Universidad Tecnológica de Bolívar

Computación de Alto Desempeño: Entorno de ejecución paralela basado en procesadores de arquitectura ARM

Autor:
Carlos Puello

Revisor:
Msc. Jairo Serrano

Tésis presentada en cumplimiento parcial de los requisitos para obtener el título de Ingeniero en Sistemas

de la

Facultad de Ingeniería

Agosto 2015

UNIVERSIDAD TECNOLÓGICA DE BOLÍVAR

Resumen

Facultad de Ingeniería
Programa de Ingeniería de Sistemas

Computación de Alto Desempeño: Entorno de ejecución paralela basado en procesadores de arquitectura ARM

por Carlos Puello

En el siguiente trabajo de investigación se describe el diseño y se demuestra la implementación de un entorno de ejecución paralela utilizando como hardware, ocho nodos esclavos con procesadores de arquitectura ARM que se encuentran ensamblados en los ordenadores tipo SoC Raspberry Pi, para luego integrarlo al cluster del HPCLab de la Universidad Tecnológica de Bolívar mediante el uso de software libre y el gestor de colas de trabajo HTCondor. A este entorno se le realizan pruebas de rendimiento con la herramienta HPL (High Performance Linpack) y pruebas de funcionamiento a través del envio de trabajos con el gestor de colas HTCondor.

En este documento también se explican los componentes y sus funciones, se muestra como instalar y configurar el hardware y software utilizado para el funcionamiento óptimo y correcto del sistema de tal forma que sirva de guía para el lector.

UNIVERSIDAD TECNOLÓGICA DE BOLÍVAR

Abstract

Facultad de Ingeniería
Programa de Ingeniería de Sistemas

Computación de Alto Desempeño: Entorno de ejecución paralela basado en procesadores de arquitectura ARM

by Carlos Puello

The following research describes the design and demonstrate the implementation of a parallel execution environment using eight slave nodes with ARM architecture processors that are assembled in Raspberry Pi SoC computers, for later add to the HPCLab cluster of the Tecnologica de Bolívar University through the use of free and open source software and the job manager HTCondor. To this environment benchmark tests are done with the tool HPL (High Performance Linpack) and performance tests through the jobs sending with HTCondor.

In this thesis explain the components and its functions too, it shows how install and configure the hardware and software used for the right and optimal performance of the system so that serves as a guide for the reader.

Agradecimientos

Inicialmente agradezco la realización de este trabajo de investigación a *Dios todo poderoso* que me observa desde lo más alto del cielo, por iluminarme el buen camino, por darme esa fortaleza, perseverancia y sabiduría que me llevaron a lo que considero fue una *victoria* de las mil batallas que me faltan por enfrentar.

Agradezco también a mi tutor MSc Jairo Enrique Serrano, quien lastimosamente por cosas de la vida no fue profesor en alguna de las asignaturas que recibí durante el curso de la carrera, que fue la persona que con sus grandes virtudes y sabiduría, aportó aquellos conocimientos claves para el éxito de esta.

A mi familia por brindarme todo su apoyo incondicional, a mis compañeros de clase y a aquellos profesores que contribuyeron y añadieron ese grano de conocimiento para el éxito de mi carrera: *Juan Salvador*, *Efrain Herrera*, *Juan C. Palomino*, *Isaac Zuñiga*, *William Caicedo*, *Giovany Vasquez*, entre otros.

Contenido

Lista de Figuras

Lista de Tablas

Abreviaturas

CISC	**C**omplex **I**nstruction **S**et **C**omputer
CPU	**C**entral **P**rocessing **U**nit
FLOPS	**F**loating **P**oint **O**peration per **S**econd
HPC	**H**igh **P**erformance **C**omputing
HPL	**H**igh **P**erformance **L**inpack
HTC	**H**igh **T**hroughput **C**omputing
RISC	**R**educed **I**nstruction **S**et **C**omputer

Dedico esta tésis a estas personas que cambiaron este mundo:

Nikola Tesla

Alan Turing

J. Bardeen, W. H. Brattain y W. B. Shockley

Dennis Ritchie

Tim Berners-Lee

Serguéi Brin y Larry Page

mi Vicky Vicky ;)

y otros...

Capítulo 1

Descripción del Proyecto

Contenido

1.1 Introducción

En las tres últimas décadas se ha podido apreciar el gran adelanto tecnológico de las computadoras, destacándose las grandes unidades de almacenamiento y las velocidades de cómputo y comunicación. Sin embargo, a medida que la computación avanza o evoluciona, la solución de problemas se torna más complicado, ya que van apareciendo modelos más complejos, con mayor cantidad de datos y tiempos de respuestas más limitados. Afortunadamente, los entornos de ejecución paralela, como es el caso de los clúster HPC y HTC, han venido a relucir para satisfacer todos estos requisitos de poder de cómputo, por medio de las tecnologías de la computación distribuida y la programación en paralelo.

En la mayoría de los casos, el hardware que se utiliza para estos entornos, emplea procesadores basados en arquitecturas CISC (x86, x64) o RISC (PowerPC,

SPARC), los cuales fueron diseñados para servidores, computadoras de escritorios y computadores portátiles [1]. Estos procesadores ofrecen un buen rendimiento y desempeño pero desafortunadamente en algunos casos son muy costosos y requieren de mucha potencia energética sobre todo cuando son utilizados en grandes cantidades, generando así altas temperaturas que se deben controlar utilizando sistemas de enfriamiento los cuales también necesitan de mucha energía eléctrica. No obstante, gracias a las nuevas tecnologías de hardware aplicados en los smartphones, tabletas y otros dispositivos móviles, se han venido desarrollando ordenadores tipo SoC (System on Chip) –*también conocidos como single boards*– que emplean procesadores de arquitectura ARM (basados en arquitectura RISC), los cuales gracias a su diseño permite tener un buen rendimiento y desempeño en aplicaciones específicas consumiendo poca energía y a un bajo costo [2]. Actualmente, uno de los ordenadores tipo SoC más famosos y utilizados para la programación y aprendizaje de software es la Raspberry Pi, gracias a sus buenas prestaciones y precio accesible; por lo tanto, se ha convertido en una buena herramienta para explorar la computación paralela ya sea para el aprendizaje en el hogar o la enseñanza e investigación en las universidades [3].

Los procesadores ARM también han comenzado a tener un lugar en el mercado de las computadoras personales, laptops y servidores. Grandes compañías como la SAMSUNG ya ofrecen laptops con procesadores ARM, como la ARM Series 3 Chromebook(XE303C12) la cual emplea el procesador Exynos 5250 (1.7GHz, 1MB L2 Cache). Debido a que la tendencia de los procesadores ARM es que sean más potentes en términos de rendimiento y sigan manteniendo un precio y consumo de energía bajo, este escenario para el mercado de los sistemas de computación de alto desempeño presenta una gran oportunidad para utilizar estos procesadores. Tanto es así, que en Europa, la BSC (Barcelona Supercomputing Center) coordina el proyecto Mont-Blanc[4] en el cual planean utilizar CPUs ARM combinados con GPU para lograr un desempeño de cómputo exaescala.

En esta investigación se describe la implementación de un clúster de alto desempeño utilizando ordenadores tipo SoC con procesadores ARM y software libre. Debido al costo, popularidad y facilidad de adquisición se utilizarán los ordenadores Raspberry Pi adquiridos por el Laboratorio de Computación de Alto Desempeño (HPCLab) de la UTB. Posteriormente una vez realizada la implementación del cluster con el framework HTCondor, se integrará todo el conjunto

al HPCLab de la UTB ya sea para nuevas investigaciones o aprendizaje de los estudiantes. Una vez terminada la integración de las RaspberryPi con el HPCLab, se realizarán pruebas de funcionamiento y rendimiento, junto con una guía con instrucciones y recomendaciones para el uso óptimo del clúster.

El presente trabajo se encuentra organizado de la siguiente manera:

En el capítulo 2 se realiza un estado del arte de que lo que se ha hecho y se está haciendo respecto a la computación de alto desempeño y entornos de ejecución paralela con procesadores de arquitectura ARM. De igual forma, en este mismo capítulo se redacta la teoría básica y se definen conceptos referente al tema de la computación de alto desempeño, procesadores ARM y el gestor de colas HTCondor.

En el capítulo 3 se establece la metodología aplicada para cumplir los objetivos propuestos, desarrollar la investigación y obtener los productos de la misma.

En el capítulo 4 se realiza el diseño y la implementación del clúster. Se describen los pasos y procedimientos para la configuración de las Raspberry Pi y la instalación y configuración del gestor de colas HTCondor.

En el capítulo 5 se realiza una prueba de funcionamiento del cluster y otra prueba empírica de rendimiento utilizando la herramienta de software HPL (High Performance Linpack), y en el capítulo 6 se establecen los resultados, experiencias, análisis y resultados finales de dichas pruebas.

Finalmente, en el capítulo 7 se describen las conclusiones del proyecto realizado.

1.2 Objetivos

1.2.1 Objetivo General

Implementar un entorno de ejecución paralela en el Laboratorio de Computación de alto desempeño (HPCLab) de la UTB, utilizando CPU's de arquitectura ARM y herramientas de software libre.

1.2.2 Objetivos Específicos

- Realizar un estado del arte respecto al hardware y software dentro del ámbito de los entornos de ejecución paralela basados en procesadores de arquitectura ARM.
- Implementar el entorno utilizando nodos ARM (Raspberry Pi) e integrarlo al HPCLab mediante el framework HTCondor.
- Realizar pruebas de funcionamiento y rendimiento utilizando el gestor de colas HTCondor y la herramienta HPL.
- Elaborar documentación en donde se evidencien los procedimientos para la implementación del entorno y su uso óptimo.

1.3 Contexto y Descripción del Problema

En los últimos años, el gran avance tecnológico ha venido evolucionando de forma drástica, permitiendo así que se construyan ordenadores de gran capacidad de procesamiento. Gracias a esta capacidad, se pueden lograr grandes estudios de investigación como simulaciones, análisis de datos, procesamiento de imágenes, física computacional, virtualización, entre otros, aprovechando las tecnologías de la programación en paralelo y la computación distribuida. Unas de las principales arquitecturas que más se utilizan para estos propósitos, son la x86 y x64 [1]. La desventaja de estas arquitecturas frente a otras es su precio y el consumo energético, por tal razón, se planteó la problemática de *¿cómo implementar un entorno de ejecución paralela utilizando procesadores de arquitectura ARM?*, ya que, aunque estos tienen una capacidad de procesamiento inferior a los de arquitectura x86_64, poseen la ventaja de tener un consumo energético y precio inferior a otras arquitecturas. Por otra parte, en el HPCLab de la Universidad Tecnológica de Bolívar, se utiliza el gestor de colas HTCondor, por tal motivo, se despertó el interés en demostrar *¿cómo este framework se puede implementar en un clúster de alto desempeño utilizando Raspberry Pi para luego integrarlo al HPCLab?*, de tal manera que le sirva a los estudiantes para el aprendizaje o investigaciones en el área de la computación de alto desempeño y la programación paralela.

1.4 Justificación

Gracias al gran desarrollo e innovación que se está presentando en el hardware de los dispositivos móviles, se está utilizando este mismo hardware para la fabricación de computadoras personales y ordenadores de tarjetas para diferentes propósitos, como robótica, domótica, sistemas de entretenimiento e incluso para la enseñanza de ciencias de la computación, debido a su baja complejidad, tamaño y precio de producción de los procesadores que estos contienen: *procesadores de arquitectura ARM*. Tantas son las buenas propiedades y prestaciones de estos procesadores que la industria del hardware y software del mundo de la computación de alto desempeño, ya están diseñando y desarrollando hardware para este campo. Por lo anterior, desde el punto de vista tecnológico, con este proyecto de investigación se demostrará si se puede implementar utilizando el gestor de colas HTCondor un clúster heterogéneo, no solamente a nivel de sistema operativo y de capacidad de memoria aleatoria de los nodos, sino a nivel de arquitectura de los procesadores de dichos nodos, esto quiere decir, que si el clúster puede contener nodos que funcionan con procesadores x86, x64 y ARM.

Uno de los beneficios de esta investigación, es que le proveerá al lector interesado en la computación de alto desempeño, una guía con conceptos y procedimientos para que amplíe o tenga el debido conocimiento en cómo implementar utilizando software libre, un clúster real(no con máquinas virtualizadas) con la ventaja de ser económico gracias al bajo precio y versatilidad de las Raspberry Pi y su procesador de arquitectura ARM. De igual forma, le permitirá a los estudiantes o investigadores del HPCLab de la UTB tener la oportunidad y experiencia implementando clúster heterogéneos teniendo en cuenta de no poner en riesgo la seguridad ni la infraestructura computacional del HPCLab, ya que no sería necesario el acceso físico a las instalaciones y cableado del servidor SPIDER, debido a la portabilidad y fácil instalación de las Raspberry Pi.

Capítulo 2

Marco Teórico

Contenido

2.1 Estado del Arte

La utilización de SoC's(System on Chip) con procesadores de arquitectura ARM en clústers de computación de alto desempeño, se ha reportado en la literatura. Rajovic et al. [5] construyeron un prototipo del primer cluster HPC a gran escala

utilizando procesadores ARM multinúcleo, a este cluster lo llamaron "Tibidabo"; conformado por 128 nodos los cuales cada uno emplea un ordenador tipo SoC Nvidia Tegra2, integrado por un procesador dual-core ARM Cortex-A9 de 1GHz, un módulo Q7 con 1 GB de memoria RAM, 16GB de almacenamiento eMMC y una NIC Ethernet de 1Gb/s. En el prototipo implementaron la librería MPICH2 v1.4.1 y el administrador de trabajos(job-worker) SLURM; el objetivo de este proyecto fue realizar una evaluación de eficiencia energética, en el cual alcanzaron los 120MFLOPS/W , logrando así que se igualara a clusters basados en los procesadores Intel Xeon X5660 y AMD Opteron 6128, los cuales en la edición de noviembre de 2012 de la lista Green500, ocuparon la posición 395 y 396 respectivamente.

Sukaridhoto et al [6], construyeron un cluster de bajo consumo energético para la enseñanza y práctica de los estudiantes. A nivel de hardware el cluster lo implantaron con 16 nodos, cada uno utilizando la tarjeta SoC Pandaboard con procesador dual-core ARM Cortex-A9 MPcore de 1.2GHz, 1GB de memoria RAM DDR2, 8GB de almacenamiento SD y una NIC Ethernet de 100Mbps y el sistema operativo Debian Linux 3.7. Para la programación emplearon la librería OpenMPI y para las pruebas de rendimiento las librerías PMB(Pallas MPI Benchmark) y HPL(High Performance Linkpack).

Por otra parte, empresas del sector de fabricación de semiconductores y hardware para computadoras, ya están tomando la tendencia en fabricar productos con estándares para cluster basados en procesadores ARM. En 2014 las empresas IDT[7], Orange Silicon Valley y NVIDIA diseñaron y desarrollaron un cluster utilizando procesadores NVIDIA Tegra K1 y la tecnología de interconexión IDT RapidIO. El sistema es escalable para conectar más de 2000 nodos en un rack, alcanzando 23TFLOPS por 1U y 800TFLOPS en el rack completo.

En este mismo año la compañía de fabricación de procesadores y tarjetas gráficas AMD, pone a la venta un kit de desarrollo para servidores compatible con el factor de forma microATX para que sea utilizado en centros de datos. La tarjeta tiene ensamblado un procesador de 64 bits Opteron A1100(de cuatro u ocho núcleos) basado en la familia ARM Cortex-A57, de 4MB y 8MB de memoria caché L2 y L3 respectivamente. En el lanzamiento de este kit, se demostró la ejecución del framework Apache Hadoop para procesamiento distribuido de grandes conjuntos de datos por medio de cluster de ordenadores.

Para el caso de las tarjetas Raspberry Pi, también se han implementado clusters de alto desempeño [8, 9] pero utilizan las librerías OpenMPI y MPICH; para el caso del framework HTCondor, no se encontró ninguna publicación oficial que reportara el uso de este framework en estas tarjetas.

2.2 Clúster de Computadores

Un cluster es un sistema distribuido de procesamiento paralelo el cual está conformado por 2 o más computadoras independientes llamadas nodos, que trabajan conjuntamente para la solución de un problema. La ventaja principal de estos sistemas es que permiten una solución flexible, de bajo coste(en comparación con las supercomputadoras y mainframes) y de gran escalabilidad para aplicaciones que requieran una elevada capacidad de cómputo y memoria[10].

Las características fundamentales de un cluster son las siguientes:

- Todo el clúster es visto por el usuario como un solo sistema.
- Cada nodo necesita por lo menos un elemento de proceso(CPU), memoria y una interfaz(NIC) para comunicarse con la red del cluster.
- Necesitan software especializado.
- Escalabilidad, si el sistema operativo lo permite, sólo hace falta conectar más computadoras a la red del cluster y configurarlas para obtener una mejora en el rendimiento y disponibilidad del sistema. De igual forma, tiene que poder detectar automáticamente nuevos nodos conectados para proceder a su utilización.
- La tecnología de cluster de servidores por balanceo de carga mejora la respuesta a las peticiones conmutando estas entre los diversos nodos del cluster.

Los componentes básicos de un cluster son:

- **Nodos:** pueden ser simples computadoras, sistemas multiprocesador o estaciones de trabajo. Estos nodos pueden conectarse mediante una simple red Ethernet, Fast Ethernet, Gigabit Ethernet o por medio de tecnologías especiales de alta velocidad como Myrinet, Infiniband, SCI.

- **Sistemas Operativos:** tienen que ser de fácil uso y acceso, y además, permitir múltiples procesos y usuarios. Middleware: es un software que generalmente actúa entre el sistema operativo y las aplicaciones con el fin de proveer una interfaz única de acceso al sistema, denominada SSI(Single System Image), la cual permite generar al usuario la sensación que utiliza una única computadora.

- **Herramientas para la optimización y mantenimiento del sistema:** migración de procesos, checkpoint-restart(parar uno o varios procesos, migrarlas a otro nodo y continuar su funcionamiento), balanceo de cargas, tolerancia a fallos, etc.

- **Ambientes de programación paralela:** permiten implementar algoritmos que hacen uso de recursos compartidos: CPU, memoria, datos y servicios.

2.2.1 Clasificación de los Cluster

Según [2], los clúster se pueden clasificar con base a diferentes criterios y factores como se indican a continuación:

- En términos de funcionamiento, los clusters pueden ser de Alto Desempeño (HPC-High Performance Computing), Alta Disponibilidad (HAC-High Availability Computing), Alto Rendimiento o Productividad (HTC High Throughput Computing) o de Balanceo de Cargas(LB-Load Balancing).

- En términos de los nodos, los clusters pueden ser clasificados como dedicados y no-dedicados. En el caso de los dedicados, todos los nodos son dedicados únicamente a ejecutar procesos y tareas del cluster. En el caso de los no-dedicados, los nodos son usados independientemente por el respectivo usuario y simultáneamente realizan o ejecutan procesos o tareas del cluster.

- En términos de la configuración del nodo, pueden ser homogéneos o heterogéneos. En el caso de los clusters homogéneos, todos los nodos tienen arquitectura similar y el mismo sistema operativo. en los clusters heterogéneos, los nodos tienen arquitecturas diferentes y poseen diferentes sistemas operativos.

2.3 Computación de Alto Desempeño

La computación de alto desempeño o HPC (siglas en inglés de High Performance Computing) hoy en día es una de las herramientas principales para el desarrollo de la ciencia, se aplica para satisfacer las necesidades de poder de cómputo el cual se requiere para la simulación y modelado de fenómenos físicos como el cambio climático, la producción de energía, diseño de fármacos, diseño de materiales; el análisis de conjuntos de datos de gran tamaño o volumen, como los de la secuencia del genoma humano, las observaciones astronómicas; y para el diseño de productos que van desde el ensamblado de dispositivos electrónicos hasta el diseño de aviones.

A principio de los años 60's se comenzó a utilizar el término supercomputadora gracias a las computadoras *CDC 6600* que operaba a una velocidad de 9 MFLOPS(millones de operaciones aritméticas en coma flotante por segundo) y a la *ILLIAC IV* que fue la primera computadora en usar un diseño paralelo (no Von Neumann), la cual tenía 64 computadores escalares idénticos operando en paralelo. A partir de aquí, el avance y las mejoras en el hardware han sido constantes y notables, y es donde empiezan a aparecer los sistemas de cómputo de alto desempeño sin que aún se conocieran bajo este nombre.

2.3.1 Arquitecturas de Computación Paralela

La arquitectura de una computadora paralela es aquella que cuenta con varias unidades de procesamiento y permite el procesamiento paralelo, ocasionando así, que se puedan realizar diversas configuraciones y obtener distintas arquitecturas paralelas. La disposición de las conexiones entre los elementos de procesamiento, el uso de la memoria, etc., determina el origen de distintas arquitecturas, teniendo cada una de las plataformas de HPC características, limitaciones y bondades específicas. A continuación se presentan algunas de las taxonomías más divulgadas.

Flynn [11] consideró para su clasificación dos aspectos: el flujo de instrucciones y el flujo de datos, considerando la multiplicidad de cada uno, presentó la siguiente clasificación:

- SISD (sigla en inglés de *Single Instruction Single Data*): Esta categoría representa la clásica máquina de Von Neumann. Se encuentran todas las

computadoras con una única CPU en las cuales, las instrucciones se ejecutan secuencialmente.

- SIMD (sigla en inglés de *Single Instruction Multiple Data*): Estas arquitecturas cuentan con varias CPU, donde todas ejecutan el mismo flujo de instrucciones sobre distintos datos.

- MISD (sigla en inglés de *Multiple Instruction Single Data*): En esta categoría, las CPU's reciben diferentes instrucciones, las cuales operan sobre el mismo flujo de datos.

- MIMD (sigla en inglés de *Multiple Instruction Multiple Data*): En este caso, las unidades de procesamiento pueden ejecutar simultáneamente diferentes instrucciones sobre distintos conjuntos de datos trabajando asíncronamente.

Otra taxonomía existente y muy utilizada en arquitecturas con múltiples unidades de procesamiento [12] se basa en la organización del sistema de memoria, dejando como resultado las siguientes categorías:

- Multiprocesadores de memoria compartida: todos los procesadores comparten y tienen acceso a la memoria del sistema paralelo.

- Multiprocesadores de memoria distribuida: cada unidad de procesamiento posee una memoria local y se comunica con el resto por medio de mensajes enviados a través de la red.

Debido a que la clasificación de Flynn no puede reunir a todos los sistemas dentro de sus cuatro categorías, por ejemplo, los procesadores vectoriales y las arquitecturas híbridas, actualmente otra taxonomía ampliamente extendida propone agrupar los computadores paralelos en base a grandes clases de arquitecturas [13, 14]. A continuación se describen las cinco categorías:

- Multiprocesadores Simétricos (SMP, siglas en inglés de *Symmetric Multiprocessor*): son equipos con varias CPU's muy acopladas(comparten gran cantidad de los recursos incluida la memoria). También se les conoce como sistemas de grano grueso(coarse-grain).

- Equipos con acceso a memoria no uniforme (NUMA, siglas en iglés de *Non-Uniform Memory Access*): están compuestos por varias unidades de procesamiento, cada uno de los cuales posee su propia memoria. La característica particular de esta clase de computadores es que a pesar de que cada CPU posee su propia memoria, se dispone de un direccionamiento de memoria global.

- Computadores Masivamente Paralelos (MPP, siglas en inglés de *Massively Parallel Processing*): son equipos con varias unidades de procesamiento poco acopladas, en los que cada procesador accede a su propia memoria. También suelen denominarse sistemas de grano fino(fine-grain). Clusters de PC's: están constituidas por un conjunto de computadoras personales (PC's) interconectados mediante una red de datos.

- Grids: se constituyen de un conjunto de equipos autónomos conectados mediante una red. En este caso los equipos pueden ser heterogéneos en cuanto arquitectura, velocidad, sistema operativo, e incluso pertenecer a distintas redes ubicadas geográficamente.

2.3.2 Programación Paralela

Para la implantación de un cluster de alto rendimiento, es necesario que la aplicación a ejecutar sea paralela, sin embargo, existen diversas técnicas de programación paralela, cada una de las cuales se adapta mejor a las características de un tipo específico de arquitectura paralela. Debido a que la programación paralela se basa en el paradigma *divide-and-conquer*, el problema se divide en subproblemas para que sean resueltos simultáneamente en las diferentes unidades de procesamiento de forma cooperativa [15]. Existen dos estrategias principales para efectuar dicha división e incluso la combinación de ambas:

- Descomposición funcional: en esta estrategia se dividen las distintas tareas a realizar por el algoritmo entre las diferentes unidades de procesamiento. Principalmente lo que se busca con esta estrategia es resolver problemas de gran dimensión y que requieren gran poder de cómputo en el menor tiempo posible. Este tipo de aplicaciones son las que se ejecutan en los entornos HPC.

- Descomposición de datos: en este caso todas las unidades de procesamiento disponibles ejecutan las mismas tareas, pero cada una trabaja sobre un subconjunto de los datos del problema, de tal forma, que en esta estrategía o paradigma se prioriza la ejecución de la mayor cantidad posible de tareas sin importar el tiempo que tomen en finalizar su ejecución. A este tipo de aplicaciones se les conoce como embarazosamente paralelas (del inglés *embarrassingly parallel*) [9] y son las que se ejecutan en los entornos HTC.

De igual forma como ocurre con las estrategias de división de problemas, las distintas técnicas de programación paralela se encuentran muy unidas al hardware, y en particular, a la forma en que las unidades de cómputo se comunican entre sí y comparten información. A continuación se presentan las técnicas o paradigmas de programación paralela más difundidas y su relación con las diferentes arquitecturas:

- Memoria Compartida: este modelo es típicamente utilizado sobre multiprocesadores de memoria compartida en donde existen varios procesadores interconectados que pueden compartir un mismo sistema de memoria [16]. Los programas paralelos desarrollados bajo esta técnica, se descomponen en varios procesos que comparten los datos asociados a una porción de su espacio de direcciones. La coordinación y cooperación entre los procesos se realizan a través de la lectura y escritura de variables compartidas y a través de variables de sincronización. Varias librerías han sido desarrolladas siguiendo este paradigma, como el caso de OpenMP y ante el gran auge de las tarjetas gráficas con GPU de procesadores paralelos aparecieron herramientas como CUDA [17].

- Pase de Mensajes: esta técnica consiste en un grupo de procesos que se comunican y coordinan entre sí por medio del intercambio explícito de mensajes para realizar una tarea determinada. El mensaje es originado en un sitio (la unidad de procesamiento que envía), luego es transmitido a través de la red de interconexión, y recibido en otro lugar (la unidad de procesamiento receptora); por lo anterior, en estos sistemas cada procesador debe conocer su dirección y la del resto de procesadores de la red, poder crear y enviar mensajes, poder recibir mensajes, eliminar los datos del mensaje y manipularlos; y tener acceso directo sólo a su memoria local, siendo indirecto el acceso al resto de memorias de los demás procesadores. Para la implantación de estos

sistemas, la literatura ha reportado el uso de estándares y librerías que contienen rutinas optimizadas las cuales ocultan los detalles de comunicación. Las librerías más utilizadas son MPI (siglas en inglés de *Message Passing Interface*) y PVM (siglas en inglés de *Parallel Virtual Machine*) [18].

Según [19–21] es posible una combinación de ambos paradigmas, es decir se puede desarrollar una aplicación donde se aplique el pase de mensajes para comunicar los procesos en diferentes computadoras y dentro de ellas aplicar el paradigma de memoria compartida.

2.4 Procesadores ARM

La familia de procesadores ARM (siglas en inglés de *Advanced RISC Machine*) se basan en la arquitectura RISC (siglas en inglés de *Reduced Instruction Set Computer*) con el propósito de aprovechar las propiedades de ser simples, con buen poder de cómputo y de bajo coste de fabricación. Estos procesadores fueron desarrollados a comienzo de los años 80's por la empresa ACRON Computers, la cual, en ese entonces fabricaba sus propios procesadores para usarlas en sus computadoras. Por el anterior hecho, en ACORN notaron que los clientes podrían no interesarles sus computadoras, de tal forma que decidieron crear en el año 1990 la compañía ARM Holdings para que se encargara del diseño y gestión de las nuevas generaciones de procesadores ARM, llegando al punto que la compañía ARM no fabrica procesadores(chips), sino que brinda las licencia de los diseños de la arquitectura a otras compañías para que estas si los fabriquen ya sea aprovechando las tecnologías de fabricación SoC (siglas en inglés de *System on Chip*) para que integren componentes como memoria RAM, GPU's, controladoras de red, etc., o fabricar microcontroladores de propósito específico. El primer procesador ARM fue lanzado en el año 1985 como un prototipo llamado ARM1, el cual carecía de instrucciones de multiplicación y de co-procesador. Posteriormente fue lanzado de manera comercial el ARM2 que se caracterizaba por tener un bus de datos de 32 bits, un espacio de direccionamiento de 26 bits, un conjunto de 16 registro y carecer de memoria caché como si lo tenía su sucesor ARM3.

2.4.1 Arquitectura de los Procesadores ARM

Los procesadores ARM son diseñados en base a la arquitectura RISC, la cual, partía de la idea de intentar disminuir el tiempo de ejecución al reducir el conjunto de instrucciones de la CPU. Las principales características de un procesador RISC son:

- Relativamente pocas instrucciones y modos de direccionamiento.
- Acceso a memoria limitado a instrucciones de carga y almacenamiento.
- Todas las operaciones son realizadas dentro de los registros de la CPU.
- Control por circuiteria en lugar de microprogramado.

Adicionalmente, la arquitectura ARM dispone de las siguientes características:

- Modos de direccionamiento de auto-incremento y auto-decremento para optimizar ciclos en programas.
- Carga y almacenamiento de múltiples instrucciones para maximizar el rendimiento de datos.
- Ejecución condicional de todas las instrucciones para maximizar el rendimiento de ejecución.

Las arquitecturas del conjunto de instrucciones(ISA) implementadas recientemente en los procesadores ARM son la T32(Thumb), A32(de 32 bits), A64(de 64 bits). Por otra parte, existen las extensiones de arquitecturas como la Jazzelle, para soportar aceleración JAVA, TrustZone, para cuestiones de seguridad, SIMD y Advanced SIMD(NEON) para multimedia y procesamiento de señales y decodificación respectivamente. Existen muchas variedades de procesadores ARM, con diferentes capacidades y características, cada uno de los cuales implementa su propia versión de arquitectura ISA soportada. El patrón de nomenclatura referente a la versión de la arquitectura se denota por ARMv{$\boldsymbol{n}$} donde $\boldsymbol{n}$ es un número sucesivo que hace referencia a la arquitectura. Por otro lado, la familia de procesadores fueron nombrados cronológicamente, empezando con la familia ARM1 diseñado en 1985 hasta el ARM11 diseñado en el 2002. A partir del año 2005, la nomenclatura de la familia se cambió al patrón Cortex-{*letra*}{*número*}.

En la Tabla 2.1 se listan las arquitecturas y sus respectivas familias de los procesadores ARM.

2.4.2 Movilidad

Gracias a la perteneciente simplicidad, diseño y bajo consumo eléctrico de los procesadores ARM, se pueden fabricar microprocesadores y microcontroladores de reducido tamaño y bajo costo, los cuales se han convertido en la tecnología dominante de la electrónica móvil integrada, logrando así, que actualmente se utilizan en tablets, smartphones, notebooks, reproductores de vídeo y música, calculadoras e incluso en los sistemas de frenos de los autos y periféricos de computadoras como los discos duros y routers. Actualmente la empresa Qualcomm fabrica la serie de procesadores de 64bits Snapdragon 800 de cuatro núcleos basados en la arquitectura ARMv8-A, los cuales alcanzan velocidades de hasta 2.7GHz manteniendo un bajo consumo energético. Este procesador lo integran en smartphones como el XPeria Z3, HTC One M9, LG Flex2, en tabletas como Amazon Fire HDX, Samsung Galaxy TabPro, e incluso en cámaras digitales como la LYTRO ILLUM y smartglasses como la ODG R-6S Glasses.

2.5 HPClab

HPCLab es un laboratorio de la Facultad de Ingeniería de la Universidad Tecnológica de Bolívar que tiene como objetivo ofrecer una alta capacidad de cómputo a la comunidad científica local y regional como herramienta fundamental para el

Arquitectura	**Familia**
ARMv1	ARM1
ARMv2	ARM2, ARM3
ARMv3	ARM6, ARM7
ARMv4	StrongARM, ARM7TDMI, ARM9TDMI
ARMv5	ARM7EJ, ARM9E, ARM10E, XScale
ARMv6	ARM11, ARM Cortex-M
ARMv7	ARM Cortex-A, ARM Cortex-M, ARM Cortex-R
ARMv8-A	Cortex-A72, Cortex-A57, Cortex-A53
ARMv8-R	No han sido lanzados a la fecha

Tabla 2.1: Arquitectura y familia de procesadores ARM

desarrollo de investigación básica y aplicada; generando, apropiando y difundiendo conocimiento en el área de computación de alto desempeño, aplicado al análisis y solución de problemas en ciencias e ingeniería en el ámbito local aportando así al desarrollo de la región [22]. El hardware disponible en el HPCLab es el siguiente:

- Un servidor HP ProLiant DL785G5 de 8 procesadores Quad Core AMD Opteron, 256 GB de memoria RAM.

Figura 2.1: Servidor SPIDER HP ProliantDL78 del HPCLab.

- Dos servidores DELL PowerEdge R710 de 2 procesadores Quad Core Intel Xeon 5500.

Figura 2.2: Servidor DELL PowerEdge R710 del HPCLab.

- Un servidor DELL PowerEdge R220 de Procesador Intel® Xeon E3-1220 de 3.1GHz, memoria RAM de 8GB y 500GB de almacenamiento.

Figura 2.3: Servidor DELL PowerEdge R220 del HPCLab.

Adicionalmente el laboratorio HPCLab cuenta con:

- Red FastEthernet
- Sistema eléctrico ininterrumpible por un periodo de 1 hora
- Sistema de refrigeración por aire acondicionado

2.6 Hardware y Software del Proyecto

En el siguiente apartado se describe el hardware y software utilizado para la implementación del clúster.

2.6.1 Ordenador de tarjeta Raspberry Pi

Raspberry Pi es una computadora ensamblada en una tarjeta de circuito impreso del tamaño de una tarjeta de crédito desarrollada en el Reino Unido por la Raspberry Pi Foundation. Al igual que una PC de escritorio, cuenta con puertos USB, HDMI, Audio, Video Compuesto(en algunos modelos), LAN; adicionalmente slots para conectar módulos como cámara y un conjunto de pines GPIO los cuales hacen que esta tarjeta sea de propósito general y utilizada en un sinfín de aplicaciones.

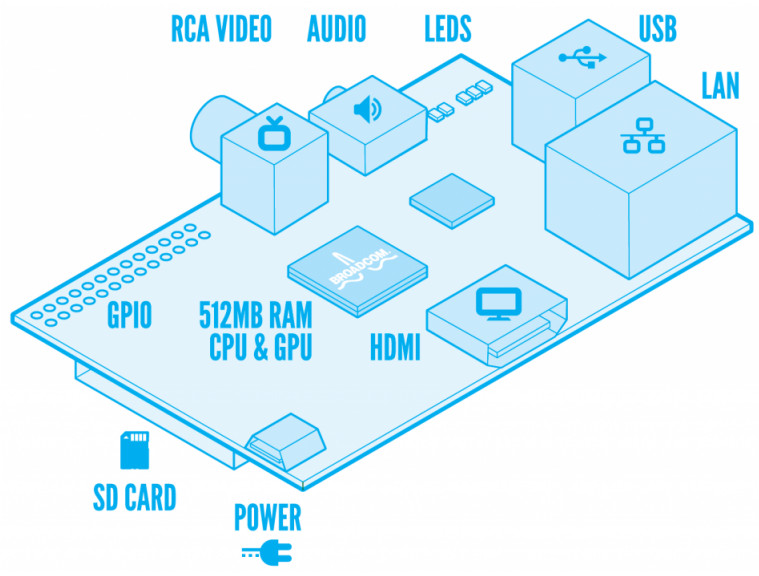

Figura 2.4: Componentes Básicos de una Raspberry Pi.

Actualmente existen varios productos de Raspberry Pi. En la Tabla 2.2 se muestran cada uno de los modelos y sus respectivas especificaciones:

		Raspberry Pi 1				Raspberry Pi 2
		Modelo A	Modelo A+	Modelo B	Modelo B+	Modelo B
Fecha lanzamiento		Feb 2012	Nov 2014	Feb 2012	Jul 2014	Feb 2015
CPU	Arquitectura	ARMv6	ARMv6	ARMv6	ARMv6	ARMv7
	Familia	ARM1176JZFS	ARM1176JZFS	ARM1176JZFS	ARM1176JZFS	ARM Cortex A7
	Frecuencia (MHz)	700	700	700	700	900 (4 núcleos)
Memoria RAM		256	256	512	512	1024
Puertos GPIO		26	40	26	40	40
Puerto HMDI		Si	Si	Si	Si	Si
Puertos USB		1	1	2	4	4
Video RCA		Si	No	Si	No	No
Puerto Ethernet		No	No	Si	Si	

Tabla 2.2: Especificaciones técnicas de los diferentes modelos de Raspberry Pi

2.6.2 Gestor de Colas de Trabajo: HTCondor

HTCondor anteriormente llamado Condor, es una herramienta de software especializada que permite la administración y gestión de tareas que requieran gran poder de cómputo. Al igual que las funciones realizadas por otros sistemas por lotes, HTCondor dispone de mecanismos de colas de trabajos, políticas de planificación, esquemas de prioridad, monitoreo y administración de recursos. El principio básico de funcionamiento de HTCondor es que el usuario les envía sus trabajos, ya sean paralelos o seriales, para que HTCondor los coloque en una cola, decida cuando y donde ejecutar los trabajos basándose en alguna política, monitoree minuciosamente el progreso y finalmente le informa al usuario cuando el o los trabajos hayan terminado de ejecutarse.

Una de las tantas característica de HTCondor es que puede ser usado para administrar clusters conformados por nodos de cómputo dedicado. Aún así, también tiene la capacidad de configurarse para utilizar solamente nodos de cómputo no dedicados, de tal forma que emplea mecanismos para aprovechar los tiempos ociosos de la CPU siempre y cuando estos no presenten eventos en el mouse o teclado, por ejemplo, presionar alguna tecla. En caso dado que un nodo no dedicado(previamente se encuentre disponible u ocioso) esté ejecutando alguna tarea enviada por HTCondor y algún usuario de dicho nodo realice algún evento en el teclado o mouse, HTCondor está en la capacidad de detectar este evento y parar la tarea en dicho nodo y reenviarla a otro nodo que sí se encuentre en estado ocioso o disponible(checkpoint). Debido al hecho anterior, HTCondor se implementa en

muchas universidades para aprovechar los tiempos ociosos de las computadoras que se encuentran a disposición en los laboratorios de los estudiantes como lo implementaron en [23]. Otra característica muy importante y que lo convierte en una herramienta con una gran ventaja en comparación con otros gestores de trabajos, es que tiene un diseño limpio que simplifica el envío de trabajo de los usuarios por medio del mecanismo **ClassAds** (siglas en inglés de *Classified Advertisements*), el cual permite a los usuarios pedir recursos necesarios y deseados para ejecutar los trabajos.

Para la implementación y envíos de trabajos a través de HTCondor, se debe contar con los nodos master(maestro), worker(esclavo o trabajador) y un universo. A continuación se hará una definición general de cada uno.

- **Nodo Master** (maestro): es el encargado de administrar y coordinar las tareas que van a ser ejecutadas en el clúster. Esto quiere decir, que es capaz de determinar cuáles nodos esclavos se encuentran disponibles o cuales son ideales para ejecutar una tarea. Por otra parte, este nodo también permite al usuario el acceso al cluster, por consiguiente, por cada grupo o entorno HTCondor(pool) debe existir únicamente un solo nodo master. Es de importancia resaltar, que este nodo también puede hacer el rol de worker y submit. A este nodo también se le conoce como Central Manager.

- **Nodo Worker** (esclavo o trabajador): este tipo de nodo es el que está en capacidad de recibir y ejecutar las tareas enviadas por el nodo submit.

- **Universo**: en HTCondor un universo es un entorno de ejecución el cual depende del tipo de aplicación que se desea ejecutar. Los universos son: *standard*, el cual provee confiabilidad y migración de las tareas por medio del mecanismo de checkpoint, para que un programa pueda ser enviado por medio de este universo, ha debido ser compilado con las librerías de HTCondor. *vanilla*, es el universo por defecto si no se especifica, aunque el administrador puede configurar HTCondor para que otro universo lo sea; es el menos restrictivo con los programas que se puedan enviar, ya que acepta cualquier programa escrito en cualquier lenguaje de programación; su desventaja es que no permite mecanismo de checkpoint. El universo *grid* permite a los usuarios enviar trabajos usando la interfaz de HTCondor, estos trabajos son enviados para ser ejecutados sobre recursos que se encuentren en una grid. El universo *java* permite enviar trabajos escritos en lenguaje

JAVA. El universo *parallel* es para enviar programas que requieran múltiples máquinas(nodos esclavos) para ejecutar un solo trabajo o aplicación paralela, para utilizarlo se debe hacer uso del estándar MPI (siglas en inglés de *Message Passing Interface*).

Capítulo 3

Metodología

Muchos autores hablan de varios tipos, enfoques o formas de investigación. Hernández et al(2010) clasifican la investigación en tres enfoques: cuantitativo, cualitativo y mixto. El enfoque cuantitativo usa la recolección de datos para probar hipótesis, con base en la medición numérica y el análisis estadístico, para establecer patrones de comportamiento y probar teorías. El enfoque cualitativo utiliza la recolección de datos sin medición numérica para descubrir o afinar preguntas de investigación en el proceso de interpretación. Y el enfoque mixto es la combinación de estos.

Bienmann(2001) y Tamayo(2004) definen dos tipos o formas de investigación: la investigación básica e investigación aplicada. La básica, que también suele llamarse como investigación pura o teórica se apoya dentro de un contexto teórico y su propósito fundamental es el de desarrollar teoría mediante el descubrimiento de amplias generalizaciones o principios. Por otra parte, la investigación aplicada, también conocida como práctica, empírica o dinámica, guarda íntima relación con la teoría, pues tiene su fundamento en ésta, pero su interés principal está en la aplicación práctica y en la utilización de los conocimientos alcanzados para aplicarlos en problemas, circunstancias y características concretas. Con esta clasificación y definición anterior, el proyecto que se presenta en este documento es una investigación aplicada con propósito de extender la capacidad de cómputo del HPCLab de la Universidad Tecnológica de Bolívar a través de la implementación de un clúster de nodos con procesadores de arquitectura ARM.

El presente estudio se realizó a través de las siguientes fases para su desarrollo y cumplimiento de los objetivos propuestos:

- Fase 1: Elaboración del estado del arte.
  Durante esta fase se realizó una recopilación y revisión documental en fuentes primarias como libros, artículos, revistas y publicaciones con conceptos y detalles técnicos concernientes al hardware, software y tecnologías existentes que permitan la implementación de entornos de ejecución paralela utilizando procesadores de arquitectura ARM ensamblados en ordenadores de tarjeta SoC.

- Fase 2: Diseño del sistema.
  En esta fase una vez que se realizó el análisis de los elementos de hardware y tecnologías consultadas en la fase anterior, se elaboró un diseño de red e infraestructura en donde se muestren los diferentes dispositivos y ordenadores (con sus respectivas direcciones IP) que conformaron el entorno, haciendo uso de las tecnologías de los clústeres, para luego realizar la implementación del mismo.

- Fase 3: Implementación del clúster
  Esta fase se dividió en varias etapas. La primera implicó instalar en una sola Raspberry Pi el gestor de colas HTCondor (middleware) sobre el sistema operativo Raspbian basado en Debian y agregarla al clúster del HPCLab por medio de la configuración del nodo maestro SPIDER. Luego que el cluster funcionó perfectamente con una sola Raspberry Pi, se continúo con la segunda etapa que consistió en realizar una imagen(clon) de la memoria SD de la Raspberry Pi de la primera etapa e instalar dicha imagen en las otras 7 Raspberry Pi, de tal forma, que solamente se tuvo que cambiar el hostname, archivo hosts y dirección ip, logrando así que se simplificará un poco más la tarea de configuración del S.O e instalación de HTCondor para ahorrar tiempo. La tercera y última etapa consistió en agregar las 8 Raspberry Pi al clúster del HPCLab.

- Fase 4: Prueba de Desempeño y Funcionamiento
  En esta etapa se realizarón dos pruebas. La primera prueba consistió en determinar la capacidad de cálculo del sistema por medio de la herramienta HPL (High Performance Linpack) y en la segunda se realizó el envío de un trabajo a través del gestor de colas HTCondor. Este trabajo contenía una aplicación programada en lenguaje C++ la cual calcula la cantidad de números primos que existen en un rango o intervalo. El propósito de esta

prueba es comparar los tiempos que toma en finalizar el trabajo en los nodos con procesador ARM y los nodos con procesador x86_x64.

- Fase 5: Elaborar Documentación
  Esta última fase es transversal a las 4 anteriores y se redacta la documentación que describen los procedimientos y recomendaciones para la implementación y uso del clúster, de tal forma que sirva como guía para el aprendizaje y uso óptimo del mismo.

Capítulo 4

Diseño e Implementación del Cluster

Contenido

En el siguiente capítulo se realiza el diseño de la infraestructura del clúster y su implementación. Se describen los pasos y procedimientos para la configuración de las Raspberry Pi y la instalación y configuración del gestor de colas HTCondor en los nodos esclavos y el nodo maestro.

4.1 Infraestructura implementada

Para el diseño de un clúster, lo primero que se debe definir es el propósito y tipo de cluster. De acuerdo a la definición y arquitectura, el cluster que se implementó en este proyecto es HTC y básicamente se encuentra formado por:

- Nodo Maestro SPIDER: encargado de enviar, supervisar y administrar los trabajos que se envíen a los nodos esclavos.
- Nodos esclavos: 8 Raspberry Pi capaces de ejecutar las tareas o trabajos que les sean asignadas.
- Red de comunicación: debido a la red física actual que se encuentra en el HPCLab y las interfaces de red del nodo maestro y los nodos esclavos, se va a utilizar una red con el estándar Ethernet en la cual, los nodos se conectarán a través de una topología en estrella utilizando un switch y cableado CAT5e, ambos suministrados por la Facultad de Ingeniería de la Universidad Tecnológica de Bolívar. Esta no es la mejor configuración y tipo de red para un cluster, debido a parámetros como la alta latencia, pero si uno de los más usuales por su bajo coste, y velocidad de transferencia adecuada.

En la Figura 4.1 se muestra el diseño de red y los equipos que conforman al cluster.

Respecto al direccionamiento IP, se solicitó al administrador de la red de comunicación del HPCLab 8 direcciones IP que se van a utilizar de forma estática en los nodos esclavos. Estas direcciones se encuentran especificadas en Tabla 4.1.

4.2 Instalación del sistema operativo Raspbian Wheezy

Al momento de realizar la instalación de HTcondor sobre el sistema operativo Raspbian Whezzy de la Raspberry Pi, se instaló satisfactoriamente por medio del supositorio la versión 7.2.3. Desafortunadamente, después de hacer varias configuraciones de este framework no se pudo unir la Raspberry Pi al clúster del HPCLab. Por lo anterior, se optó por actualizar el S.O Raspbian Whezzy a Jessie,

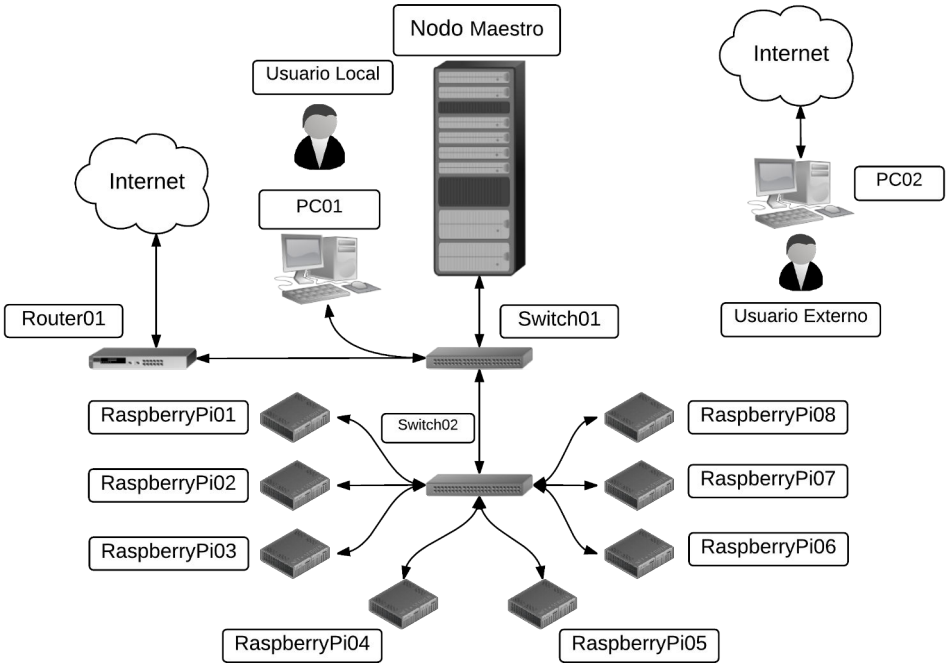

Figura 4.1: Infraestructura del cluster.

y volver a instalar HTCondor a través del supositorio dejando como resultado la versión 8.2.3 con la cual sí se pudo unir la Raspberry Pi al clúster HPCLab. Por tal motivo, en el siguiente apartado se describe el proceso de actualización del sistema operativo Raspbian Whazzy a Jessie.

4.3 Actualización del sistema operativo Raspbian Wheezy a Raspbian Jessie

La Raspberry Pi módelo B, tiene de manera predeterminada el sistema operativo(en lo adelante S.O) Raspbian basado en Debian Wheezy. Desafortunadamente, HTCondor en esta versión de sistema operativo no funcionó de manera correcta, por consiguiente, se optó por probar y realizar una actualización del mismo a la versión Debian Jessie, logrando así que HTCondor si funcionara de manera adecuada. A continuación, se muestra el proceso de actualización del sistema operativo Rapbian Wheezy(v7) a Raspbian Jessie(v8):

Equipo	Hostname	Dirección IP
Router01	"No aplica"	172.16.9.1
Nodo Maestro	spider	172.16.9.110
RaspberryPi01	rpi-worker01	172.16.9.115
RaspberryPi02	rpi-worker02	172.16.9.116
RaspberryPi03	rpi-worker03	172.16.9.117
RaspberryPi04	rpi-worker04	172.16.9.118
RaspberryPi05	rpi-worker05	172.16.9.119
RaspberryPi06	rpi-worker06	172.16.9.120
RaspberryPi07	rpi-worker07	172.16.9.121
RaspberryPi08	rpi-worker08	172.16.9.122
PC01	"No aplica"	172.16.9.x
PC02	"No aplica"	x.x.x.x

Tabla 4.1: Hosts, Hostname y direcciones IP del sistema

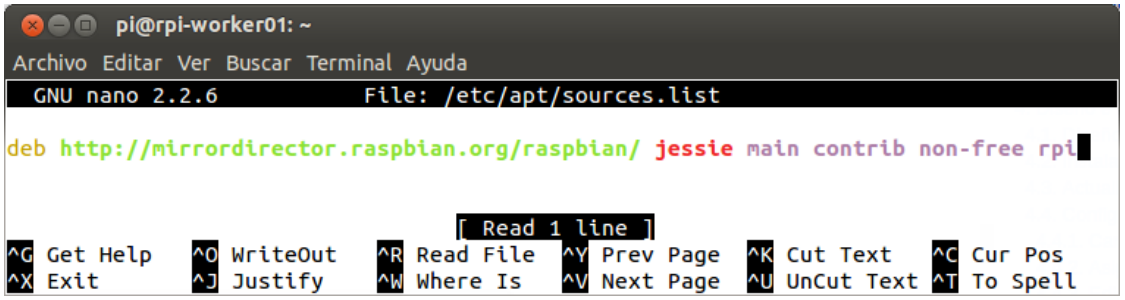

Figura 4.2: Edicion del archivo *sources.list* para la actualización del S.O a Debian Jeesie

Ejecutar la terminal y una vez autenticado como usuario root, se procede a editar el archivo **sources.list** con el comando **nano**:

```
# nano /etc/apt/sources.list
```

Dentro del archivo **sources.list** se cambia la palabra *wheezy* por *jessie* y se guardan los cambios de tal forma que el archivo quede como la Figura 4.2.

Posteriormente se procede a actualizar el S.O ejecutando los siguientes comandos:

```
# apt-get update
# apt-get --download-only dist-upgrade
# apt-get dist-upgrade
```

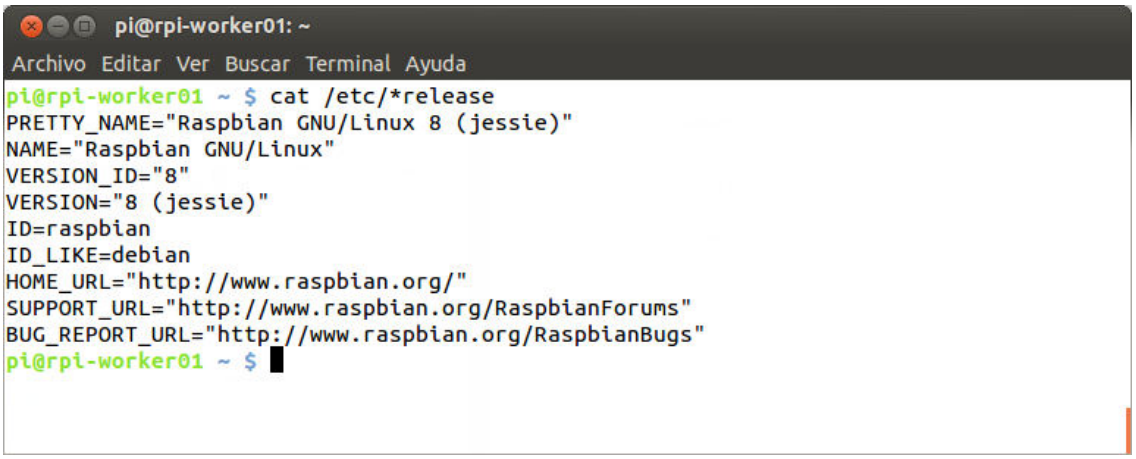

Figura 4.3: Salida generada por el comando `cat /etc/*release` para ver detalles del S.O

Cuando finalice la actualización, se reinicia la Raspberry Pi ejecutando el comando "***reboot***"

Una vez reiniciada la Raspberry Pi, se puede verificar la actualización del S.O ejecutando el comando:

```
$ cat /etc/*release
```

La Figura 4.3 muestra la salida generada por el comando "***cat /etc/*release***"

4.4 Configuración de la RaspberryPi 01

Las configuraciones pertinentes al S.O de las Raspberry's consisten en cambiar el hostname, asignar dirección ip estática y agregar todos los equipos del cluster en el archivo */etc/hosts*. A continuación se muestran dichas configuraciones:

4.4.1 Cambio de Hostname

Para realizar el cambio de hostname(nombre de equipo) en la Raspberry Pi se puede hacer directamente desde el menú de configuración de la Raspberry Pi "`raspi-config`" o por medio de la edición del archivo "`hostname`". A continuación se muestran las dos maneras de cambiar el nombre del equipo:

- **Cambio de nombre de equipo editando el archivo *hostname* :**
  Cómo usuario ***root*** se edita el archivo `/etc/hostname` y se reemplaza el nombre por defecto *raspberry* por el de interés, en este caso ***rpi-worker01***

  ```
  # nano /etc/hostname
  ```

  Una vez guardado los cambios Reiniciar la Raspberry con el comando "`reboot`" y una vez finalizado el reinicio se verifica el cambio de nombre con el comando "`hostname`".

- **Cambio de nombre de equipo desde el menú de configuración:**
  Para acceder al menú de configuración de la Raspberry Pi, desde la terminal se ejecuta como usuario ***root*** el comando "`raspi-config`" y se selecciona la opción **Advanced Options** como lo muestra la Figura 4.4

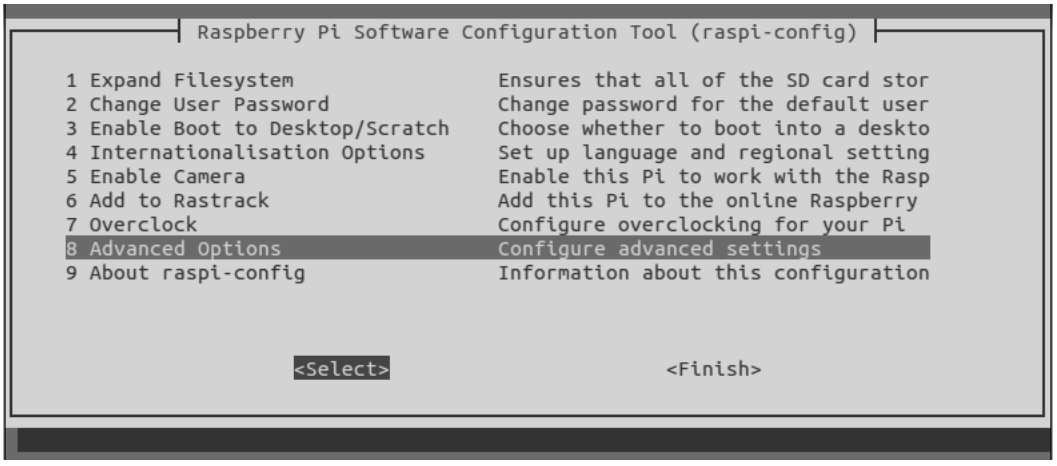

Figura 4.4: Menú de configuración `raspi-config`

Luego seleccionar la opción **Hostname**(ver Figura 4.5):

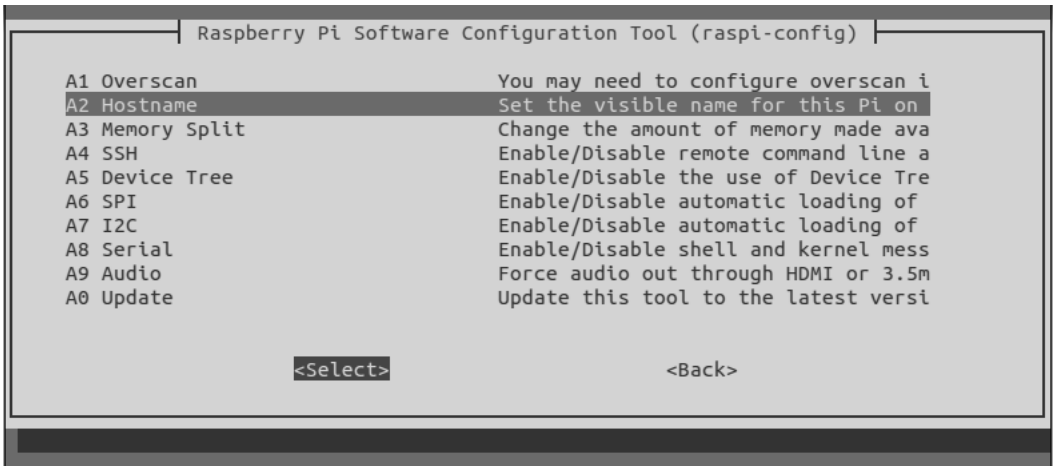

Figura 4.5: Menú de configuración, opción `Hostname`

El mensaje que se muestra a continuación(ver Figura 4.6) es simplemente para informar los caracteres permitidos para el nombre del equipo. Seleccionar **OK**.

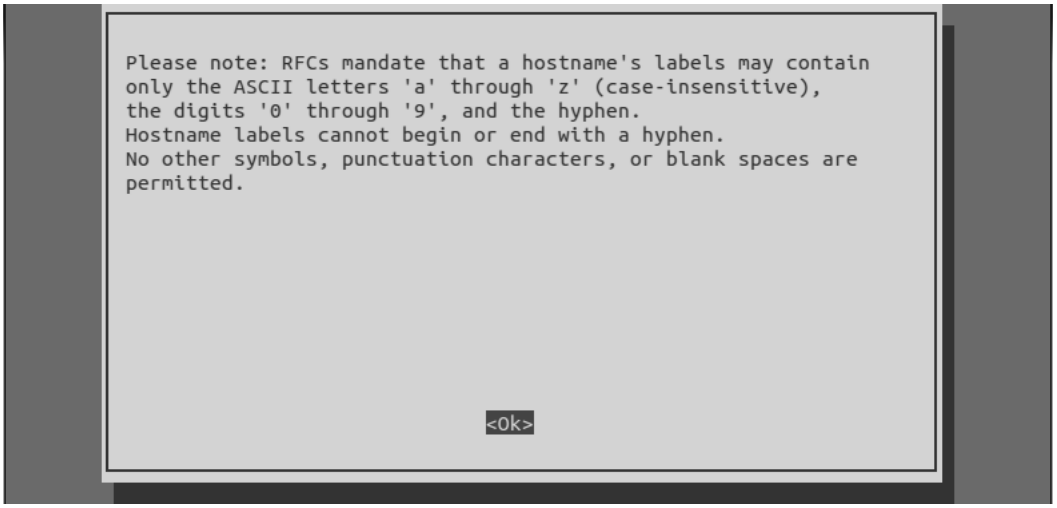

Figura 4.6: Caractéres permitidos para el `hostname`

En el siguiente formulario(ver Figura 4.7) el nombre de equipo por defecto es `raspberry`, para este caso se cambia por `rpi-worker01.unitecnologica.edu.co`

Figura 4.7: Formulario para cambiar el `hostname`

Una vez se cambia el nombre se reinicia la Raspberry Pi y se verifica el cambio de nombre del equipo con el comando "`hostname`"

4.4.2 Configuración Dirección IP Estática

Para configurar la dirección ip de forma estática en Raspbian, como usuario ***root*** se edita el archivo `interfaces` ubicado en el directorio `/etc/network/` y se elimian las configuraciones existentes (por recomendación, comentar con el caracter # al inicio de cada línea) las cuales establecen el direccionamiento dinámico, y reemplazarlas con los siguientes ajustes y valores que fueron establecidos en la etapa de diseño:

```
iface eth0 inet static
address 172.16.9.115
netmask 255.255.255.0
network 172.16.9.0
broadcast 172.16.9.255
gateway 172.16.9.1
```

Guardar los cambios con las teclas `Ctrl + X` y escribir `Yes`. Una vez guardado los cambios se reinicia las Raspberry Pi y se verifican los cambios con el comando `ifconfig`.

4.4.3 Edición del archivo `hosts`

Este archivo es utilizado por el S.O para guardar la correspondencia entre un nombre de dominio y un ordenador ya sea que pertenezca a la red local, la internet o el mismo ordenador, a través de su dirección IP. También puede usarse para bloquear el acceso a ciertos dominios o contenidos de internet.

El formato de las configuraciones de este archivo es el siguiente:

- Se debe introducir la dirección IP a la que resolverá, uno o más espacios o tabulaciones y el dominio a resolver.
- Se pueden introducir más de un dominio a resolver en la misma línea separados por uno o más espacios o tabulaciones.
- Cada correspondencia de dirección IP y dominio debe ir en una línea distinta.
- Las líneas que comienzan por uno más # equivalen a comentarios y no se computan.
- Las líneas en blanco tampoco se computan.

En el S.O Raspbian este archivo se encuentra en la ruta `/etc/hosts` y para editarlo, como usuario ***root*** se ejecuta el comando `# nano /etc/hosts`. Las configuraciones pertinentes para integrar un nodo al clúster del HPCLab son las siguientes:

```
# IP nodo maestro hostname nodo maestro nombre de dominio del nodo maestro
172.16.9.110    spider spider.unitecnologica.edu.co

# IP máquina local hostname máquina local nombre de dominio del nodo esclavo
172.16.9.115    rpi-worker01 rpi-worker01.unitecnologica.edu.co
```

4.5 Procedimiento de instalación del gestor de colas de trabajo HTCondor

Como se mencionó en el Capítulo 2, HTCondor es una herramienta de software especializada que permite la administración y gestión de tareas que requieran gran

poder de cómputo. En el siguiente acápite se describe el proceso de instalación del gestor de colas de trabajo HTCondor en el S.O Raspbian versión Jessie.

Desde la terminal ejecutar el comando:

```
$ sudo apt-get install condor
```

Luego, el instalador muestra el mensaje avisando que las configuraciones de HTCondor se pueden realizar durante el proceso de instalación o dejarlas por defecto(ver Figura 4.8). Debido a que no hay más opciones se presiona la tecla enter en la opción `OK` y cuando aparezca el segundo mensaje se selecciona `NO` (ver Figura 4.9) para realizar las configuraciones manualmente una vez instalado HTCondor.

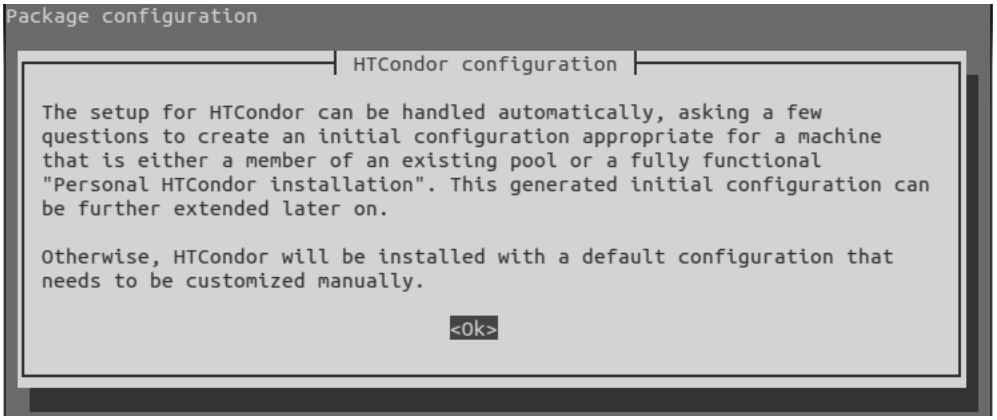

Figura 4.8: Mensaje de inicio de instalación de HTCondor

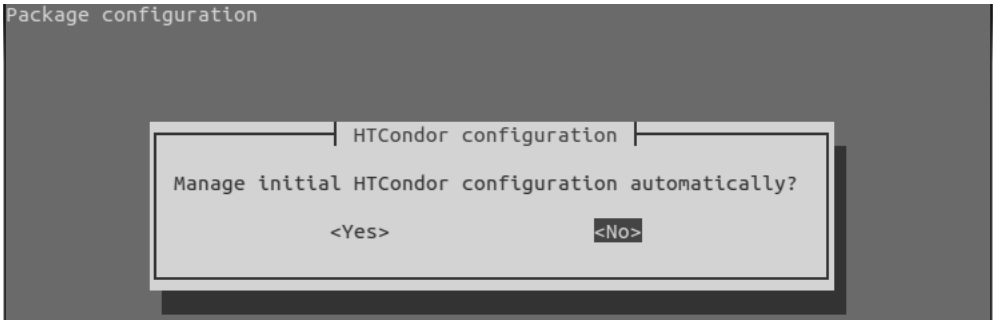

Figura 4.9: Opción de configuraciones automáticas de HTCondor

Una vez finalizado el proceso de instalación se verifica la versión de HTCondor instalada con el comando `condor_version` y este genera la salida que se muestra en la Figura 4.10.

```
pi@rpi-worker01 ~ $ condor_version
$CondorVersion: 8.2.3 Jan 07 2015 BuildID: Debian-8.2.3~dfsg.1-5+rpi1 Debian-8.2
.3~dfsg.1-5+rpi1 $
$CondorPlatform: ARMV7L-Raspbian_ $
pi@rpi-worker01 ~ $
```

Figura 4.10: Salida generada por el comando `condor_version`

4.6 Configuración de HTCondor

La configuración del gestor de colas HTCondor consiste en editar como usuario ***root*** el archivo `/etc/condor_config.local`. Los principales parametros o macros a configurar son los siguientes:

- `UID_DOMAIN`: esta macro se utiliza para indicar bajo que usuario es ejecutado el trabajo. Si esta macro es diferente entre el nodo que envia el trabajo y el nodo que lo ejecuta, HTCondor ejecuta el trabajo bajo el usuario *nobody*. En cambio, si son iguales, HTCondor ejecuta el trabajo bajo el usuario que envió el trabajo.

- `CONDOR_HOST`: esta macro se utiliza para definir la macro `$(COLLECTOR_HOST)`. Normalmente el `condor_collector` y `condor_negotiator` corren en la misma máquina. Si por alguna razón no se ejecutan en la misma máquina, no se necesitaría `$(CONDOR_HOST)`. Algunas de las macros de seguridad basadas en el host utilizan `$(CONDOR_HOST)` de forma predeterminada.

- `SEC_PASSWORD_FILE`: para las máquinas con sistema operativo basado en UNIX, corresponde a la ruta con el nombre del archivo que contiene las contraseñas de autenticación por medio de credenciales.

- `TRUST_UID_DOMAIN`: como medida de seguridad adicional, cuando HTCondor está a punto de generar un trabajo, se asegura que el `UID_DOMAIN` de la máquina a ejecutar el trabajo sea una subcadena del nombre de host de dicha máquina. Sin embargo, en algunos casos, puede haber múltiples `UID` que no se corresponden claramente a los nombres de dominio de Internet. En estos casos, los administradores pueden utilizar nombres para describir los dominios `UID` que no son subcadenas de los nombres de host de las máquinas. Para que esto funcione, HTCondor no debe hacer esta comprobación de

seguridad regular. Si el valor de `TRUST_UID_DOMAIN` se define como `True`, HTCondor no realizará esta prueba, y confiará en el `UID_DOMAIN` que es presentado por la máquina cuando se intenta generar un trabajo, en lugar de hacer que el nombre de host de la máquina coincide con el `UID_DOMAIN`. Cuando no está definida, el valor predeterminado es `False`, ya que es más seguro para realizar esta prueba.

- `SOFT_UID_DOMAIN`: es una variable booleana que por defecto es `False` en caso de no ser definida. Cuando HTCondor está a punto de ejecutar un trabajo como un usuario en particular (en vez de como usuario *nobody*), se verifica que el `UID` dado se encuentra en el archivo de contraseñas y coincide con el nombre de usuario. Sin embargo, en las instalaciones que no cuentan con todos los usuarios en el archivo de contraseñas de cada máquina, esta comprobación fallará y el intento de ejecución será abortado. Para evitar que HTCondor no realice esta comprobación, se ajusta esta variable de configuración en `True` causando que HTCondor ejecute el trabajo bajo el `UID` del usuario.

- `HAS_MY_SOFTWARE`: es una variable booleana que si se define con valor `True` le reporta a la máquina que se necesita de software especial instalado y se debe especificar en los requerimientos del trabajo para que solo sea ejecutado en los nodos que tengan dicho software especial.

- `DAEMON_LIST`: en esta macro se listan los demonios que debe iniciar HTCondor. Se separan por coma y puden ser: *`MASTER, COLLECTOR, NEGOTIATOR, STARTD, SCHEDD`*

A continuación se muestran las respectivas configuraciones que se deben realizar en este archivo tanto para el nodo maestro y los nodos esclavos, ya que en algunas macros se configuran de diferente forma:

4.6.1 Configuración del archivo `condor_config.local` en el nodo maestro

Las configuraciones de las macros en el archivo `condor_config.local` correspondientes al nodo maestro del HPCLab se pueden ver en la Figura 4.11

```
UID_DOMAIN = unitecnologica.edu.co
##########################################################################
# Human readable name for your Condor pool
COLLECTOR_NAME = "Mi cluster en $(UID_DOMAIN)"
# A shared file system (NFS), e.g. job dir, is assumed if the name is the same
#FILESYSTEM_DOMAIN = $(UID_DOMAIN)
ALLOW_WRITE = *.$(UID_DOMAIN)
CONDOR_ADMIN = root@$(FULL_HOSTNAME)
##########################################################################
# The following should be the full name of the head node
CONDOR_HOST = spider.unitecnologica.edu.co
##########################################################################
# Port range should be opened in the firewall (can be different on different
    machines)
# This 9000-9999 is coherent with the iptables configuration in the T3
    documentation
IN_HIGHPORT = 9999
IN_LOWPORT = 9000
# This is to enforce password authentication
SEC_DAEMON_AUTHENTICATION = required
SEC_DAEMON_AUTHENTICATION_METHODS = password
SEC_CLIENT_AUTHENTICATION_METHODS = password,fs,gsi,kerberos
SEC_PASSWORD_FILE = /etc/condor/password
ALLOW_DAEMON = condor_pool@*
##  Sets how often the condor_negotiator starts a negotiation cycle
##  for negotiator and schedd).
#  It is defined in seconds and defaults to 60 (1 minute), default is 300.
NEGOTIATOR_INTERVAL = 20
##  Scheduling parameters for the startd
TRUST_UID_DOMAIN = TRUE
# start as available and do not suspend, preempt or kill
START = TRUE
SUSPEND = FALSE
PREEMPT = FALSE
KILL = FALSE
SOFT_UID_DOMAIN = TRUE

HAS_MY_SOFTWARE = TRUE

DAEMON_LIST = MASTER, COLLECTOR, NEGOTIATOR, STARTD, SCHEDD
```

Figura 4.11: Configuraciones de las macros en el archivo condor_config.local del nodo maestro

4.6.2 Configuración del archivo `condor_config.local` en los nodos esclavos

Las configuraciones de las macros en el archivo `condor_config.local` correspondientes a los nodos esclavos del HPCLab se pueden ver en la Figura 4.12

```
UID_DOMAIN = unitecnologica.edu.co
##########################################################################
# Human readable name for your Condor pool
COLLECTOR_NAME = "Mi cluster en $(UID_DOMAIN)"
# A shared file system (NFS), e.g. job dir, is assumed if the name is the same
#FILESYSTEM_DOMAIN = $(UID_DOMAIN)
ALLOW_WRITE = *.$(UID_DOMAIN)
CONDOR_ADMIN = root@$(FULL_HOSTNAME)
##########################################################################
# The following should be the full name of the head node
CONDOR_HOST = spider.unitecnologica.edu.co
##########################################################################
# Port range should be opened in the firewall (can be different on different
    machines)
# This 9000-9999 is coherent with the iptables configuration in the T3
    documentation
IN_HIGHPORT = 9999
IN_LOWPORT = 9000
# This is to enforce password authentication
SEC_DAEMON_AUTHENTICATION = required
SEC_DAEMON_AUTHENTICATION_METHODS = password
SEC_CLIENT_AUTHENTICATION_METHODS = password,fs,gsi,kerberos
SEC_PASSWORD_FILE = /etc/condor/password
ALLOW_DAEMON = condor_pool@*
##  Sets how often the condor_negotiator starts a negotiation cycle
##  for negotiator and schedd).
#  It is defined in seconds and defaults to 60 (1 minute), default is 300.
NEGOTIATOR_INTERVAL = 20
##  Scheduling parameters for the startd
TRUST_UID_DOMAIN = TRUE
# start as available and do not suspend, preempt or kill

START = True
SUSPEND = False
CONTINUE = False
PREEMPT = False
KILL = False
SOFT_UID_DOMAIN = TRUE
HAS_MY_SOFTWARE = TRUE
WANT_SUSPEND = True
WANT_VACATE = True
DAEMON_LIST = MASTER, STARTD
```

Figura 4.12: Configuraciones de las macros en el archivo condor_config.local de los nodos esclavos

Capítulo 5

Pruebas

Contenido

En el presente capítulo se describen las pruebas que se le realizaron al sistema. En el subcapítulo 5.1 se describe la prueba para verificar la operación del cluster como un único recurso enviando trabajos a través del gestor de colas de trabajo HTCondor para que sean ejecutados en los nodos esclavos con procesadores de arquitectura ARM y comparar los tiempos de ejecución cuando estos mismos trabajos sean ejecutados sobre nodos esclavos con procesadores de arquitectura x86_x64.

Por otra parte, en esta investigación se consideró el interés en realizar una medición de desempeño, básicamente para observar el desempeño del cluster con las Raspberry Pi, claro está, no se espera un rendimiento aproximado o igual a los alcanzados por clusters de la actualidad, por obvias razones de especificaciones técnicas de hardware que poseen las Raspberry Pi. Para esta prueba de desempeño (benchmark) del cluster, la cual se describe en el subcapítulo 5.2, se utilizará la librería HPL (High Performance Linkpack) debido que es la más utilizada para medir el desempeño de clusteres de la industria y universidades, tanto así, que se utiliza en el TOP500 para calificar sistemas HPC alrededor del mundo.

5.1 Cálculo de Números Primos en un rango con HTCondor

La siguiente prueba consiste en verificar la operación del clúster por medio del envío de trabajos con el middleware HTCondor en los nodos esclavos con procesadores de arquitectura ARM y posteriormente estos mismos trabajos enviarlos a nodos esclavos con procesadores de arquitectura x86_x64 para comparar los tiempos en que fueron realizados. Los trabajos consisten en una aplicación escrita en lenguaje C++ la cual determina la cantidad de números primos que existen en un rango(ver Anexo).

HTCondor necesita de un archivo con extensión `.submit` en el cual se configuran las propiedades, requerimientos, la cola de trabajos y las salidas. En la Figura 5.1 se muestra el archivo `.submit` para enviar el trabajo a los nodos esclavos con procesadores ARM:

```
Universe   = vanilla
Executable = primos
Requirements = Arch == "armv6l"
should_transfer_files = YES
when_to_transfer_output = ON_EXIT

Log        = resultados/primos.log
Output     = resultados/primos_$(PROCESS).out
Error      = resultados/primos_$(PROCESS).error

Arguments = 900001 1000000
Queue

Arguments = 800001 900000
Queue

Arguments = 700001 800000
Queue

Arguments = 600001 700000
Queue

Arguments = 500001 600000
Queue

Arguments = 400001 500000
Queue

Arguments = 300001 400000
Queue

Arguments = 200001 300000
Queue
```

Figura 5.1: Archivo .submit para enviar trabajos en nodos con CPU ARM

5.2 HPL (High Performance Linpack)

HPL es una herramienta de software libre y código abierto que implementa el paquete Linpack(conjunto de subrutinas FORTRAN que resuelven problemas de álgebra lineal como ecuaciones lineales y multiplicación de matrices) pero modificado para sistemas de cómputo de memoria distribuida. El objetivo de la herramienta HPL no es medir el desempeño general del sistema, por el contrario, evalúa el desempeño en un área muy específica; básicamente lo que realiza es calcular la solución de ecuaciones lineales de alta densidad, permitiendo así, la medición de operaciones con números de punto flotante(de 64 bits de precisión) por segundo y el tiempo que le tomó en calcular dicha solución.

Para su ejecución es necesario tener en el sistema una implementación MPI (Message Passing Interface) para realizar la comunicación y transferencia de datos entre los nodos, y una implementación de los subprogramas BLAS (Basic Linear Algebra Subprograms) o VSIPL (Vector Signal Image Processing Library).

Para obtener el máximo rendimiento durante la prueba, se deben tener en cuenta varios factores que están íntimamente ligados al sistema, como lo son, la red de comunicación de los nodos, la cantidad de nodos y la memoria total del sistema. Por otra parte, se debe tener una correcta configuración de parámetros (tuning) fundamentales al momento de realizar la prueba con HPL, como lo son:

- ***NB***, tamaño del bloque que se va a utilizar para la distribución de trabajo. Su valor es empírico y se recomienda que se encuentre en el intervalo de 32 a 256.

- ***N***, tamaño del problema: hay que darle un valor de acuerdo al número de nodos y la cantidad de memoria disponible. Una forma sencilla y aproximada de calcular este valor es mediante la fórmula:

$$N = \sqrt{\frac{(M * 0.8 * W) * 1024^2}{8}} \tag{5.1}$$

  donde:
  M es la memoria disponible por nodo
  W es la cantidad de nodos del cluster

- *PxQ*(malla o cuadrícula de procesos): la sumatoria de ambos corresponde a la cantidad de nodos del cluster. Deben de ser lo más cercano posible y en caso de no ser iguales P debe ser menor que Q.

Las configuraciones anteriores se realizan en el archivo `HPL.dat` que se genera en el directorio `bin` donde fue compilado HPL, el cual es llamado por el ejecutable `xhpl` al momento de su ejecución en la prueba [1].

El primer benchmark que se le realizó al sistema consistió en variar *NB* con los valores 32,64,98,128,160,192,224 y 256; y valor fijo de *N=19200* en una malla de *PxQ=1x8*. Para el segundo benchmark, se tomaron los mismos valores de *NB* y *N* pero en una malla de *PxQ=2x4*.

Por cuestiones prácticas, las configuraciones del archivo `HPL.dat` se realizaron con la herramienta online de la empresa Advanced Clustering [2] en la que solamente se debe ingresar la cantidad de nodos, núcleos por nodo, cantidad de memoria disponible por nodo y el tamaño del bloque *NB* para que esta genere el archivo `HPL.dat`. En la Figura 5.2 se muestran las configuraciones generadas por la herramienta y la modificación de la cantidad de *NB* (# of NBs) y *NBs* para que el benchmark se realice en una sola ejecución.

Para la ejecución del benchmark, se implementó la librería OpenMPI en su versión 1.6.5, la cual permite realizar la comunicación y transferencia de datos entre los nodos, y la herramienta ATLAS 3.10.2(siglas en inglés de Automatically Tuned Linear Algebra Software) que implementa BLAS para las operaciones de matrices y vectores. El comando para ejecutar la prueba de rendimiento fue:

```
$ mpirun -np 8 -hostfile machines xhpl
```

donde `np` es el número de procesos a ejecutar, y `machines` es un archivo en texto plano donde se especifican los nodos y núcleos de CPU que van a ejecutar los procesos(ver anexo).

A manera de ejemplo, en la Figura 5.3 se muestra la salida que se genera una vez finalizada la prueba de rendimiento con las configuraciones del archivo HPL.dat anterior. Allí se muestra un resumen de las configuraciones básicas y se aprecia el tiempo que duró la prueba, 9469 segundos, alcanzando un rendimiento de 498MFlops.

[1] En el idioma inglés esta configuración es conocida como *tuning*
[2] http://www.advancedclustering.com/act-kb/tune-hpl-dat-file/

```
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
6            device out (6=stdout,7=stderr,file)
1            # of problems sizes (N)
19200        Ns
8            # of NBs
32 64 96 128 160 192 224 256          NBs
0            PMAP process mapping (0=Row-,1=Column-major)
1            # of process grids (P x Q)
2            Ps
4            Qs
16.0         threshold
1            # of panel fact
2            PFACTs (0=left, 1=Crout, 2=Right)
1            # of recursive stopping criterium
4            NBMINs (>= 1)
1            # of panels in recursion
2            NDIVs
1            # of recursive panel fact.
1            RFACTs (0=left, 1=Crout, 2=Right)
1            # of broadcast
1            BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1            # of lookahead depth
1            DEPTHs (>=0)
2            SWAP (0=bin-exch,1=long,2=mix)
64           swapping threshold
0            L1 in (0=transposed,1=no-transposed) form
0            U  in (0=transposed,1=no-transposed) form
1            Equilibration (0=no,1=yes)
8            memory alignment in double (> 0)
##### This line (no. 32) is ignored (it serves as a separator). ######
0                               Number of additional problem sizes for PTRANS
1200 10000 30000                values of N
0                               number of additional blocking sizes for PTRANS
40 9 8 13 13 20 16 32 64        values of NB
```

Figura 5.2: Configuraciones archivo HPL.dat para el benchmark con HPL

```
================================================================================
HPLinpack 2.1  --  High-Performance Linpack benchmark  --   October 26, 2012
Written by A. Petitet and R. Clint Whaley,  Innovative Computing Laboratory, UTK
Modified by Piotr Luszczek, Innovative Computing Laboratory, UTK
Modified by Julien Langou, University of Colorado Denver
================================================================================

An explanation of the input/output parameters follows:
T/V    : Wall time / encoded variant.
N      : The order of the coefficient matrix A.
NB     : The partitioning blocking factor.
P      : The number of process rows.
Q      : The number of process columns.
Time   : Time in seconds to solve the linear system.
Gflops : Rate of execution for solving the linear system.

The following parameter values will be used:

N      :   19200
NB     :      32
PMAP   : Row-major process mapping
P      :       2
Q      :       4
PFACT  :   Right
NBMIN  :       4
NDIV   :       2
RFACT  :   Crout
BCAST  :  1ringM
DEPTH  :       1
SWAP   : Mix (threshold = 64)
L1     : transposed form
U      : transposed form
EQUIL  : yes
ALIGN  : 8 double precision words

--------------------------------------------------------------------------------

- The matrix A is randomly generated for each test.
- The following scaled residual check will be computed:
      ||Ax-b||_oo / ( eps * ( || x ||_oo * || A ||_oo + || b ||_oo ) * N )
- The relative machine precision (eps) is taken to be               1.110223e-16
- Computational tests pass if scaled residuals are less than                16.0

================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR11C2R4       19200    32     2     4            9469.97              4.983e-01
HPL_pdgesv() start time Sat Jun 13 12:44:22 2015

HPL_pdgesv() end time   Sat Jun 13 15:22:12 2015

--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=        0.0010690 ...... PASSED
================================================================================
```

Figura 5.3: Salida por pantalla cuando se realiza el benchmark con HPL

Capítulo 6

Resultados y Análisis

Contenido

6.1 Resultados del cluster con el middleware HTCondor

Los ordenadores de tarjeta Raspberry Pi, los cuales poseen procesadores de arquitectura ARM, se pudieron agregar con éxito al cluster del HPCLab gracias a este gestor de colas multiplataforma, dejando como resultado un cluster heterogéneo no solo a nivel de sistema operativo y memoria, sino también a nivel de arquitectura de la CPU. La Figura 6.1 muestra el estado del cluster al ejecutar en la terminal el comando `condor_status`. Allí se puede apreciar que el cluster está conformado por 40 nodos esclavos con CPU de arquitectura *x86_64* y 8 nodos esclavos con CPU de arquitectura *armv6l* los cuales representan a las Raspberry Pi.

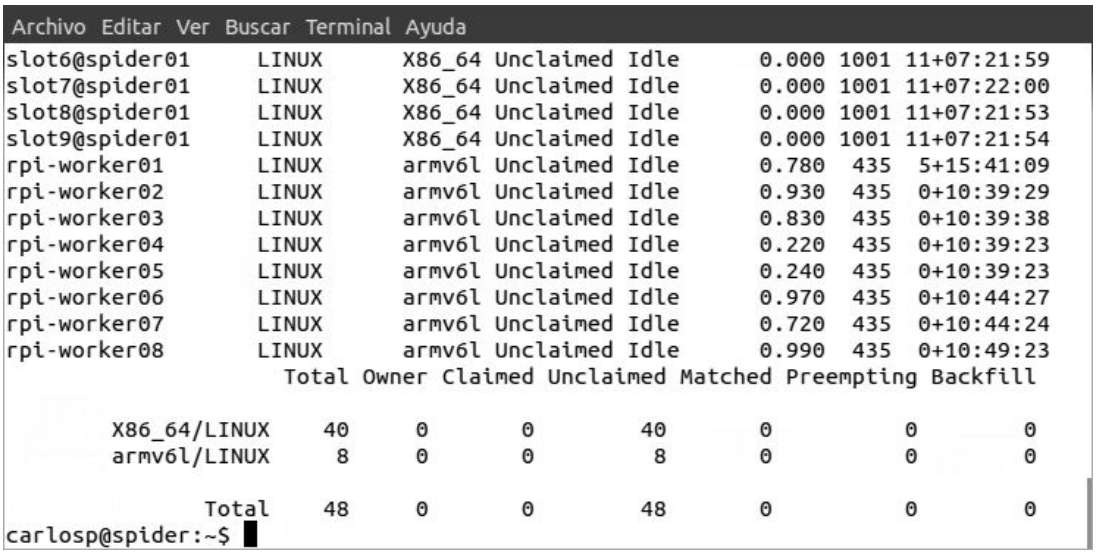

Figura 6.1: Salida generada por el comando `condor_status` para ver el estado del clúster con HTCondor

6.2 Resultados de la prueba del Cálculo de Números Primos en un rango

En la Tabla 6.1 se muestran los tiempos de ejecución de cada trabajo enviado con el gestor de colas HTCondor en los nodos con CPU de arquitectura ARM y x8_x64. A simple vista, se puede notar una gran diferencia en los tiempos, pero es debido a la superioridad de las especificaciones técnicas de los nodos con CPU de arquitectura x86_64, se tratan de procesadores que alcanzan frecuencias de hasta 2.7 GHz acompañados con capacidad de memoria RAM disponible de 996 MB; todo lo anterior frente a los 0.8 GHz y 435 MB de memoria RAM disponible de las Raspberry Pi.

6.3 Resultados de la prueba de rendimiento con HPL

Las dos ejecuciones de la prueba de rendimiento con HPL, se realizaron con los valores de N=19200 y NB variable con con valores desde 32 hasta 256 para las cuadrículas PxQ=1x8 y PxQ=2x4. Cada prueba se ejecutó 5 veces para evitar que

Intervalo	Tiempo(mins) Nodos ARM	Tiempo(mins) Nodos x86_x64	Cantidad No's Primos
900001 - 1000000	12.81	0.52	7,224
800001 - 900000	11.67	0.47	7,323
700001 - 800000	10.47	0.42	7,408
600001 - 700000	9.19	0.37	7,445
500001 - 600000	7.91	0.32	7,560
400001 - 500000	6.46	0.27	7,678
300001 - 400000	5.19	0.21	7,863
200001 - 300000	3.77	0.15	8,013
TOTALES	**67.46**	**2.73**	**60,514**

Tabla 6.1: Tiempos de ejecución de los trabajos con HTCondor para el cálculo de números primos en un rango.

queden almacenado datos en la memoria principal o en la caché. Los resultados obtenidos se promediaron y se resumen en la Tabla 6.2.

NB	*N = 19200; PxQ=1x8*		*N = 19200; PxQ=2x4*	
	MFLOPS	Tiempo(horas)	MFLOPS	Tiempo(horas)
32	493,8	2,655	498,3	2,631
64	525,6	2,494	547,5	2,394
96	529,0	2,478	554,6	2,364
128	539,8	2,429	559,1	2,345
160	509,5	2,573	554,8	2,363
192	502,5	2,609	546,0	2,401
224	510,6	2,567	518,7	2,527
256	474,6	2,762	358,3	3,658

Tabla 6.2: Resultados del rendimiento con HPL para valores de *N=19200* y *NB* variable

En la Figura 6.2 se puede determinar el mejor valor de NB y la mejor cuadrícula de PxQ que permite alcanzar el mayor rendimiento del cluster. El mayor rendimiento alcanzado (***Rmax***) fue de 559,1 MFLOPS con un valor de NB=128 y cuadrícula de PxQ=2x4, ejecutado en 2.34 horas, alcanzando una diferencia de 19,3MFLOPS frente al mejor rendimiento de la cuadrícula PxQ=1x8. Cuando el valor de NB es igual o superior a 224 se presenta una pérdida de rendimiento, lo cual quiere decir que a valores de bloque NB muy grandes el sistema sufre desbordamiento y a valores muy pequeños el sistema no alcanza el mayor rendimiento pero si lo aproxima.

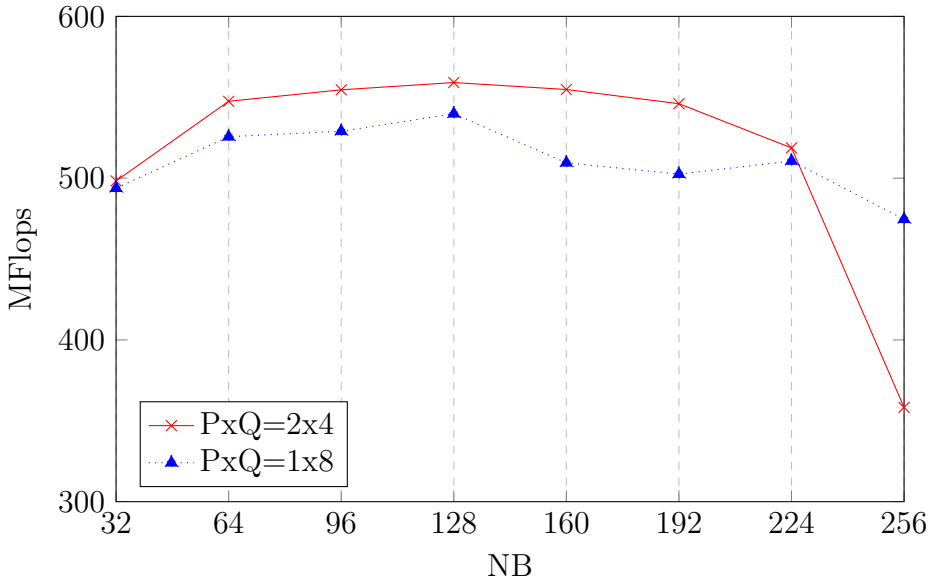

Figura 6.2: Rendimiento del benchmark con HPL.

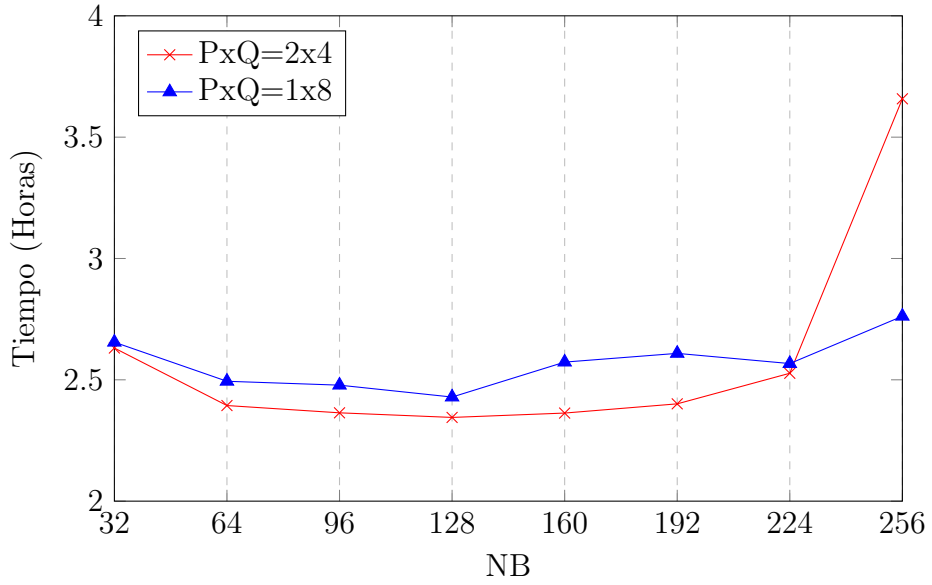

Figura 6.3: Tiempos de ejecución del benchmark con HPL.

De acuerdo a [24], quien contribuyó en la implementación de HPL, el pico de rendimiento teórico (***Rpeak***) se puede determinar contando el número de operaciones de suma y multiplicación con números de punto flotante (de doble precisión) que se pueden realizar durante un periodo de tiempo, usualmente el ciclo de reloj de la CPU. Partiendo de lo anterior, para determinar el ***Rpeak*** del sistema, se utiliza la Ecuación (6.1).

$$\textbf{\textit{Rpeak}} = (\#\text{ total de núcleos}) * (\text{Frecuencia por núcleo}) * (\text{Operaciones por ciclo}) \tag{6.1}$$

Para este caso del cluster formado por las 8 Raspberry Pi, los cuales tienen una CPU capaz de realizar 0.5 operaciones de suma y multiplicación de punto flotante por ciclo [1], y una frecuencia de 800MHz, el valor ***Rpeak*** según la Ecuación (6.1) es de 3.2 GFlops.

Con el valor de ***Rpeak*** y ***Rmax***, aplicando la ecuación (6.2), el cluster implementado logra una eficiencia aproximada del 17%. No es de sorprender este valor de eficiencia obtenido, ya que el sistema se ve muy afectado por otros factores, principalmente la tecnología de red utilizada, la cual es muy propensa a sufrir latencia y cuellos de botella, y los relacionados con la configuración de HPL y las librerías que esta implementa.

$$\textbf{\textit{Eficiencia}} = \frac{Rmax}{Rpeak} \tag{6.2}$$

[1] http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0301h/Cegdejjh.html

Capítulo 7

Conclusiones y Recomendaciones

Contenido

7.1 Conclusiones

Con esta investigación se pudo demostrar que con el gestor de colas HTCondor, se pudieron agregar al clúster del HPCLab de la UTB nodos esclavos con procesadores de arquitectura ARM ensamblados en ordenadores tipo SoC Raspberry Pi, dejando como resultado un cluster heterogéneo no solo a nivel de sistema operativo y memoria, sino también a nivel de arquitectura de la CPU.

Durante la prueba de funcionamiento del cluster con las Raspberry Pi y el gestor de colas HTCondor, en la cual se enviaron trabajos con una aplicación programada en lenguaje C++ que calculaba la cantidad de números primos en un rango, no se obtuvieron resultados muy buenos en cuestión de tiempos de ejecución, ya que a estos nodos les tomó más de una hora culminar la ejecución, frente a los 3 minutos que se tomó en ejecutarse en los nodos con procesadores de arquitectura x86_x64. Lo anterior era de esperarse considerando la gran ventaja de hardware que tienen los nodos con CPU x86_x86 del HPCLab frente a los nodos con CPU ARM de las Raspberry Pi.

Por otra parte, con el objetivo de obtener métricas de rendimiento, se realizaron benchmarks con la herramienta HPL y las 8 Raspberry Pi, en la cual, en el mejor de los casos con valores de N=19000, NB=128 y PxQ=2x4, el clúster alcanzó el mayor rendimiento de 559 MFLOPS logrando una eficiencia de 17%. Esta baja eficiencia se debe principalmente a las especificaciones técnicas de la Raspberry las cuales son muy básicas en cuestión de frecuencia de la CPU y cantidad de memoria aleatoria. De igual forma, como en esta prueba los nodos se comunican entre sí (a través del pase de mensajes con OpenMPI) con un alto flujo de datos y la red utilizada es de tecnología FastEthernet de 100 Mbits, resultó ser muy baja ya que esta presenta alta latencia causando que se produjeran cuellos de botella, los cuales son la causante del bajo rendimiento. Cabe resaltar que la Raspberry Pi carece de interfaces y módulos que permitan utilizar otra tecnología de red de mayor velocidad de transferencia o ancho de banda como la, GigabitEthernet, InfiniBand o Myrinet.

Con las pruebas que se realizaron al sistema, queda demostrado que los 8 nodos esclavos con procesadores de arquitectura ARM ensamblados en las Raspberry Pi modelo B que se agregaron al HPCLab de la UTB, tienen un bajo rendimiento en la ejecución de trabajos en paralelo, sin embargo, hay que tener en cuenta que estas tarjetas son muy económicas en comparación con los nodos esclavos con CPU de arquitectura x86_x64 lo cual justifica su hardware básico y rendimiento.

Debido a la innovación, aumento de rendimiento, tendencia y bajo costo de los procesadores de arquitectura ARM, muchas compañías enfocadas al mercado de los sistemas clúster, están despertando el interés en implantar estos procesadores en dichos sistemas como una opción económica y de ahorro energético para el cliente final.

Por último, en esta investigación se dejó implementado un clúster -*heterogéneo y con nodos dedicados*- funcional en el que se pueden ejecutar aplicaciones paralelas, y se produjo su respectiva documentación para la implementación y uso, de tal forma que le permita a los estudiantes de la UTB o cualquier otro usuario, explorar el mundo de la computación de alto desempeño sin exponer la seguridad ni la infraestructura física del HPCLab.

7.2 Recomendaciones

Debido a las experiencias obtenidas durante la ejecución de este proyecto de investigación, se exponen las siguientes recomendaciones en cuanto a la implementación, mantenimiento y uso del cluster del HPCLab en todo su conjunto con los nodos esclavos de procesadores de arquitectura X86_x64 y ARM y software utilizado:

- Para el uso óptimo y mantenimiento del cluster del HPCLab de la UTB, es necesario estar familiarizado con el sistema operativo LINUX y cualquiera de sus distribuciones, de igual forma con el gestor de colas HTCondor.
- Si el tiempo de ejecución de los trabajos no es una prioridad, o dicho de otra forma, ejecutar trabajos que no requieran alto procesamiento, o simplemente para realizar pruebas de aplicaciones en paralelo, utilizar los nodos esclavos con procesadores de arquitectura ARM ensamblados en las Raspberry Pi, ya que estos resultan ser ideales para estos propósitos.

Capítulo 8

Trabajo Futuro

Teniendo en cuenta que en este trabajo se demostró que el cluster implementado con las 8 Raspberry Pi modélo B no logró un buen rendimiento, o no esta listo para ejecutar trabajos que requieran alta capacidad de procesamiento, aún se puede seguir intentando en sacar el máximo provecho del hardware de las Raspberry Pi aumentando la frecuencia de la CPU a través del overclocking, y reducir la memoria para gráficos de tal forma que así se aumentará la memoria RAM la cual es compartida con la GPU. Aún así, es interesante realizar pruebas de ejecución de aplicaciones paralelas que brinden solución a problemas reales como la simulación de minimización de proteinas, modelado numérico hidrodinámico 2D y 3D o análisis de variables bioclimáticas e índices de sequía continua de superficie.

Por otra parte, hay que seguir explorando otras opciones de procesadores ARM disponibles en el mercado, ya que estos estan siendo desarrollados con muy buenas carateristicas que los hacen ideales para implementarlos en clusteres, tal es el caso de los procesadores Opteron A1100 series de la empresa AMD los cuales se caracterizan por tener hasta 8 núcleos, memoria caché de nivel 3 y alcanzar frecuencias superiores a los 2GHz conservando su economía y bajo consumo energético.

Appendix A

Código en lenguaje C++ para Cálculo de números primos en un rango

```
1  #include <iostream>
2  #include <cstdlib>
3  #include <ctime>
4  using namespace std;
5  bool es_primo(int num){
6     if(num == 1 || num == 2){
7        return true;
8     }else{
9        for(int i = 2; i < num; i++){
10          if(num % i == 0){
11             return false;
12          }
13       }
14       return true;
15    }
16 }
17 int main(int argc, char *argv[]){
18    unsigned int x = atoi(argv[1]);
19    unsigned int y = atoi(argv[2]);
20    int cont_div = 0;
21    int cont_primos = 0;
22    clock_t tiempo_inicial = clock();
23    for(int i = x; i <= y; i++){
24       if(es_primo(i) == true){
25          cont_primos++;
26       }
27    }
28
29    clock_t tiempo_final = clock();
30    double tiempo_ejecucion = double(tiempo_final-tiempo_inicial)/CLOCKS_PER_SEC;
```

```
31      cout<<"En el intervalo ["<<x<<","<<y<<"] hay "<<cont_primos<<" numeros primos"
         <<endl;
32      cout<<"Tiempo de ejecucion: "<<tiempo_ejecucion/60<<" minutos"<<endl;
33      cout<<"Tiempo de ejecucion: "<<tiempo_ejecucion<<" segundos"<<endl;
34      cout<<"======================================================"<<endl;
35      return 0;
36  }
```

Referencias

[1] Kritikakos P. *Low-Power High Performance Computing.* pages 3–5, 2011.

[2] Buyya R. *High Performance Cluster Computing, Vol 1: Architectures and Systems. Prentice Hall*, pages 9–13, 1999.

[3] Balakrishnan N. *Building and benchmarking a low power ARM cluster.* page 9, 2012.

[4] Mont-blanc. URL `http://www.montblanc-project.eu/`.

[5] Rajovic N., Rico A., Puzovic N., et al. *Tibidabo: Making the Case for an ARM-Based HPC System.* pages 6–10, 2013.

[6] Sukaridhoto S., KHalilullah A., and Pramadihanto D. *Further Investigation on Building and benchmarking a low power embedded Cluster for Education. International Seminar on Applied Technology, Science, and Art (4 th APTECS 2013)*, pages 2–7, 2013.

[7] Integrated Device Technology Inc. *IDT, Orange Silicon Valley, NVIDIA Accelerate Computing Breakthrough with RapidIO-based Clusters Ideal for Gaming, Analytics.* URL `http://www.idt.com/about/press-room/idt-orange-silicon-valley-nvidia-accelerate-computing-\breakthrough-rapidio-based-clusters-ideal-gami`. Accessed: 2015-02-23.

[8] Dennis A. *Raspberry Pi Super Cluster. Packt Publishing*, pages 7–8, 41–48, 2013.

[9] Cloutier M., Paradis C., and Weaver M. *Design and Analysis of a 32-bit Embedded High-Performance Cluster Optimized for Energy and Performance*, volume Extended Edition. 2014.

[10] Prabhu C. Grid and cluster computing. *PHI Learning Private Limited*, pages 99–105, 2008.

[11] Flynn M. Some computer organizations and their effectiveness. *IEEE Trans. Computers, 21*, page 948–960, 1972.

[12] Aguilar J and Leiss E. *Introducción a la computación paralela*. Editorial: Universidad de los Andes. Mérida – Venezuela, 2004.

[13] Hager G. and Wellein G. Concepts of high performance computing. Regionales Rechenzentrum Erlangen (RRZE) Friedrich-Alexander-Universitat Erlangen-Nurnberg:43–47, 2008.

[14] Eijkhout V., Chow E., and Van de Geijn R. Introduction to high performance scientific computing. Evolving Copy:76–83, 2014.

[15] Blelloch G. Programming parallel algorithms. in communications of acm. page 39, 1996.

[16] Leopold C. *Parallel and Distributed Computing: A survey of models, paradigms, and approaches*. John Wiley & Sons inc., 2001.

[17] Isaza G. and Duque N. Arquitecturas y modelos de programación en computación grid. *Scientia et Technica Año XIII, No 37*, 2007.

[18] Sunderam V. PVM: A framework for parallel distributed computing, Concurrency: Practice and Experience. 2:315–339, 1990.

[19] Jacobsen D., Thibault J, and Senocak I. An mpi-cuda implementation for massively parallel incompressible flow computations on multi-gpu clusters. *48th AIAA Aerospace Sciences Meeting*, pages 1–9, 2010.

[20] Pennycook J. Performance analysis of a hybrid mpi/cuda implementation of the naslu benchmark. *InSIGMETRICS Perform. Eval. Rev., 38(4)*, pages 23–29, 2011.

[21] Heinlein J., Gharachorlo K., et al. Integration of message passing and shared memory in the stanford flash multiprocessor. *Scientia et Technica Año XIII, No 37*, 1994.

[22] Universidad Tecnológica de Bolívar. HPCLab - Laboratorio de Computación de Alto Desempeño. URL `http://hpclab.unitecnologica.edu.co/`. Accessed: 2015-02-25.

[23] Gubb D. and Holmes V. Implementation of a condor pool at the university huddersfield. 2013.

[24] Dongarra J. Performance of various computers using standard linear equations software. *University of Tennessee Computer Science Technical Report, CS-89-85, 2014*, page 3, 1986.