procesos con Job\_ID = 20)
  - > moskillall –J20 (mata todos los procesos con Job\_ID = 20)

## mosrun – Ejecutar trabajos en lotes

mosrun puede enviar lotes de trabajos a otros nodos en el cluster local

Existen 2 tipos:

- E – Ejecutar procesos nativos de Linux
- m – Ejecutar procesos MOSIX pero su nodo home puede ser otro nodo del cluster
- La combinación de -E -b intenta asignar un trabajo de Linux al mejor nodo disponible
- La combinación de -M y -b intenta asignar el nodo home y el nodo donde se inicia la ejecución del proceso, al mejor nodo disponible
- El -E{dir} y -M{dir} especifica el directorio donde los trabajos deben ser ejecutados
- Por omisión se ejecutará el el directorio con el nombre del directorio actual en el nodo

–Ejemplos

```
> mosrun -E -rmos4 miprogramaLinux  
> mosrun -M/tmp -b miprogramaMosix
```

## mosps – Ver los procesos de MOSIX

mosps (como ps) provee información acerca de los procesos MOSIX (y posee varias de las cualidades de ps) incluyendo:

- WHERE – donde este el proceso
- FROM – origen del proceso
- ORIGPID – pid original en el nodo home
- CLASS – clase de proceso
- Para procesos MOSIX definidos por mosrun -G{class}
- Otros valores son batch y native
- FRZ – ¿Por qué está congelado el proceso?: A – Congelado automático (debido a la carga); E – expulsado; M – manualmente; --NOTFROZEN; N/A – no puede ser congelado; DUE – cuando el usuario está en espera de que se complete el proceso
- NMIGS – Numero de migraciones hasta ahora del proceso o sus ancestros

## mosps – Opciones

mosps soporta la mayoría de las opciones de ps

Las opciones especiales de mosps son:

- I – Muestra las direcciones de IP de los nodos
- h – Muestra los nombres de los host
- M – Muestra solo el último componente del host
- L - Muestra solo los procesos locales
- O – Muestra solo los procesos locales de otros nodos
- n – Muestra NMIGS
- V – Muestra solo los procesos huésped
- P – Muestra el ORIGPID
- D – Muestra DUE
- J{JobID} – Muestra solo los procesos con ese JobID

## mosps – Ejemplo:

> mosps –AMn

PID	WHERE	FROM CLASS	FRZ	NMIGS	TTY	CMD
24078	cmos-18	here local	-	1	pts/1	mosrun -b testload
24081	here	here local	-	0	pts/1	mosrun -b testload
24089	cmos-16	here local	-	1	pts/1	mosrun -b testload
30145	queue	here N/A	N/A	N/A	pts/1	mosqueue -b testload
30115	here	here local	M	0	pts/1	mosrun testload
30253	here	mos3 local	N/A	N/A	?	/sbin/remote

## mosrun – Cola

- Con la opción -q, mosrun deja el trabajo en una cola
- Los trabajos de todos los nodos dentro de un cluster comparten una cola
- La política de la cola es First-come-first-serve osea, el primero en ingresar es el primero en ser atendido, con algunas excepciones
- Los usuarios pueden asignar prioridad a sus trabajos, usando la opción -q{pri}
- El menor valor de pri tiene prioridad más alta

- La prioridad por omisión es 50. Puede ser cambiada por el sysadmin
- Para ejecutar trabajos con pri<50 se debe coordinar con el administrador del cluster
- Ejemplos:
  - > mosrun -q -b -m1000 miprograma (agrega un programa MOSIX a la cola)
  - > mosrun -q60 -G -b -J1 miprograma (agrega un trabajo de prioridad baja a un multi-cluster)
  - >mosrun -q30 -E -m500 miprograma (agrega un lote de trabajo de alta prioridad)

## mosq – Ver y controlar la cola

- mosq list – Muestra una lista con los trabajos en la cola
- mosq listall – Muestra los trabajos de la cola que están ejecutándose y los que están esperando en la cola
- Mosq delete {pid} – Elimina los trabajos en espera de la cola
- Mosq run {pid} – Ejecuta un proceso en espera
- Mosq cngpri {newpri}{pid} – Cambia la prioridad de un trabajo en espera
- Mosq advance {pid} – Mueve un trabajo en espera a la cabeza de las prioridades en la cola
- Mosq retard {pid} – Mueve un trabajo en espera al final de las prioridades en la cola
- 
- mosq - example
- > mosq listall
- PID USER MEM(MB) GRID PRI FROM COMMAND
- 21666 lior - NO RUN here mosrun -b testload
- 21667 lior - NO 50 here mosrun -b testload
- 21155 lior - NO 50 cmos-17 mosrun testload
- > mosq cngpri 20 21155
- > mosq listall
- PID USER MEM(MB) GRID PRI FROM COMMAND
- 21666 lior - NO RUN here mosrun -b testload
- 21155 lior - NO 20 cmos-17 mosrun testload
- 21667 lior - NO 50 here mosrun -b testload

## migrate – Controlar los procesos de MOSIX

- migrate {pid} {numero del nodo | ip-address | host-name} – mueve el proceso al nodo dado
- migrate {pid} home – Pide al proceso regresar al home
- migrate {pid} freeze – Pide al proceso que se congele
- migrate {pid} continue – Descongela el proceso
- migrate {pid} checkpoint – Marca el proceso
- migrate {pid} checkstop – Marca el proceso y lo detiene
- migrate {pid} exit – Marca el proceso y sale

## Características no soportadas

Algunas utilidades y bibliotecas usan características que no están soportadas por MOSIX. Normalmente, esas características no son críticas. Si es ejecutada una utilidad y aparece un mensaje como:

–MOSRUN: Shared memory (MAP\_SHARED) not supported under MOSIX

o

–MOSRUN: system-call 'futex' not supported under MOSIX

Intente usar la opción -e de mosrun para sobrellevar el problema

–Ejemplo: En vez de mosrun ls -la usar mosrun -e ls -la

## **APENDICE II - Problemas Presentados Durante la Realización del Proyecto**

Problema	Paralelizar el renderizado de Blender
Motivos	Necesario para utilizar la potencialidad de los equipos
Solución	Realizar llamadas al sistema ejecutando motor de renderizado de Blender en segundo plano para cada frame a procesar usando una aplicación que crea fork() de sí misma hasta terminar el trabajo de renderizado.
Problema	Creación del cluster físicamente
Motivos	Inexistencia del laboratorio Alfa Gaviota, el cual se encuentra retrasado por gestiones burocráticas
Solución	Configuración del cluster en máquinas virtuales y cluster físico con configuración de software similar pero con equipos domésticos
Situación Final	Posteriormente este problema fue solucionado con la llegada de los equipos a la universidad.
Problema	Imposibilidad para realizar pruebas de funcionamiento de la aplicación creada para renderizado
Motivos	Inexistencia del laboratorio Alfa Gaviota, el cual se encuentra retrasado por gestiones burocráticas
Solución	Prueba de aplicación en cluster con equipos domésticos, estas no reflejan el funcionamiento óptimo.
Situación Final	Posteriormente este problema fue solucionado con la llegada de los equipos a la universidad.
Problema	Imposibilidad para realizar pruebas de rendimiento de la aplicación creada
Motivos	Inexistencia del laboratorio Alfa Gaviota, el cual se encuentra retrasado por gestiones burocráticas
Solución	Se espero hasta la llegada del laboratorio finalizado el mes de Noviembre
Problema	Paralelización de la aplicación para utilizar GPU
Motivos	No se cuenta con equipos con soporte de procesamiento OpenCL o CUDA para el desarrollo. Inexistencia del laboratorio Alfa Gaviota
Solución	Se espero hasta la llegada del laboratorio finalizado el mes de Noviembre

## APENDICE III – Código de aplicación Blender BG

```
//gcc RenderGPUandCPU.c -o RenderBG -Wall
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <string.h>
#define MAX 20
#define MAXCPU 15

//MAX = Cantidad de frames que se renderizan a la vez

int main(int argc, char *argv[]){
    if(argc==5){
        int inicio = atoi(argv[3])-1; //Se aumenta en un frame al inicio, pero se elimina en el
        primer ciclo
        int final = atoi(argv[4]);
        int hijos=0;
        int flag=0;

        //Define message como la orden de renderizar solo en los mejores nodos
        char *bin = "mosrun -E -j1,2,3 blender -b -o ./ ";
        char *message = (char*) malloc(strlen(bin)+strlen(argv[2])+strlen(argv[1]));
        strcat(message, bin);
        strcat(message, argv[2]);
        strcat(message, " ");
        strcat(message, "-f ");

        //Define message como la orden de renderizar en todos los nodos
        char *bin2 = "mosrun -E -j1,2,3,4,5 blender -b -o ./ ";
        char *message2 = (char*) malloc(strlen(bin)+strlen(argv[2])+strlen(argv[1]));
        strcat(message2, bin2);
        strcat(message2, argv[2]);
        strcat(message2, " ");
```

```

strcat(message2, "-f ");

if(strcmp(argv[1], "GPU") == 0){
//GPU Render - Envia menos carga a los nodos con GPU de menos capacidad
while(inicio<final){

    if(hijos<=MAX){

        inicio=inicio+1; //Eliminacion del frame inicial en cada ciclo
        if(fork() == 0){

            char frame[4];
            sprintf(frame, "%d", inicio);

            if(flag == 0){

                strcat(message, frame);
                printf("%s", message); //Muestra la llamada al sistema (fork)
que se esta ejecutando

                system(message);
                exit(1);
            }else{

                strcat(message2, frame);
                printf("%s", message2); //Muestra la llamada al sistema (fork)
que se esta ejecutando

                system(message2);
                exit(1);
            }
        }

    }else{
        //printf("NO FORK\n");

    }
    if(flag == 0) flag = 1;
    else flag = 0;
    hijos = hijos + 1;
}else{
    wait(NULL);
    hijos = hijos - 1;
}
}

```

```

}else{
//CPU Render
while(inicio<final){
    if(hijos<=MAXCPU){
        inicio=inicio+1; //Eliminacion del frame inicial en cada ciclo
        if(fork()==0){
            char frame[4];
            sprintf(frame, "%d", inicio);
            strcat(message2, frame);
            printf("%s", message2);
            system(message2);
            exit(1);
        }else{
            //printf("NO FORK\n");
        }
        hijos=hijos+1;
    }else{
        wait(NULL);
        hijos=hijos-1;
    }
}
}

}else{
printf("Modo de uso:\n");
printf("RenderParalelo [CPU | GPU] [Archivo.blend] [frame inicial] [frame final]\n");
}

system("echo \\ \\");
return 0;
}

```

## APENDICE IV - Modificaciones VCL

### Script Original:

```
#!/bin/sh -
#
# chkconfig: 2345 95 5
# description: MOSIX-VCL is a Virtual OpenCL platofrm
#           for combining the power of many remote GPUs
#
# vcl      Script to stop/start MOSIX-VCL
#
# Author:    Amnon Shiloh
### BEGIN INIT INFO
# Provides: MOSIX-VCL
# Required-Start: $network
# Should-Start:
# Required-Stop:
# Should-Stop:
# Default-Start: 2 3 4 5
# Default-Stop: 0 1 6
# Short-Description: MOSIX-VCL
# Description: VCL - Virtual OpenCL for combining the power of many remote GPUs
### END INIT INFO
#
rc_reset() {
    rc_status=0
}

rc_status() {
    return $rc_status
}
```

```

rc_check() {
    case $? in [1-9]*) rc_status=$? ;;
    esac
}

rc_failed() {
    rc_status=$1
}

rc_exit() {
    exit $rc_status
}

[ -x /etc/init.d/functions ] && . /etc/init.d/functions

stop_host() {
    [ -x /sbin/broker.old ] && {
        killproc /sbin/broker.old
        rc_check
    }
    killproc /sbin/broker
    rc_check
}

```

```

stop_backend() {
    s=0
    [ -x /sbin/opencld.old ] && {
        checkproc /sbin/opencld.old
        case $? in 7) : ;;
        0) s=1
            killproc /sbin/opencld.old
            rc_check
            ;;
        esac
    }
    checkproc /sbin//opencld
}

```

```

case $? in 7) : ;;
  0) s=1
    killproc /sbin/opencld
    rc_check
    ;;
esac
case $s in 1) sleep 1 ;; esac
}

start_host() {
  [ -e /etc/vcl/is_host ] || return 0
  if [ -e /etc/vcl/nodes ]
  then
    if [ -e /etc/vcl/mosix ]
    then
      args="-f/etc/vcl/nodes -m"
    else
      args=-f/etc/vcl/nodes
    fi
  else
    case `cat /etc/vcl/mosix 2>/dev/null` in
      1) args=-m ;;
      2) args=-M ;;
      *) echo Identity of back-end nodes is not defined.
         rc_failed 2
         return 2 ;;
    esac
  fi
  [ -x /sbin/broker.old ] && checkproc /sbin/broker.old && {
    echo Already Running.
    rc_failed 1
    return 1
  }
  checkproc /sbin/broker && {
    echo Already Running.

```

```

        rc_failed 1
        return 1
    }
    startproc /sbin/broker $args
    rc_check
}

start_backend() {
    [ -e /etc/vcl/is_back_end ] || return 0
    [ -x /sbin/opencl.old ] && checkproc /sbin/opencl.old &&
{
    echo Already Running.
    rc_failed 1
    return 1
}
checkproc /sbin/opencl && {
    echo Already Running.
    rc_failed 1
    return 1
}
startproc /sbin/opencl
rc_check
}

rc_reset

case "$1" in
start)
    echo "Starting MOSIX-VCL..."
    start_host
    start_backend
    ;;
stop)
    echo "Stopping MOSIX-VCL..."
    stop_host

```

```

stop_backend
;;
status)
any=n
[ -e /etc/vcl/is_host ] && {
    any=y
    checkproc /sbin/broker
    case $? in 0) echo Hosting-node is running. ;;
                  *) rc_check
                     echo Hosting-node is not running. ;;
    esac
}
[ -e /etc/vcl/is_back_end ] && {
    any=y
    checkproc /sbin/opencld
    case $? in 0) echo Back-end is running. ;;
                  *) rc_check
                     echo Back-end is not running. ;;
    esac
}
case $any in n) rc_failed 3 ;; esac
;;
restart|reload)
echo "Restarting MOSIX-VCL..."
stop_host
stop_backend
start_host
start_backend
;;
start_host)
if [ -e /etc/vcl/is_host ]
then
    echo "Starting MOSIX-VCL host..."
    start_host
else

```

```

        echo "This is not a MOSIX-VCL hosting-node."
fi
;;
start_backend)
if [ -e /etc/vcl/is_back_end ]
then
        echo "Starting MOSIX-VCL back-end..."
        start_backend
else
        echo "This is not a MOSIX-VCL back-end node."
fi
;;
stop_host)
echo "Stopping MOSIX-VCL host..."
stop_host
;;
stop_backend)
echo "Stopping MOSIX-VCL back-end..."
stop_backend
;;
restart_host)
echo "Restarting MOSIX-VCL host..."
stop_host
if [ -e /etc/vcl/is_host ]
then
        start_host
else
        echo "This is not a MOSIX VCL hosting-node."
fi
;;
restart_backend)
echo "Restarting MOSIX-VCL back-end..."
stop_backend
if [ -e /etc/vcl/is_back_end ]
then

```

```
start_backend
else
    echo "This is not a MOSIX VCL back-end node."
fi
;;
*)
echo "Usage: vcl {start|stop|status|restart|reload}"
echo " or  vcl {start_host|stop_host|restart_host}"
echo " or  vcl {start_backend|stop_backend|restart_backend}"
exit 1
esac

true
rc_status -v
rc_exit
```

**Script Modificado:**

```
#!/bin/sh -  
#  
# chkconfig: 2345 95 5  
# description: MOSIX-VCL is a Virtual OpenCL platofrm  
#           for combining the power of many remote GPUs  
#  
# vcl      Script to stop/start MOSIX-VCL  
#  
# Author:    Amnon Shiloh  
### BEGIN INIT INFO  
# Provides: MOSIX-VCL  
# Required-Start: $network  
# Should-Start:  
# Required-Stop:  
# Should-Stop:  
# Default-Start: 2 3 4 5  
# Default-Stop: 0 1 6  
# Short-Description: MOSIX-VCL  
# Description: VCL - Virtual OpenCL for combining the power of many remote GPUs  
### END INIT INFO  
#
```

PATH=/sbin:/usr/sbin:/usr/local/sbin:/bin:/usr/bin:/usr/local/bin

```
#. /etc/rc.status  
. /lib/lsb/init-functions
```

[ -x /etc/init.d/functions ] && . /etc/init.d/functions

```
stop_host() {  
#     [ -x /sbin/broker.old ] && {  
#         killproc /sbin/broker.old  
#         rc_check  
#     }  
#     /sbin/start-stop-daemon --stop --exec /sbin/broker
```

```

#      rc_check
}

stop_backend() {
    s=0
    [ -x /sbin/opencld ] && {
        #status_of_proc /sbin/opencld
        case $? in 7) : ;;
        0) s=1
            /sbin/start-stop-daemon --stop --exec /sbin/opencld
#
        rc_check
            ;;
        esac
    }
    #status_of_proc /sbin/opencld
    #case $? in 7) : ;;
    # 0) s=1
    #      /sbin/start-stop-daemon --stop --name /sbin/opencld
#
    #      rc_check
    #      ;;
    #esac
    case $s in 1) sleep 1 ;; esac
}

start_host() {
    [ -e /etc/vcl/is_host ] || return 0
    if [ -e /etc/vcl/nodes ]
    then
        if [ -e /etc/vcl/mosix ]
        then
            args="-f/etc/vcl/nodes -m"
        else
            args=-f/etc/vcl/nodes
        fi
    else

```

```

        case `cat /etc/vcl/mosix 2>/dev/null` in
            1) args=-m ;;
            2) args=-M ;;
            *) echo Identity of back-end nodes is not defined.
#
#           rc_failed 2
#           return 2 ;;
esac
fi

#[ -x /sbin/broker ] && status_of_proc /sbin/broker && {
#    echo Already Running 1.
#
#    #    rc_failed 1
#    #    return 1
#}
# status_of_proc /sbin/broker && {
#    echo Already Running 2.
#
#    #    rc_failed 1
#    #    return 1
#}
#/sbin/start-stop-daemon --start --exec /sbin/broker -- $args
#
#    rc_check
[ -x /sbin/broker ] && {
    echo Already Running.
#
#    rc_failed 1
#    return 1
}
}

start_backend() {
    [ -e /etc/vcl/is_back_end ] || return 0

#status_of_proc /sbin/opencld
#[ -x /sbin/opencld ] && {
#    echo Already Running.
#
#    rc_failed 1
}

```

```

#      return 1
#
#[ -x /sbin/opencld ] && status_of_proc /sbin/opencld &&
#
#{
#      echo Already Running.
#
#      rc_failed 1
#      return 1
#
#}
#
#status_of_proc /sbin/opencld && {
#      echo Already Running.
#
#      rc_failed 1
#      return 1
#
#}
/sbin/start-stop-daemon --start --exec /sbin/opencld
#
rc_check
if [ -x /sbin/opencld ]
then
      echo Already Running.
#
rc_failed 1
      return 1
else
      echo no corriendo
fi

}

#rc_reset

case "$1" in
start)
      echo "Starting MOSIX-VCL..."
      start_host
      start_backend
      ;;
stop)

```

```

echo "Stopping MOSIX-VCL..."
stop_host
stop_backend
;;
status)
any=n
[ -e /etc/vcl/is_host ] && {
    any=y
    status_of_proc /sbin/broker
    case $? in 0) echo Hosting-node is running. ;;
                  *) #rc_check
                     echo Hosting-node is not running. ;;
    esac
}
[ -e /etc/vcl/is_back_end ] && {
    any=y
    status_of_proc /sbin/opencld
    case $? in 0) echo Back-end is running. ;;
                  *) #rc_check
                     echo Back-end is not running. ;;
    esac
}
case $any in n) ;; esac #rc_failed 3 ;; esac
;;
restart|reload)
echo "Restarting MOSIX-VCL..."
stop_host
stop_backend
start_host
start_backend
;;
start_host)
if [ -e /etc/vcl/is_host ]
then
    echo "Starting MOSIX-VCL host..."

```

```

start_host
else
    echo "This is not a MOSIX-VCL hosting-node."
fi
;;
start_backend)
if [ -e /etc/vcl/is_back_end ]
then
    echo "Starting MOSIX-VCL back-end..."
    start_backend
else
    echo "This is not a MOSIX-VCL back-end node."
fi
;;
stop_host)
echo "Stopping MOSIX-VCL host..."
stop_host
;;
stop_backend)
echo "Stopping MOSIX-VCL back-end..."
stop_backend
;;
restart_host)
echo "Restarting MOSIX-VCL host..."
stop_host
if [ -e /etc/vcl/is_host ]
then
    start_host
else
    echo "This is not a MOSIX VCL hosting-node."
fi
;;
restart_backend)
echo "Restarting MOSIX-VCL back-end..."
stop_backend

```

```
if [ -e /etc/vcl/is_back_end ]
then
    start_backend
else
    echo "This is not a MOSIX VCL back-end node."
fi
;;
*)
echo "Usage: vcl {start|stop|status|restart|reload}"
echo " or  vcl {start_host|stop_host|restart_host}"
echo " or  vcl {start_backend|stop_backend|restart_backend}"
exit 1
esac

true
#rc_status -v
#rc_exit
```

## APENDICE V – Script de Automatización Nodo

```
#!/bin/bash
#
#Script de Post-Instalacion de nodos MOSIX
#Aplica cambios y configuracion a los nodos y server
#
if [ $# -lt 1 ]; then
    echo `sintaxis: ./asdf.sh numero_nodo "
    echo ` donde numero_nodo es un numero natural ej: 1, 2, 3 ..."
    exit 1
fi

echo "
"
echo ` ` "El Script procedera a realizar los cambios necesarios para integrar un nodo al Cluster"
echo ` ` "Espere unos segundos mientras el Script sigue la ejecucion..."
echo "
"
sleep 10
clear

echo ` ` "Se va a configurar el nodo"$1". Espere un momento porfavor..."
sleep 5

#edita sudoers
echo "
"
echo ` `">> Agregando el nodo a los usuarios ""sudo"""
sleep 3
echo '

nodo'$1'      ALL=(ALL) ALL
blender        ALL=(ALL) ALL
```

```

' >> /etc/sudoers

#edita interfaz de red
echo \\ ">> Configurando la Interfaz de Red"
sleep 3
cp interfaces /etc/network
echo '

#GeT
iface eth0 inet static
address 192.168.1.10'$1'
netmask 255.255.255.0
gateway 192.168.1.1

' >> /etc/network/interfaces

#añade archivos MOSIX para el cluster
echo \\ ">> Copiando archivos MOSIX al nodo\"$1\""
sleep 3
cp userview.map /etc/mosix
cp mosix.map /etc/mosix

echo \\ ">> Creando configuracion MOSIX para el nodo\"$1\""
sleep 3
echo '192.168.1.10'$1'
' >> /etc/mosix/mosip

#instala directorios NAS en el nodo
echo \\ ">> Instalando NAS Network Attached Storage"
mkdir /media/NAS
sleep 3
mkdir /media/NAS/Respaldo
mkdir /media/NAS/Documentos

```

```

mkdir /media/NAS/Videos
mkdir /media/NAS/Imagenes
mkdir /media/NAS/Musica
mkdir /media/NAS/Modelos
mkdir /media/cddvd

#edita fstab para NAS y cd dvd
echo \\">>> Editando fstab"
sleep 3
sudo sed -i '/cdrom0/d' /etc/fstab
echo '
#cd dvd
/dev/sdc0    /media/cddvd    udf,iso9660 user,noauto 0 0
#NAS
192.168.1.106:/nfs/Backups /media/NAS/Respaldo nfs rw 0 0
192.168.1.106:/nfs/Documents /media/NAS/Documentos nfs rw 0 0
192.168.1.106:/nfs/Movies /media/NAS/Videos nfs rw 0 0
192.168.1.106:/nfs/Pictures /media/NAS/Imagenes nfs rw 0 0
192.168.1.106:/nfs/Music /media/NAS/Musica nfs rw 0 0
192.168.1.106:/nfs/Modelos /media/NAS/Modelos nfs rw 0 0

' >> /etc/fstab
#añade usuario blender para el cluster
mount -a
echo \\">>> Añadiendo usuario blender"
sleep 3
adduser blender --/home /media/NAS/Documentos/blender

#añade nuevos repositorios a sources.list
echo \\">>> Añadiendo los nuevos Repositorios"
sleep 3
echo '
#Chromium Browser
deb http://ppa.launchpad.net/chromium-daily/ppa/ubuntu lucid main
deb-src http://ppa.launchpad.net/chromium-daily/ppa/ubuntu lucid main

```

```

#Blender
deb http://ppa.launchpad.net/cheleb/blender-svn/ubuntu maverick main
deb-src http://ppa.launchpad.net/cheleb/blender-svn/ubuntu maverick main

#BACKPORTS
deb http://backports.debian.org/debian-backports squeeze-backports main contrib non-free

#Non-Free
deb http://ftp.cl.debian.org/debian/ squeeze non-free contrib

' >> /etc/apt/sources.list

#instala keys para los nuevos repositorios
echo \\ ">> Instalando Keys de Repositorios"
sleep 3
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 108A879C
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 4E5E17B5

#actualiza repositorios
echo \\ ">> Actualizando Repositorios"
sleep 3
sudo aptitude update

#instala paquetes necesarios para trabajar en nodos
echo \\ ">> Instalando Paquetes"
sleep 3
sudo aptitude -t squeeze-backports install nautilus-dropbox
sudo aptitude install htop vim screen gparted blender chromium-browser netspeed guake gnome-do sun-java6-jre sun-java6-plugin fuseiso9660 flashplugin-nonfree filezilla mencoder ffmpeg ganglia-monitor
echo "
"
echo \\ ">>>> Se ha terminado de configurar el nodo\"$1\" <<<<"
```

sleep 3

## APENDICE VI – Pruebas de Rendimiento

### DistKeyGen

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <openssl/rsa.h>
//#include <iostream.h>
#include <string.h>
#include <time.h>

void keyGenetator(int fileIndex, int keyLength);

void keyGenerator(int fileIndex, int keyLength)
{
    int err=0;
    int size = 0;
    char * buffer = 0;
    char * buffer2 = 0;

    FILE * publicKeyFile = NULL;
    FILE * privateKeyFile = NULL;

    char fileName[256];
    char publicKeyFileName[512];
    char privateKeyFileName[512];

    for(int i = 0; i< 1000; i++)
    {
        if( i < 10)
        {
            sprintf(publicKeyFileName, "%d00%d-PublicKey", fileIndex, i);
```

```

        sprintf(privateKeyFileName, "%d0%d-PrivateKey", fileIndex,i);
    }
    else if(i >= 10 && i < 100)
    {
        sprintf(publicKeyFileName, "%d0%d-PublicKey", fileIndex, i);
        sprintf(privateKeyFileName, "%d0%d-PrivateKey", fileIndex,i);
    }
    else
    {
        sprintf(publicKeyFileName, "%d%d-PublicKey", fileIndex, i);
        sprintf(privateKeyFileName, "%d%d-PrivateKey", fileIndex,i);
    }

RSA * rsa;
rsa = RSA_generate_key(keyLength, RSA_F4, NULL, (char*)stdout);

int lenE = 0;
unsigned char * tmp = new unsigned char[keyLength];
unsigned char *p;

//public key
p = tmp;
lenE=i2d_RSAPublicKey(rsa,&p);

buffer = new char[lenE];
memcpy(buffer, tmp, lenE);
publicKeyFile = fopen(publicKeyFileName, "wb");
if (publicKeyFile == NULL)
    perror("publicKey");
fwrite(buffer, sizeof(char), lenE, publicKeyFile);
fclose(publicKeyFile);

//private key
p = tmp;
int lenD=i2d_RSAPrivateKey(rsa,&p);

```

```

        buffer2 = new char[lenD];
        memcpy(buffer2, tmp, lenD);
        privateKeyFile = fopen(privateKeyFileName, "wb");
        if (privateKeyFile == NULL)
            perror("PrivateKey");
        fwrite(buffer2, sizeof(char), lenD, privateKeyFile);
        fclose(privateKeyFile);

        RSA_free(rsa);
        delete [] buffer;
        delete [] buffer2;
        delete [] tmp;
    }

}

main()
{
    system("date");
    pid_t pid;
    int rv;
    time_t ltime;

    int i = 0;
    for (i = 0; i < 40; i++)
    {
        switch(pid=fork()) {
        case -1:
            perror("fork"); /* something went wrong */
            exit(1); /* parent exits */
        case 0:
            keyGenerator(i, 2048);
            printf("CHILD: I'm finished here! %d\n", getpid());
            system("date");
        }
    }
}

```

```

    exit(rv);

default:
    if(i == 2)
    {
        printf("PARENT: About to quit!\n");
    }
}

}

```

### ForKit

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

main()
{
    int rv;
    time_t ltime;
    pid_t pid;
    double x=123;
    long int i=0;
    long int j=0;

system("date");
    int h = 0;
    for (h = 0; h< 50; h++)
    {
        switch(pid=fork()) {
        case -1:
            perror("fork"); /* something went wrong */
            exit(1); /* parent exits */
        case 0:
            while(j<10000){

```

```
j++;
i=0;
while(i<1000000) {
    x=sin(x)*sqrt(x);
    i++;
}
system("date");
exit(rv);

default:
    if(h == 5)
    {
        exit(rv);
    }
}
}
```

### **Start ForKit**

```
#!/bin/bash
# running forkit 10 times
for ((LOOP=1; LOOP<=5; LOOP++))
do
echo "started forkit $LOOP. time"
mosrun -e ./forkit
./forkit
done
```

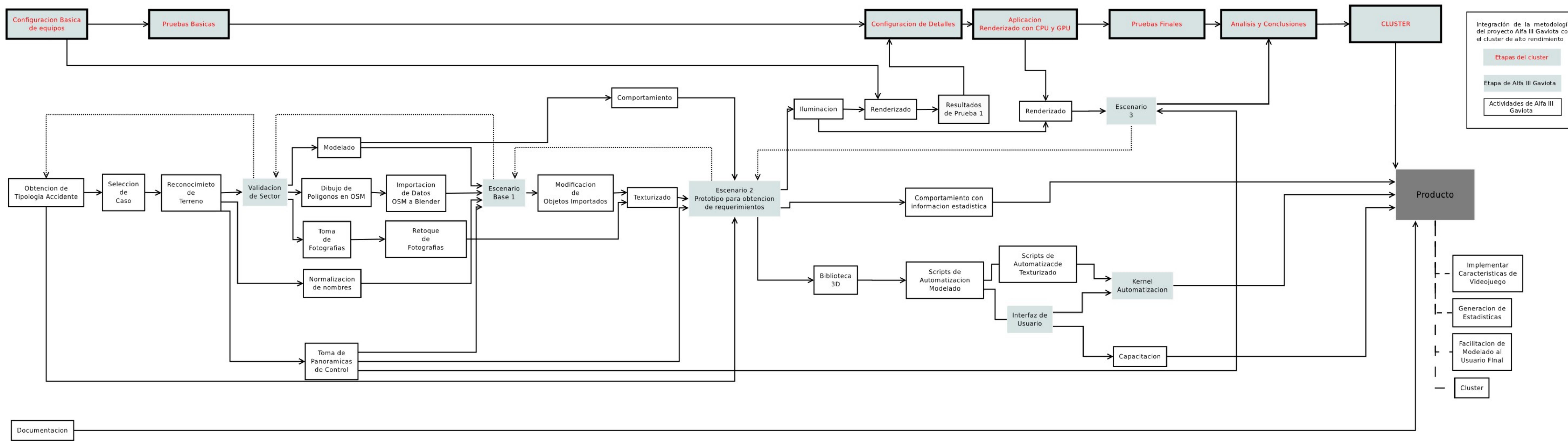
### **AWK**

```
#!/bin/bash
date; awk 'BEGIN {for(i=0;i<100000;i++)for(j=0;j<100000;j++)}' ;date
```

### **Start AWK**

```
#!/bin/bash
# running forkit 10 times
for ((LOOP=1; LOOP<=10; LOOP++))
do
echo "started forkit $LOOP. time"
mosrun -e ./script_awk.sh &
done
```

## Apendice VII - Diagrama del Proyecto Alfa III Gaviota



## Glosario

FLOPS	: (Floating Point Operations Per Second) Medida computacional para el rendimiento que un equipo. Generalmente medido utilizando el benchmark LINPAK
LINPAK	: Prueba de rendimiento que resuelve sistemas de ecuaciones lineales
HPC	: (Hight Performance Computing/Computers) Computación/Computador de alto rendimiento
NAS	: (Network Attached Storage) Almacenamiento de red mediante algún protocolo
MIDDLEWARE	: Aplicación que facilita la comunicación entre dispositivos en un sistema distribuido
TOP500	: Web especializada en computación de alto rendimiento, la cual genera un rating de los supercomputadores mas rápidos del mundo
CPU	: (Central Proccess Unit) Unidad de procesamiento central, se encarga de hacer los calculos necesarios en un computador.
GPU	: (Graphics Proccess Unit) Unidad de procesamiento gráfico, especializada en procesos gráficos.
GPGPU	: (General-Purpose Computing on Graphics Processing Units) Computación de propósito general en unidades de procesamiento gráfico.
PVM	: (Parallel Virtual Machine) Maquina virtual paralela. Conjunto de herramientas y librerías que emulan un entorno de propósito general para procesamiento paralelo.
MPI	: (Message Passing Interface) es una especificación estándar para una librería de funciones de paso de mensajes.
TCP/IP	: (Transmission Control Protocol/Internet Protocol) protocolo que se encarga de la comunicación entre hosts sobre una red establecida.
UDP/IP	: (User Datagram Protocol/Internet Protocol) similar a TCP/IP pero proporciona una manera directa de enviar y recibir datagramas sobre una red IP
RSA	: (Rivest, Shamir y Adleman) es un sistema criptográfico de clave pública

# Bibliografía

## Literatura

Mora Hinojosa, Pablo. Implementación de un cluster en ambiente FREEBSD y su integración con ambiente GNU/LINUX utilizando MPI. Chile, Universidad del Bío-Bío ,  
2007

Johnson, Sandra K. Performance tuning for LINUX servers. 2005

A. Barak y A. Shiloh. The MOSIX Management System for Linux Clusters, Multi-CLusters, GPU Clusters and Clouds.

A. Barak, La'adan y A. Shiloh. Scalable Cluster Computing whith MOSIX for Linux. Proc. 5Th Annual Linux Expo, Raleigh, NC, pp. 95-100, 1999.

Miquel Catalán i Coït *El manual para el clustering con openMosix*, Copyright c miKeL a.k.a.mc2 & Kris Buytaert. Versión 1.0 - 6 de Septiembre de 2004

Barak A., Ben-Num T., Levy E., and Shiloh A., A Package for OpenCL Based Heterogeneous Computing on Clusters with Many GPU Devices. Proc. Workshop on Parallel Programming and Applications on Accelerator for Clusters (PPAAC10), IEEE Cluster 2010, Crete, Sep. 2010.

A. Munshi, B. Gaster, T. Mattson, J. Fung, D. Ginsburg. OpenCL Programming Guide. Estados Unidos, Pearson Education. 2011

A. Barak and A. Shiloh, The MOSIX management system for Linux cluster, multi-clusters, GPU clusters and Clouds, <http://www.mosix.org/pub/MOSIX wp.pdf>.

Amnon Barak and Amnon Shiloh. The Virtual OpenCL (VCL) Cluster Platform. VCL, <http://www.mosix.org/txt vcl.html>.

Nvidia. NVIDIA\_OpenCL\_ProgrammingGuide. NVIDIA, [www.nvidia.com](http://www.nvidia.com).

## **Web**

Chilehardware.cl. OpenCL llega a ser la 2º herramienta más usada en cómputo HPC

[en línea] <http://www.chw.net/2011/10/opencl-llega-a-ser-la-2%C2%BA-herramienta-mas-usada-en-computo-hpc/> [octubre 2011]

openMosix. The openMosix Stress Test

[en línea]<<http://www.openmosixview.com/omtest/#down>> [enero 2010]

wiki.blender.org. GPU Rendering

[en línea] [http://wiki.blender.org/index.php/Doc:2.6/Manual/Render/Cycles/GPU\\_Rendering](http://wiki.blender.org/index.php/Doc:2.6/Manual/Render/Cycles/GPU_Rendering) [julio 2011]

3dwordmag.com. Pros and cons of GPU-accelerated rendering

[en línea] <http://www.3dworldmag.com/2011/01/07/pros-and-cons-of-gpu-accelerated-rendering/> [julio 2011]

codeproject.com. OpenCL™ – Portable Parallelism

[en línea]<<http://www.codeproject.com/KB/showcase/Portable-Parallelism.aspx>> [septiembre 2010]

thebigblob.com. Getting started with OpenCL and GPU Computing

[en línea]<<http://www.thebigblob.com/getting-started-with-opencl-and-gpu-computing/>> [enero 2011]

publiespe.espe.edu.ec. Artículos de interés

[en línea] <http://publiespe.espe.edu.ec/articulos/sistemas/arquitectura/arquitectura.htm> [marzo 2010]

josecc.net. Implementación de un cluster openMosix para computo científico en el instituto de ingeniería UNAM

[en línea] [http://www.josecc.net/archivos/tesis/tesis\\_html1/node5.html](http://www.josecc.net/archivos/tesis/tesis_html1/node5.html) [agosto 2006]

tutorialesdebian.com. [Recompilar el kernel de linux \(2.6.x +\) en debian y ubuntu](#)

[en línea] <http://www.tutorialesdebian.com/2010/04/recompilar-el-kernel-de-linux-2-6-x-en-debian-y-ubuntu/> [abril 2010]

mosix.org. Mosix Cluster Operating System  
[en línea] <http://www.mosix.org/> [1999- 2012]

mirza07.wordpress.com. How to Install MOSIX on Linux kernel version 2.6  
[en línea]<<http://mirza07.wordpress.com/2009/10/10/how-to-install-mosix-on-linux-kernel-version-2-6-28/>> [octubre 2009]

wikilearning.com. Instalando un Cluster tipo Mosix  
[en línea]<[http://www.wikilearning.com/tutorial/instalando\\_un\\_cluster\\_tipo\\_mosix/442-6](http://www.wikilearning.com/tutorial/instalando_un_cluster_tipo_mosix/442-6)> [octubre 2009]

mosix.org. Virtual OpenCL VCL Cluster Platform  
[en línea] [http://www.mosix.org/txt\\_vcl.html](http://www.mosix.org/txt_vcl.html) [2009-2012]