UNIVERSIDAD DEL VALLE DE GUATEMALA

Facultad de Ingeniería

Exploración del uso de un cluster en una nube pública para paralelización computacional de una aplicación científica

Trabajo de graduación presentado por Víctor Estuardo Araujo Soto

para optar al grado académico de Licenciado en Ingeniería en Ciencia de la Computación y Tecnologías de la Información

Guatemala
2014

Exploración del uso de un cluster en una nube pública para paralelización computacional de una aplicación científica

UNIVERSIDAD DEL VALLE DE GUATEMALA

Facultad de Ingeniería

Exploración del uso de un cluster en una nube pública para paralelización computacional de una aplicación científica

Trabajo de graduación presentado por Víctor Estuardo Araujo Soto

para optar al grado académico de Licenciado en Ingeniería en Ciencia de la Computación y Tecnologías de la Información

Guatemala
2014

Vo. Bo.

Ing. Bidkar Pojoy
Asesor

Tribunal examinador:

MSc. Douglas Barrios

Ing. Bidkar Pojoy

Ing. Tomás Gálvez

Fecha de aprobación: Guatemala, 4 de diciembre de 2014

PREFACIO

Este trabajo de graduación surge dentro de dos contextos principales. El primero y más general es la tendencia del crecimiento de la computación a escalas sin precedentes. Big Data y Big Compute ya no son temas que se exploran solamente en laboratorios de investigación y en empresas especializadas. Todo tipo de organizaciones empieza a utilizar recursos computacionales a grandes escalas, para atacar problemas computacionales de gran magnitud. Junto con los cambios en la demanda de las organizaciones por más poder computacional, también ha evolucionado la oferta de recursos. El paradigma de computación en la nube continúa creciendo, y a través de nubes públicas provee un medio para adquirir recursos computacionales de manera rápida para casi cualquier necesidad.

El segundo contexto es nuestro contexto local, en Guatemala y en la Universidad del Valle. Para aprovechar las nuevas tendencias en computación para fines educativos, científicos y empresariales es necesario construir la infraestructura computacional y las habilidades y conocimientos adecuados para utilizar y desarrollar estas nuevas tecnologías.

El tema de paralelización surge de la percepción de las necesidades de nuestro contexto local para aprovechar las oportunidades del contexto tecnológico, buscando que la exploración del aprovisionamiento de recursos en la nube sirva para ilustrar la capacidad de conseguir recursos computacionales de forma fácil y económica, y que esto se pueda utilizar para habilitar todo un mundo de oportunidades educativas, científicas y empresariales. ¡Ya no es necesario ir a un laboratorio del extranjero para utilizar un sistema de alto

rendimiento con MPI, o comprar y mantener un arreglo de GPUs para explorar OpenMP y OpenCL, ni ser dueño de un cluster para utilizar Hadoop!

Aunque el tema de paralelismo lleva décadas de investigación y aplicación, su adopción en Guatemala es limitada. En la Universidad del Valle de Guatemala no se cuenta todavía con un cluster o con hardware de GPU que los estudiantes puedan utilizar para aprender sobre computación paralela o para que miembros de la institución puedan desarrollar proyectos que requieran de más recursos computacionales que los que puede ofrecer una sola computadora en la actualidad.

Los detalles más específicos que dieron forma a las elecciones de MPI, Geant4 y la simulación Ultra de detectores de lluvias de rayos cósmicos provienen del acercamiento del Departamento de Física de la Universidad del Valle de Guatemala para participar en la colaboración LAGO (Latin American Gamma Ray Observatory) en 2014.

Dedico esta tesis a mis padres, Mirsa y Víctor, que toda mi vida me han guiado, apoyado y sostenido en las buenas y en las malas. Agradezco su esfuerzo y su cariño con todo mi amor. Estas líneas se escriben hoy por ellos.

Este trabajo no sería posible sin la influencia de la Lda. Zaidy Urrutia, promotora del proyecto LAGO en la Universidad del Valle de Guatemala y de muchas otras actividades científicas; el Dr. Norberto Arzate en el Centro de Investigaciones en Óptica en León, Guanajuato, México, con quien pude utilizar un cluster real y MPI durante el mes de julio de este año; y el Lic. Bidkar Pojoy, que al hacernos investigar sobre clusters en el curso de Sistemas Operativos

llamó mi atención por este tema desde 2013. Mis agradecimientos a cada uno de ellos.

ÍNDICE

LISTA DE CUADROS

LISTA DE GRÁFICOS

RESUMEN

Utilizando los servicios de Elastic Compute Cloud de la nube pública de Amazon Web Services se construyó un cluster de quince nodos (sistema paralelo de memoria distribuida) y se aprovisionó por aparte una instancia con 16 núcleos de procesamiento (sistema paralelo de memoria compartida). Ambos sistemas se configuraron con las librerías de Geant4 para simulaciones físicas y OpenMPI, una implementación de Message Passing Interface (MPI). Se paralelizó exitosamente la simulación Ultra distribuyendo los eventos de Geant4 a través de calendarización estática, logrando una ejecución hasta catorce veces más rápida en el cluster al utilizar sus quince nodos. En la instancia de 16 núcleos se logró una ejecución hasta nueve veces más rápida.

I. INTRODUCCIÓN

La computación es una herramienta utilizada para atacar una enorme diversidad de problemas, en todo el espectro de actividades humanas. Dentro de este conjunto de actividades, algunas se caracterizan por su magnitud y la gran cantidad de recursos computacionales que requieren: tiempo y memoria. Algunos de estos problemas pueden tardar años en ser resueltos en una computadora personal para cualquier caso particular.

Este tipo de problemas surge a menudo en las ciencias y en la ingeniería, donde simulaciones de modelos complicados pueden tomar demasiado tiempo para nuestras escalas de tiempo humano. Una de las estrategias para resolver un subconjunto de esta clase de problemas es utilizar varias computadoras para resolver partes independientes del problema al mismo tiempo, con la ventaja de poder reducir el tiempo de ejecución y aumentar la memoria disponible para la resolución del problema (Silberchatz 2009).

A este uso de recursos computacionales para resolver partes de un problema se le conoce como computación paralela. En ciencias e ingeniería, la computación paralela ha sido de especial importancia para resolver problemas complejos en escalas de tiempo manejables (Barney 2012).

El acceso a sistemas paralelos solía ser limitado, debido a su precio elevado y la complejidad de operar ese tipo de infraestructura. Sin embargo, el modelo de computación en la nube es una opción reciente que permite acceder a una gran diversidad de recursos computacionales. La oferta actual de recursos en la nube permite utilizar y construir sistemas paralelos de diversas clases bajo un modelo de autoservicio bajo demanda, en que se paga solamente por la capacidad utilizada. La inversión en infraestructura y la complejidad de su operación son manejadas por el proveedor del servicio de nube, eliminando estas dos grandes barreras para el uso de sistemas paralelos.

La finalidad de este trabajo es explorar y mostrar la utilización de sistemas paralelos en una nube pública, para la ejecución paralelizada de una aplicación científica. La flexibilidad de los recursos que se pueden aprovisionar a través de nubes públicas y la variedad de soluciones a nivel de software para programar paralelismo proveen muchas opciones para la construcción de sistemas paralelos. Los proveedores de nube pública como Amazon ofrecen computadoras de alto rendimiento en procesamiento, que al tener múltiples núcleos se pueden utilizar como sistemas multiprocesador y al instalar el software adecuado y unir varias de ellas en red pueden funcionar como un cluster, constituyendo un sistema paralelo multicomputador.

El enfoque de este trabajo a explorar paralelización utilizando un cluster dirige la delimitación del tema hacia el uso de MPI (Message Passing Interface), debido a su importancia en sistemas científicos de alto rendimiento. MPI también es capaz de correr en una sola computadora con varios núcleos de procesamiento por lo cual también se eligió evaluar su rendimiento en este tipo de sistema, que se puede obtener fácilmente a través del servicio de nube pública (sin embargo, MPI está optimizado y se suele utilizar para paralelismo entre computadoras; en una sola computadora se suele utilizar OpenMP y similares).

Con base en esto se configuraron los dos sistemas para probar la aplicación paralelizada: un cluster de 15 nodos y una instancia con 16 núcleos de procesamiento. Ambos se configuraron con MPI y con Geant4, un framework para la simulación del paso de partículas a través de la materia. La aplicación elegida para la paralelización es una simulación de un detector de lluvias de rayos cósmicos, llamado Ultra. Ultra se paralelizó distribuyendo los eventos de la simulación equitativamente entre el número de nodos disponibles, utilizando calendarización estática. Se midió el speedup, la razón entre el tiempo de ejecución paralelo y el tiempo de ejecución serial, para una versión de la

paralelización que utiliza comunicaciones colectivas de MPI y otra versión sin comunicaciones colectivas.

II. MARCO TEÓRICO

A. Computación paralela y clusters

La computación paralela es la realización de procesos computacionales de forma concurrente. La capacidad de realizar varios procesos de forma simultánea se contrasta con la computación serial, la cual ejecuta procesos en serie, uno tras otro, sin aprovechar las oportunidades de ejecutar tareas al mismo tiempo para obtener un mejor rendimiento.

El paralelismo en computación se logra a través de varios niveles, que van desde la ejecución concurrente en hardware hasta el diseño explícito de algoritmos paralelos (Gebali 2011). El diseño y fabricación de hardware paralelo está fuera del alcance de la mayoría de organizaciones. Sin embargo, los avances en redes y microprocesadores permiten el uso de varias computadoras como unidad de paralelismo (las cuales, a su vez, pueden tener paralelismo a nivel de procesador), como una opción común y económica para construir sistemas paralelos. A este tipo de sistema distribuido, que consiste en un grupo de computadoras bien acopladas (a menudo en una red local) que comparten recursos para funcionar en conjunto como un sistema se le denomina *cluster* de computadoras (Tanenbaum & Van Steen 2006).

En la actualidad, los clusters de computadoras se utilizan en la mayoría de sistemas parelelos para aplicaciones científicas, incluyendo a las supercomputadoras más poderosas de la actualidad.

1. Taxonomía de Flynn. La taxonomía más popular para la clasificación de arquitecturas paralelas se basa en el trabajo de Flynn (1966). Esta taxonomía se basa en el flujo de instrucciones y de datos en un procesador, cada uno de los cuales constituye una dimensión que sirve para clasificar a los sistemas en cuatro categorías:

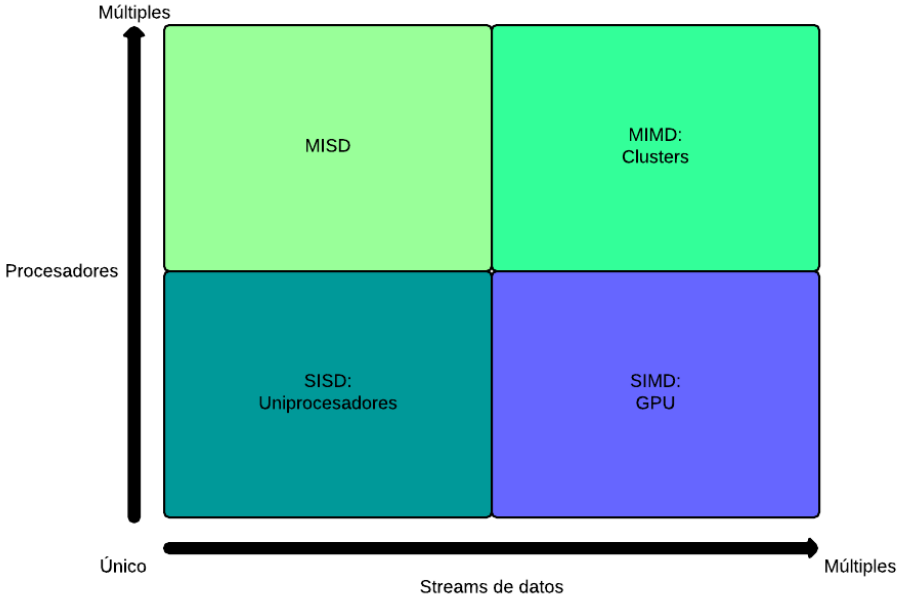
a. Single Instruction Single Data (SISD): Las computadoras convencionales de un solo procesador entran en esta categoría. En el procesador se opera una sola instrucción, sobre un único stream de datos.

b. Single Instruction Multiple Data (SIMD): Esta arquitectura consiste en procesadores idénticos que ejecutan la misma serie de instrucciones sobre conjuntos distintos de datos. Esto simplifica el control, pues se puede usar una misma unidad de control para todo el procesamiento. Los GPU contemporáneos se basan en esta arquitectura.

c. Multiple Instruction Single Data (MISD): Un único stream de datos se envía a varias unidades de procesamiento que pueden ejecutar instrucciones distintas. Este tipo de arquitectura no es comercial ni se construye comúnmente.

d. Multiple Instruction Multiple Data (MIMD): Esta arquitectura se basa en procesadores que pueden trabajar de forma asíncrona sobre conjuntos distintos de datos. Es una arquitectura diversa, que se encuentra tanto en hardware especializado como en clusters formados por sistemas distintos unidos en red y la mayoría de supercomputadoras. Los clusters contemporáneos, en particular el descrito en este trabajo, entran dentro de esta categoría (Gebali 2011).

Figura 1 — Arquitectura de Flynn: Se muestra la arquitectura de Flynn con ejemplos populares de cada categoría

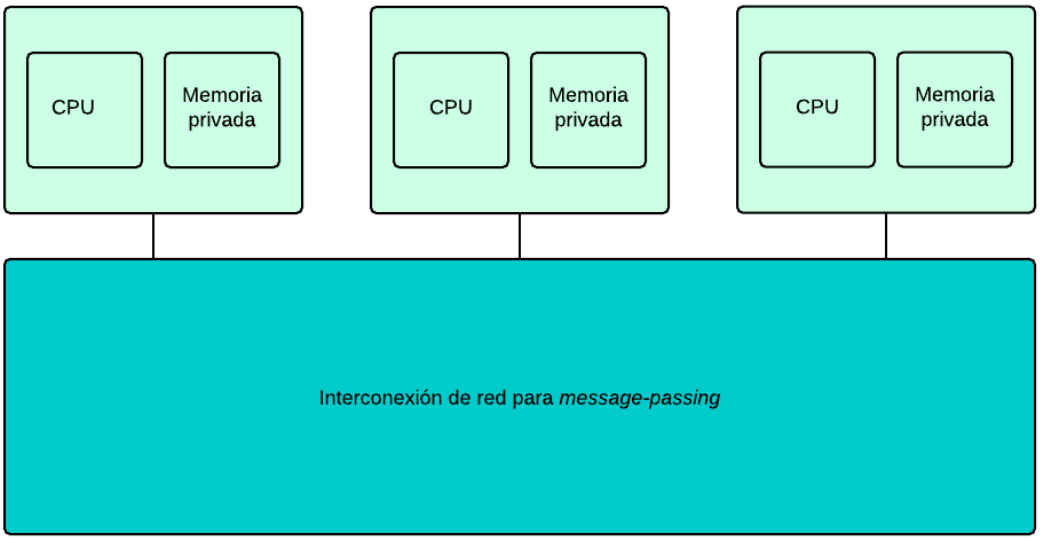


Los sistemas paralelos se distinguen además por su arquitectura de memoria. Esta distinción se basa en las limitaciones que tienen los procesadores para acceder a la memoria de todo el sistema. Las arquitecturas de memoria se clasifican de la siguiente manera (Barney 2014):

a. Memoria compartida: La memoria compartida consiste en bloques de memoria RAM, que pueden ser accesadas por varios CPU de un sistema multiprocesador. Esta característica tiene la ventaja de que los datos pueden ser leídos y modificados directamente por todos los procesadores. Sin embargo, también implican dificultades en la sincronización y la coherencia de la información. Esta arquitectura tiene problemas de escalabilidad, pues al aumentar el número de procesadores que puede acceder a la memoria se congestionan los buses de acceso a ésta, y aumentan las tareas de mantenimiento de la consistencia de la memoria caché (Rauber & Rünger 2007).

b. Memoria distribuida: La memoria distribuida consiste en que cada CPU del sistema multiprocesador posee su propio espacio de memoria. Debido a que los procesadores no tienen acceso directo a la memoria de los demás, estos sistemas deben implementar métodos de comunicación para compartir información. Un método popular en sistemas paralelos científicos es el uso de message passing, utilizando alguna implementación del estándar MPI (Message Passing Interface). El cluster utilizado en este trabajo cae dentro de la clasificación de memoria distribuida, y utiliza además el estándar MPI con la implementación OpenMPI.

Figura 2 — Arquitectura de memoria distribuida



La memoria es escalable bajo este esquema, ya que no se tienen los problemas asociados a tener varios procesadores accediendo a un espacio de memoria compartido. Sin embargo, tiene la desventaja de la complejidad añadida de manejar la comunicación entre procesos y la representación de estructuras de datos globales, pues la memoria lógica global está fragmentada en diferentes sistemas (Barney 2014).

c. Memoria híbrida distribuida-compartida: Esta es la arquitectura utilizada por la mayoría de sistemas de alto rendimiento en la

actualidad. Utiliza varios sistemas de memoria compartida que están conectados entre sí de forma distribuida (Barney 2014).

2. Ventajas de la computación paralela. Aunque la computación paralela puede añadir complejidad en programación, su uso en ciertos contextos permite tener varios beneficios en cuanto a tiempos de ejecución, costos, aprovechamiento de recursos y manejo de problemas complejos.

a. Rendimiento: El uso de procesadores más rápidos permite acelerar la ejecución de los programas. Sin embargo, la creación de procesadores cada vez más rápidos no es una solución viable debido a la potencia necesaria para su operación, a la dificultad de manejar la disipación de calor a altas frecuencias y a los límites físicos para miniaturización de transistores (Rauber & Runger 2007).

Desde un punto de vista técnico y económico, es más viable lograr un alto rendimiento escalable a través del uso de varios procesadores simples que invirtiendo en un solo procesador muy veloz. A medida que nos acerquemos a los límites de optimización de los procesadores, la optimización del rendimiento computacional se basará en gran parte en el mejoramiento del paralelismo en algoritmos, compiladores, sistemas operativos y hardware (Rauber & Runger 2007).

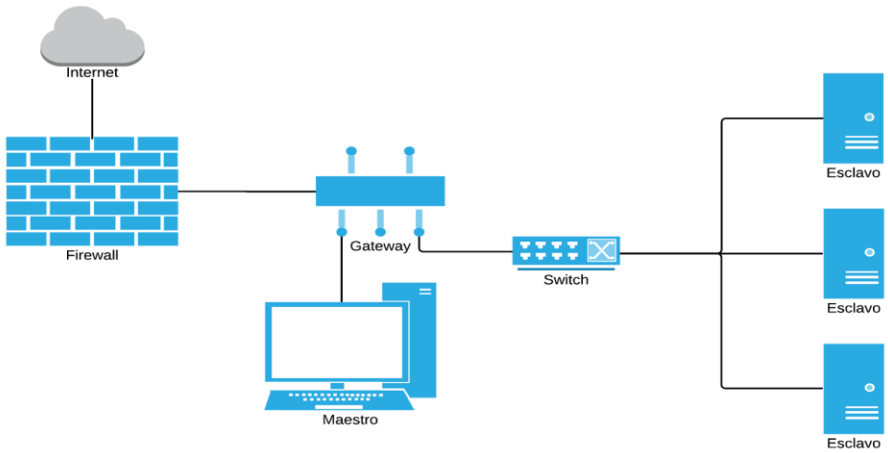
b. Modelado y simulación de problemas complejos: Nuestra forma de percibir el mundo se caracteriza por la ocurrencia de varios fenómenos de manera simultánea. Modelar problemas como los movimientos planetarios, interacciones entre sistemas de moléculas orgánicas, el clima o redes logísticas pueden ser difíciles de modelar de forma serial. La computación paralela permite modelar, simular y entender problemas complejos, conformados por eventos simultáneos que llevan una secuencia temporal, de una mejor manera que la computación serial.

El uso de paralelismo utilizando sistemas distribuidos también beneficia al modelado y la simulación al permitir resolver problemas complejos cuya magnitud impide que sean resueltos en una sola computadora, debido a limitaciones de memoria y tiempo (Barney 2014).

3. Speedup y limitaciones de la computación paralela. El speedup se define como la razón T_1 / T_P donde T_1 es el tiempo de ejecución con un solo procesador y T_P el tiempo de ejecución en P procesadores. El speedup utilizando P procesadores puede ser a lo sumo P, por lo que el número de procesadores es el límite teórico absoluto que se puede alcanzar utilizando computación paralela. En la práctica, el speedup puede ser mucho menor que P debido a los costos en tiempo de sincronización, acceder a memoria compartida o realizar tareas de comunicación entre procesadores (Cormen 2001).

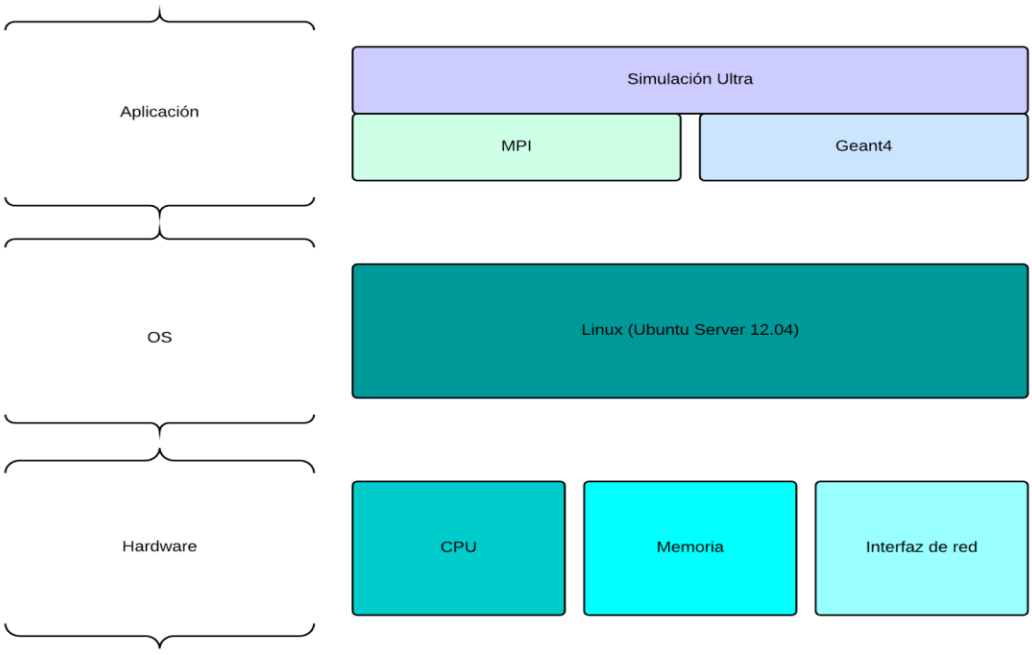
4. Conformación típica de un cluster para aplicaciones científicas. Un cluster está conformado por varias computadoras individuales unidas a través de una red. A las computadoras individuales se les denomina nodos. A menudo se posee un nodo primario que se denomina maestro, y a todos los demás se les conoce como esclavos (Tanenbaum & Van Steen 2006).

Figura 3 — Componentes típicos de un cluster



5. Estructura general de un cluster. Cada nodo individual posee hardware como procesadores y memoria, y software como sistemas operativos. El paralelismo se define en la manera de utilizar los recursos de cada nodo para lograr computación paralela. La mayoría de aplicaciones científicas utilizan el estándar MPI para la comunicación entre los procesos que están corriendo de forma paralela en los nodos del cluster (Barney 2014). En clusters más complejos también pueden existir calendarizadores o colas de tareas y sistemas de archivos distribuidos. Cabe mencionar que en un cluster suelen ocurrir varios niveles de paralelismo, desde paralelismo a nivel de instrucciones en el hardware hasta el paralelismo explícito programado en las aplicaciones que utiliza hilos o nodos como unidades lógicas de ejecución paralela. A continuación se describen los diferentes componentes necesarios para describir el cluster utilizado en este trabajo.

Figura 4 — Componentes de un cluster en capas de organización: Las capas superiores utilizan a las capas inmediatamente inferiores



6. Hardware en los nodos: Arquitectura de Von Neumann.

La arquitectura de Von Neumann es una forma de organizar e implementar las computadoras, descrita en 1945 por el físico John Von Neumann. La mayoría de computadoras electrónicas comerciales hasta la fecha se basan en esta arquitectura, cuya utilización implica un conjunto de características y limitaciones asociadas a su forma de llevar a cabo la computación (Riley 1987).

La arquitectura de Von Neumann se basa en cuatro componentes principales, relacionados respectivamente con operaciones de control para la ejecución de instrucciones; operaciones aritméticas y lógicas; memoria para almacenar instrucciones y resultados; y medios de interacción con los seres humanos que operan la computadora (Riley 1987).

a. Unidad central de procesamiento: La unidad central de procesamiento (CPU, por sus siglas en inglés, Central Processing Unit) ejecuta los programas almacenados en la memoria, obteniendo cada instrucción y ejecutándolas una tras otra. El CPU posee una unidad de control que se encarga de obtener las instrucciones, examinarlas y determinar su tipo, y una unidad aritmética-lógica (ALU, por sus siglas en inglés, Arithmetical Logical Unit) que se encarga de ejecutar las operaciones artiméticas y lógicas como sumas u operaciones booleanas. La unidad de control y la ALU son los componentes que implementan las tareas de control y las operaciones lógicas y aritméticas descritas en la arquitectura de Von Neumann (Tanenbaum 2005).

Para apoyar estas tareas, los CPUs actuales contienen una pequeña memoria de alta velocidad constituida por un número de registros, cada uno de los cuales posee un tamaño particular en bits y una función definida. Dos registros muy importantes son el contador de programa o Program Counter (PC), que apunta a la dirección en memoria de la siguiente instrucción a ejecutar, y el registro de instrucciones o Instruction Register (IR), que contiene la instrucción que se está ejecutando en el momento.

El CPU ejecuta las instrucciones a través de una serie de pasos que se conoce como el ciclo fetch-decode-execute, de la siguiente manera (Tanenbaum 2005):

- Obtener la siguiente instrucción desde la memoria y colocarla en el registro de instrucciones.
- Cambiar el contador de programa para que señale la dirección de la próxima instrucción.
- Determinar el tipo de instrucción que se obtuvo en el paso uno, con la unidad de control.
- Si la instrucción usa una palabra de la memoria, determinar dónde se encuentra.
- Si es necesario, obtener la palabra y colocarla en un registro del CPU.
- Ejecutar la instrucción.
- Ir al paso uno para ejecutar la próxima instrucción.

b. Memoria: Las computadoras contemporáneas poseen memoria de acceso aleatorio, conocida como memoria RAM por sus siglas en inglés (Random Access Memory). Los sistemas de memoria distribuida están conformados por varios CPUs con su memoria RAM asociada, la cual no puede ser leída o escrita directamente por otros CPU. En los sistemas de memoria compartida tenemos varios núcleos de procesamiento que pueden leer y escribir la misma memoria RAM. Los sistemas contemporáneos también pueden utiliza dispositivos como discos duros magnéticos cuando la memoria RAM se acaba, aunque con tiempos de lectura y escritura mucho más largos (Tanenbaum 2005), (Rauber & Rünger 2007).

7. Sistema operativo. El sistema operativo es software que administra los recursos de hardware de la computadora y provee servicios a los programas de usuarios. Cada nodo de un cluster posee un sistema operativo, a menudo basado en Unix, como alguna versión de Linux. En el contexto de un

cluster, el sistema operativo provee servicios de manejo de memoria, I/O y comunicación de red que son utilizados por capas superiores, como MPI o por las aplicaciones de usuario (Silberchatz 2009).

8. MPI. MPI (Message Passing Interface) es una especificación para sistemas paralelos de memoria distribuida, basados en el concepto de *message passing*. Message passing se basa en el envío de mensajes como forma de comunicación entre procesos, los cuales se encargan de procesarlos y realizar una acción apropiada en base al contenido del mensaje. El uso de MPI permite que procesos en computadoras distintas provean servicios encapsulados, de forma eficiente, flexible y portable (Snir *et. al* 1995).

El proceso de estandarización de MPI empezó en 1992, con el objetivo de formar un sistema portable, capaz de funcionar en diversidad de sistemas paralelos de memoria distribuida. En la actualidad, las especificaciones de MPI son designadas por el MPI Forum, que reúne a más de 40 organizaciones representantes de la industria, la academia, desarrolladores y usuarios. Cabe mencionar que MPI no es una librería en sí sino una especificación. Existen varias implementaciones, entre las cuales destacan OpenMPI y MPICH. Estas pueden ser utilizadas desde C, C++, Fortran, Python y Java (Barney 2014).

A pesar de estar diseñado originalmente para sistemas de memoria distribuida, MPI también puede funcionar en sistemas de memoria compartida o sistemas híbridos. Al utilizar MPI se corren varios procesos, usualmente en diferentes CPUs. Cada proceso tiene su propio espacio de memoria y el intercambio de información se basa en *message passing*. El paralelismo es explícito, por lo que el programador debe identificar las oportunidades de paralelismo e implementarlas con las funcionalidades de MPI.

La estructura general de los programas de MPI inicia con la declaración de los archivos de encabezados “mpi.h” o “mpif.h”, en C y Fortran

respectivamente, junto con cualquier otro encabezado o librería que requiera el programa (Barney 2014).

Luego pueden seguir declaraciones de variables y código secuencial. La parte de ejecución paralela debe empezar con la inicialización del ambiente de MPI, que se realiza con el método MPI_Init(&argc, &argv). Esta función se debe llamar una sola vez en todo el programa, y además debe preceder a cualquier otra llamada de función de MPI (Snir *et. al* 1995).

Luego siguen las llamadas de *message passing* requeridas por el programa particular. Al finalizar el trabajo paralelo, se debe llamar al método MPI_Finalize(). Éste finaliza el ambiente de ejecución de MPI, e implica que no se pueden realizar más llamadas a MPI (Snir *et. al* 1995).

9. Software de aplicación utilizado. Para cumplir el objetivo de paralelizar la ejecución de una aplicación científica, se utilizó la aplicación Ultra, que simula un detector para el estudio de lluvias de rayos cósmicos a través del paquete de software Geant4.

a. Geant4: Geant4 es una aplicación especializada en simulación del paso de partículas a través de materia, desarrollada en el Centro Europeo para la Investigación Nuclear (CERN, por sus siglas en francés). Geant4 es el sucesor de Geant3, la herramienta de simulación más utilizada para detectores de física de altas energías. Geant4 fue escrito en C++, siguiendo principios de orientación a objetos y extendiéndose para manejar requerimientos de simulación en áreas de medicina, astrofísica, fisica de rayos cósmicos, ciencia espacial y física de iones pesados (Asai 2007).

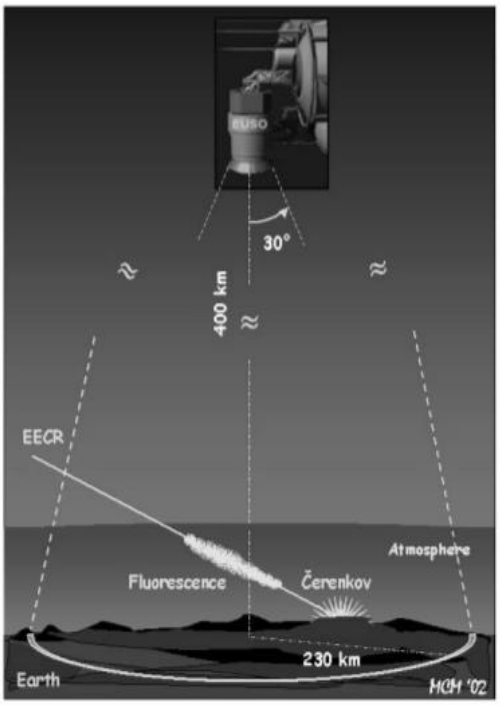
b. Estructura de una aplicación de Geant4: Geant4 es un framework que provee herramientas para la simulación de la interacción de materiales dispuestos en una geometría definida por el usuario y especificando

las interacciones físicas que ocurren. Estas características se especifican extendiendo las clases que conforman la estructura de Geant4. Las simulaciones se organizan de la siguiente manera (Asai 2007):

- Corrida (run): Es una colección de eventos que comparten el mismo detector y condiciones de física. La corrida es el equivalente en simulación a un experimento real. Se define extendiendo la clase G4Run. Adicionalmente, la clase G4RunManager maneja el procesamiento de la simulación y la definición de una clase que extiende G4UserRunAction permite definir hooks para comportamientos específicos definidos por el usuario.
- Evento: Es la unidad básica de simulación en Geant4. Está compuesto por tracks, las cuales al generarse se almacenan en una pila (stack). La simulación va ingresando y eliminando tracks en la pila, hasta que ésta queda vacía, con lo cual concluye la simulación del evento. La clase G4EventManager maneja el procesamiento de cada evento, y G4UserEventAction es el hook opcional para acciones definidas por el usuario.
- Track: Los tracks son “instantáneas” de las partículas. Son manejadas por la clase G4TrackingManager, con el hook opcional de usuario de G4UserTrackingAction.
- Step: Contiene información de “deltas” del estado de la partícula, como cambios de energía. También contiene información de dos puntos en el espacio, que incluyen volumen y material.

c. Ultra: Ultra es una simulación desarrollada utilizando Geant4 para modelar el detector ULTRA, un sistema híbrido compuesto por un sistema de detección de radiación ultravioleta y un arreglo de detectores de centelleo. La simulación del sistema de detección ultravioleta es software libre incluido en los ejemplos de aplicaciones reales de Geant4.

Figura 5 —Principio operacional de los experimentos ULTRA y EUSO



(Fuente: Espirito-Santo 2008).

El sistema de detección ultravioleta registra luz Cerenkov reflejada por el suelo, proveniente de lluvias de rayos cósmicos. La simulación define un lente Fresnel de 457 mm de diámetro, hecho de un material acrílico transmisor de radiación ultravioleta; junto con un tubo fotomultiplicador dentro de una cobertura cilíndrica de aluminio (Tome & Espirito-Santo 2008).

B. Computación en la nube

El uso de computación en la nube es la otra parte central de este trabajo. El paralelismo da las herramientas y técnicas para la ejecución concurrente de tareas computacionales, pero es la computación en la nube la que permite implementar estas técnicas bajo un modelo de consumo particular, caracterizado por pagar solo por los recursos consumidos y por evitar grandes inversiones

para la adquisición de equipo y su mantenimiento. Estas características pueden ser de beneficio para organizaciones (científicas y en general), si sus casos de uso de computación se ajustan bien a este modelo y hacen sentido financieramente.

El National Institute of Standards (NIST) de los Estados Unidos de América definió la computación en la nube de la siguiente manera:
«—Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction» (NIST 2013).

Según esta definición, el modelo de computación en la nube provee disponibilidad de recursos computacionales y tiene como características esenciales las siguientes (NIST 2013):

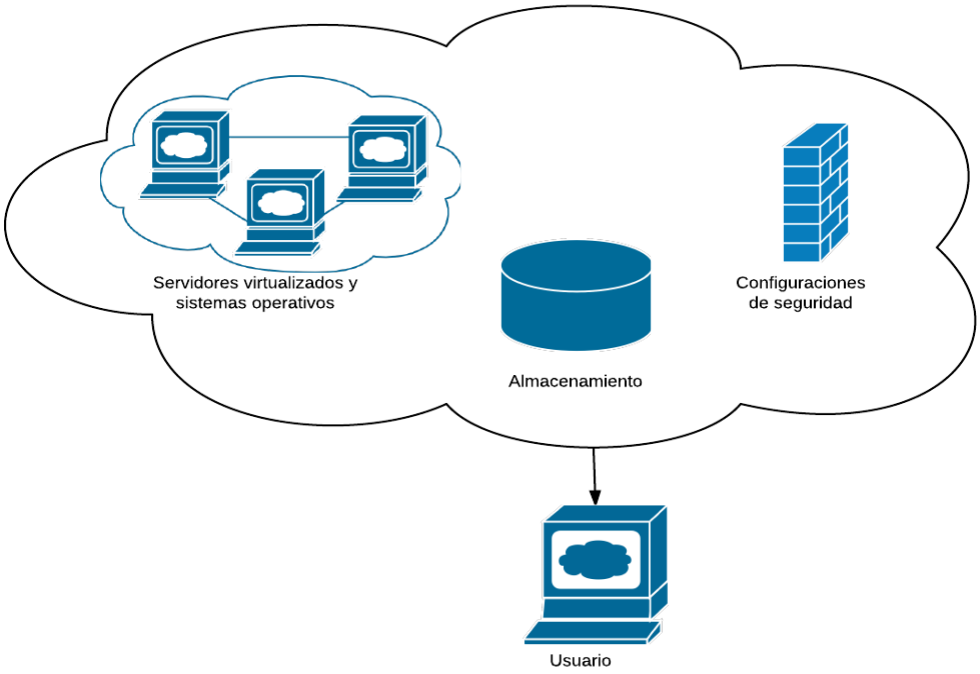
- Autoservicio bajo demanda
- Acceso ancho de red
- Mancomunación de recursos
- Elasticidad rápida
- Servicio medido

1. Modelos de servicio. La computación en la nube se da en tres modelos de servicio: el software como servicio (SAAS, por sus siglas Software as a Service), plataforma como servicio (PAAS, Platform as a Service) e infraestructura como servicio (IAAS, Infrastructure as a Service).

a. IAAS: IAAS provee al usuario con procesamiento, almacenaje de información, funcionalidades de red y otros recursos de computación sobre los cuales el usuario puede desplegar y correr software desde sistemas

operativos hasta aplicaciones. El usuario no controla la infraestructura subyaciente, pero sí controla los sistemas operativos, almacenaje, aplicaciones y algunos aspectos de la conectividad de red (como configuraciones de firewalls) (Glas & Andres 2010).

Figura 6 — Recursos populares provisionados con IAAS: En servicios como EC2 de Amazon el usuario utiliza una interfaz web para adquirir y configurar los recursos.



b. PAAS: En el modelo de servicio de PAAS, se le da al usuario la capacidad de desplegar aplicaciones que puedan correr bajo los lenguajes de programación, librerías, servicios y herramientas respaldadas por el proveedor.

El usuario no controla la configuración de redes, servidores, sistemas operativos o detalles del almacenamiento. Por lo tanto, PAAS posee un nivel de abstracción mayor al de IAAS (Glas & Andres 2010).

c. SAAS: Bajo este modelo, el cliente utiliza solamente la aplicación del proveedor, la cual corre en un ambiente de nube sin que el cliente tenga control sobre la infraestructura de nube (Glas & Andres 2010).

2. Clasificación de nubes. La clasificación del NIST (2013) agrupa a los tres modelos como nubes públicas, privadas, comunitarias e híbridas:

a. Nubes públicas: Se provisionan para ser utilizadas por el público general. Pueden ser operadas por organizaciones académicas, de negocios o de gobierno; con o sin fines de lucro.

b. Nubes privadas: Construyen su infraestructura de nube para uso exclusivo de una organización. Puede ser propiedad o estar operada por la propia organización o por alguna otra.

c. Nubes comunitarias. La infraestructura se construye para ser utilizada por una comunidad que tiene necesidades compartidas. Puede ser propiedad o estar operada por una o más de las organizaciones de la comunidad, una organización ajena o alguna combinación de las anteriores.

d. Nube híbrida: La infraestructura de nube está compuesta por una combinación de nubes comunitarias, públicas y/o privadas distintas, pero que poseen alguna capacidad de interoperación de datos y aplicaciones.

3. Elastic Compute Cloud. El cluster utilizado en este trabajo se provisionó en Elastic Compute Cloud, o EC2, un servicio IAAS de la compañía estadounidense Amazon. EC2 provee computación escalable en la nube bajo demanda. En este modelo, el usuario paga solamente por la capacidad computacional utilizada, y tiene la opción de añadir y eliminar recursos computacionales en minutos o incluso segundos de acuerdo a sus necesidades (Amazon Web Services 2014).

Amazon provee la opción de adquirir instancias spot, las cuales tienen un precio significativamente menor al de las instancias bajo demanda. Amazon subasta su capacidad no utilizada a través de las instancias spot. El usuario

establece un monto máximo a pagar y tiene derecho a utilizar estas instancias hasta que el precio spot exceda el precio que ofreció en la subasta (Amazon Web Services 2014). La disponibilidad de las instancias spot no está garantizada, pero si la aplicación no necesita correr continuamente durante mucho tiempo o si tolera interrupciones las instancias spot son una opción muy económica.

III. ANTECEDENTES

Con la creciente popularidad de servicios de IAAS en nubes públicas, el uso de estos servicios de infraestructura para correr simulaciones empezó a ser explorado por diversos autores.

La investigación durante la primera década del siglo XXI, en la cual surgieron los servicios de nube pública y se popularizaron a finales de ésta, se centraron en analizar el servicio más popular, EC2 de Amazon, para utilizar aplicaciones científicas.

Deelman *et. al* (2008) exploraron los costos asociados de correr una aplicación de astronomía utilizando los servicios Amazon de EC2 para computación y S3 para almacenamiento, examinando los efectos de diferentes planes de ejecución sobre el costo. Los autores señalaron que los modelos comerciales de nube estaban en su infancia, y que en los próximos años habría cambios en el panorama. Al momento del estudio, concluyeron que el uso de S3 era efectivo en costos para aplicaciones que hacían uso de muchos datos.

Edlund & Koopmans (2008) y Assuncao *et al.* (2009) describieron el uso de instancias de EC2 para extender las capacidades de computación de organizaciones científicas.

En 2010, Qiming He *et. al* analizaron el desempeño de las plataformas IAAS de Amazon, GoCloud e IBM utilizando benchmarks de computación paralela y una aplicación real de simulación climatológica de la NASA. Encontraron que las instancias virtualizadas sobre las que corren las aplicaciones no tienen costos significativos sobre el rendimiento, pero que el desempeño de red no era adecuado para aplicaciones HPC altamente acopladas. Señalaron que mejoras moderadas en el desempeño de red traerían

aumentos significativos del rendimiento de las aplicaciones en las nubes públicas analizadas.

Lowenthal *et. al* (2013) analizaron el desempeño de aplicaciones en EC2 añadiendo la métrica de turnaround time (tiempo de espera en colas) además del costo y el desempeño de procesamiento y comunicación. Para aplicaciones de HPC, aunque el desempeño de comunicación en EC2 es inferior en comparación a clusters dedicados en centros de supercomputación, hallaron que el turnaround time puede ser menor.

Las limitaciones halladas en comunicación de red y los análisis complejos de costos de operación están enfocados hacia el espectro más complejo de computación científica. En aplicaciones más pequeñas de HPC, o aplicaciones que no tienen necesidades altas de desempeño de comunicación, el uso de nubes públicas es una solución escalable y económica en el sentido de que no se requiere una gran inversión inicial para acceder a los recursos computacionales. Además, el desempeño de los servicios de nube va en aumento mientras los costos van bajando (Zai 2011).

IV. MARCO METODOLÓGICO

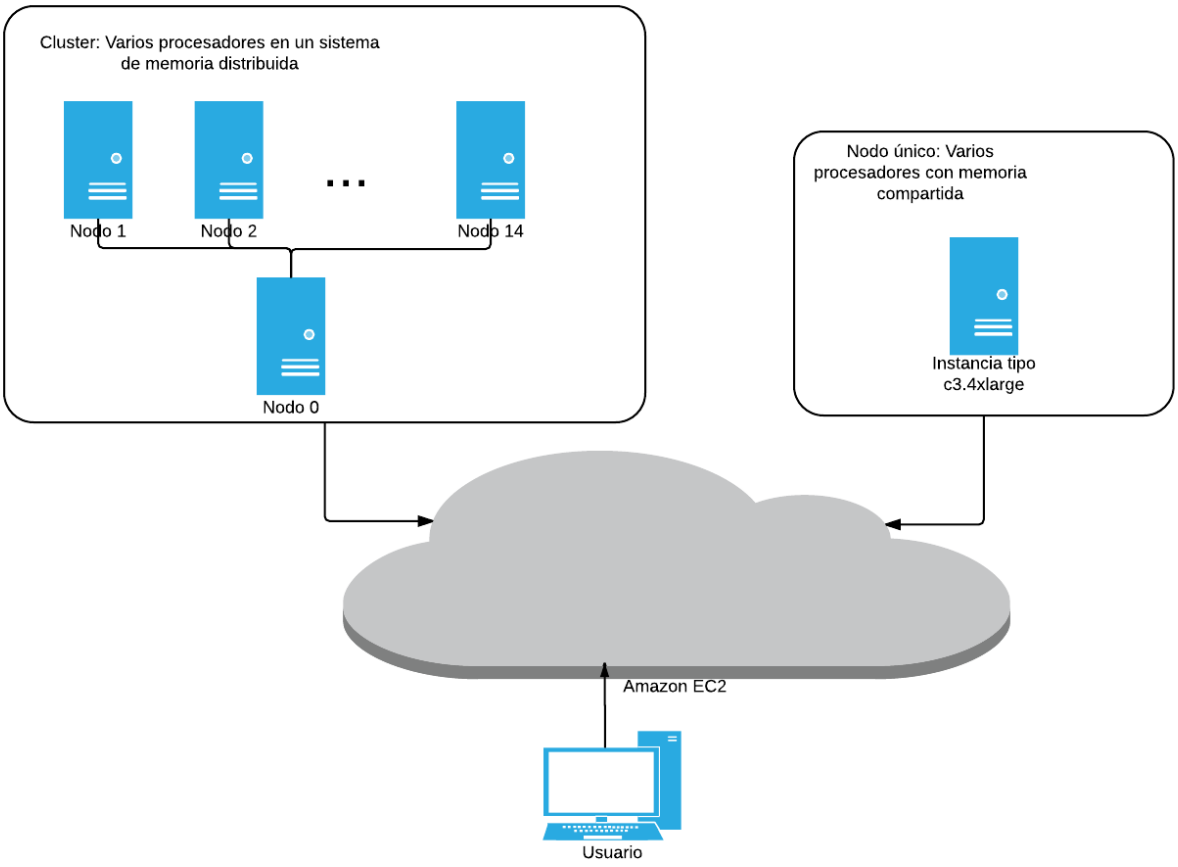
A. Aprovisionamiento de sistemas multicomputador y multiprocesador en la nube

EC2 provee una gran variedad de tipos de instancias con características diferentes de rendimiento de memoria, procesamiento e I/O. Adicionalmente, para la ejecución de aplicaciones científicas podemos utilizar sistemas de memoria distribuida o de memoria compartida. Los clusters son sistemas de memoria distribuida que además son sistemas multicomputador, debido a que están formados por varias computadoras individuales. En EC2 se pueden utilizar también sistemas de memoria compartida provisionando un nodo único con múltiples núcleos de procesamiento, lo cual constituye un sistema multiprocesador.

Con el propósito de comparar el rendimiento de MPI en el cluster con un sistema multiprocesador, se provisionaron dos sistemas diferentes en EC2 para ejecutar la simulación paralelizada. El primer sistema es un cluster beowulf de 15 nodos, cada uno de los cuales es una instancia c3.large. El segundo es un sistema conformado por una única instancia tipo c3.4xlarge el cual posee 16 procesadores.

La instancia c3.4xlarge es un único nodo. Permite probar la simulación paralelizada en una arquitectura de múltiples núcleos, donde al estar estos en el mismo sistema no necesitan comunicarse a través de red.

Figura 7 — Sistemas aprovisionados en EC2: El cluster es un sistema multicomputador, de memoria distribuida. El nodo solitario es un sistema multiprocesador con 16 núcleos, de memoria compartida.



Cabe mencionar que es posible construir un cluster conformado por cualquier tipo de instancias de EC2, incluyendo c3.4xlarge y superiores. Sin embargo, el costo de operación será mayor y para aprovechar la capacidad adquirida al máximo el código podría requerir modificaciones.

1. Características de configuración. Para su funcionamiento, los sistemas creados cuentan con las siguientes características:

- Configuración de /etc/hosts con todos los nodos del cluster para comunicación por nombre de host en lugar de dirección IP.

- Instalación del sistema de archivos distribuido Network File System (NFS), montado en el directorio /home. Todos los nodos tienen acceso a este sistema de archivos, el cual provee un directorio común.
- Instalación de OpenMPI. Esta implementación de MPI permite ejecutar programas que utilizan MPI.
- Keypairs criptográficos para comunicación segura sin contraseñas.

2. StarCluster. Es posible configurar un cluster de manera semiautomática utilizando el programa StarCluster de MIT. StarCluster automatiza el proceso de creación de un cluster en base a las características definidas en un archivo de configuración. Además de realizar las tareas anteriores, StarCluster instala la aplicación OpenGridScheduler para manejar y calendarizar la ejecución distribuida de las tareas indicadas por los usuarios. StarCluster instala también automáticamente varios paquetes de software utilizados comúnmente para tareas científicas. StarCluster está integrado con EC2 y permite agregar automáticamente volúmenes de almacenamiento de Amazon y configuraciones de seguridad. StarCluster también automatiza tareas de administración y operación como enviar y extraer archivos del cluster, cambiar el número de nodos y crear imágenes. Por todas estas ventajas, StarCluster se utilizó para provisionar el cluster final con el que se obtuvieron los resultados, y se recomienda su uso para la creación de clusters en EC2.

B. Instalación de Geant4

Geant4 se debe configurar, compilar e instalar desde código fuente. La compilación de Geant4 tiene como dependencia la herramienta de compilación Cmake, pero puede tener otras dependencias si se eligen opciones de configuración extendidas. Para instalar Geant4 se realizaron las siguientes acciones:

El primer paso para la instalación es tener el código fuente en el sistema. El código fuente se obtuvo desde el sitio web del Centro Europeo para la Investigación Nuclear (CERN). Debido a que la interacción con el cluster no es gráfica, la descarga se realizó desde la consola con el comando *wget* y la URL de la descarga. El código fuente viene comprimido y empaquetado. Se extrajo con el comando *tar* hacia el directorio de usuario.

La construcción de Geant4 se realiza con Cmake, tomando como parámetros el directorio del código fuente de Geant4 y las opciones de instalación.

- -DGEANT4_INSTALL_DATA=ON: Instala conjuntos de datos utilizados por Geant4 en las simulaciones de procesos físicos
- -DCMAKE_INSTALL_PREFIX: Directorio en que se instala la librería de Geant4 después de la compilación. Se eligió el directorio de instalación compartido en NFS /home/geant4, para que la instalación pudiera ser accesada por todos los nodos del cluster.

El proceso de construcción de Geant4 con cmake genera makefiles de Unix. Para realizar la compilación del código fuente se ejecuta el comando make desde el directorio de construcción. Es posible utilizar la opción -j para especificar el uso de varios procesadores si están disponibles, para acelerar el proceso.

La instalación se realiza con el estándar *make install* en el directorio de construcción. Las librerías se instalarán en el directorio especificado en la opción -DCMAKE_INSTALL_PREFIX en el paso de construcción.

C. Paralelización de la aplicación

Se identificó el procesamiento de eventos dentro de la aplicación Ultra como candidatos a ejecución paralela, debido a que estos son independientes

entre sí (Sutherland, Miyajima & Date 2007) (Asai 2011). Se eligió utilizar una calendarización estática y dividir los eventos dentro del número de procesadores disponibles. Debido a que el tiempo de ejecución no difiere considerablemente entre eventos, bajo este esquema no es necesario implementar balanceo de cargas, ya que cada procesador tendrá a su cargo aproximadamente la misma cantidad de eventos durante tiempos similares (Sutherland, Miyajima & Date 2007).

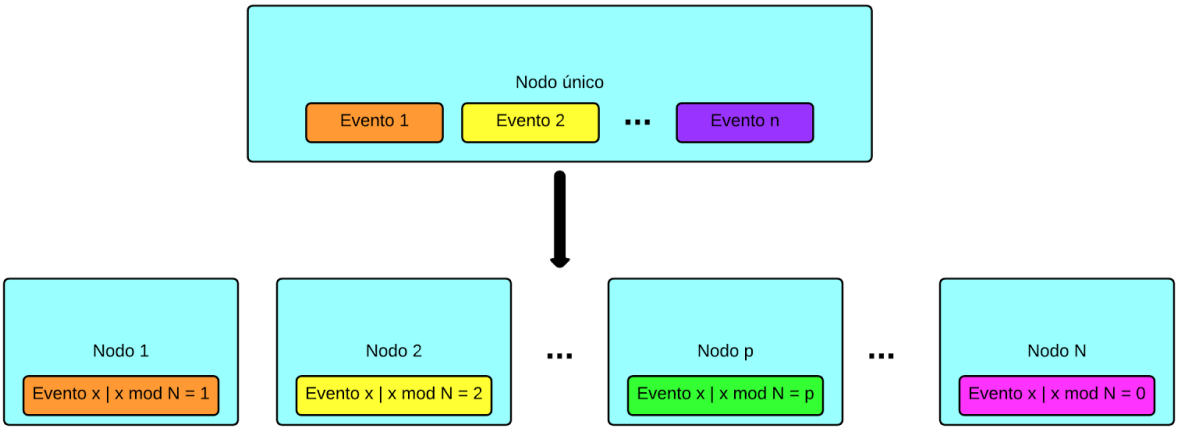
La generación de eventos se define en la extensión de la clase G4VUserPrimaryGeneratorAction, que corresponde a la clase UltraPrimaryGeneratorAction dentro de la aplicación Ultra.

La calendarización estática se basa en dividir las tareas a paralelizar (eventos de Geant4) equitativamente entre los procesadores disponibles, a diferencia de otros tipos de calendarización que distribuyen las tareas dinámicamente entre los procesadores a medida que terminan su trabajo. Para que cada procesador determine los eventos que debe trabajar, debe conocer su rango y el número total de procesadores.

1. Asignación de eventos a los diferentes procesadores. Sea E_n el conjunto de eventos asignados al procesador con rango n, e_i el i-ésimo evento, *N* el número total de procesadores . Entonces se le asigna el evento si se cumple que i m*od N es* igual a *n*:

$$E_n = \{ e_i \mid i \bmod N = n \}$$

Figura 8 — Distribución de los eventos en distintos nodos: En vez de procesar todos los eventos en un único nodo (parte superior), los eventos se distribuyen de manera equitativa entre los nodos disponibles.



2. Manejo de los resultados de la simulación. La distribución de las tareas hace que los resultados de la simulación estén dispersos en procesos distintos. Se implementó una versión que utiliza métodos colectivos de comunicación para recolectar los resultados de la simulación en el nodo con rango 0; y otra que en lugar de utilizar comunicación colectiva almacena los resultados en el sistema de archivos.

Ambos métodos tienen ventajas y desventajas. Se incluyeron para mostrar dos acercamientos posibles para la distribución de aplicaciones científicas. El método que utiliza comunicación colectiva tiene un grado más alto de acoplamiento y una complejidad mayor. Tiene la ventaja de que permite la transmisión de información entre nodos y que puede combinar o centralizar los resultados finales. El método que no involucra comunicaciones colectivas simplemente distribuye las tareas entre los distintos nodos. Los resultados deberán ser combinados luego por otros métodos. Este método es más simple y si el problema o el programa no requiere de intercambio de mensajes o información se puede utilizar llamando al ejecutable de MPI (como mpirun)

directamente, el cual se encargará de ejecutar el programa en los procesadores indicados.

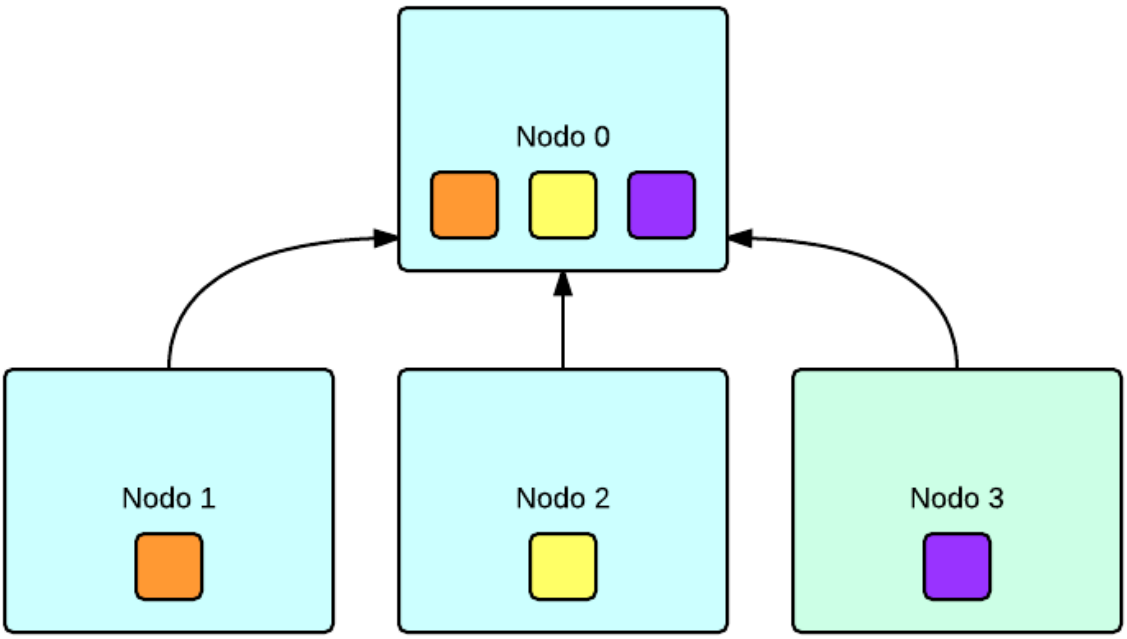
3. Combinación de resultados con MPI. Debido a que el cluster es un sistema de memoria distribuida, los nodos no pueden acceder a los datos en los otros nodos. MPI implementa una serie de patrones de comunicación colectiva, en respuesta a patrones que surgen a menudo dentro del procesamiento computacional en sistemas de memoria distribuida. Las rutinas de comunicación colectiva son comportamientos que involucran a todos los procesos que forman parte de un conjunto de procesos definidos por el usuario. Se basan en procesos de sincronización, movimiento de datos y computaciones colectivas entre varios procesos. Las rutinas de comunicación colectiva utilizadas fueron las siguientes:

a. MPI_Barrier: Rutina de sincronización que los procesos de un comunicador hasta que todos hayan alcanzado el punto de sincronización. Después de que han alcanzado este punto, los procesos continúan su ejecución.

b. MPI_Gather: En esta operación de movimiento de tipos de datos primitivos, cada uno de los procesos envía una pieza de información hacia un nodo raíz, el cual los reúne en un arreglo.

c. MPI_Gatherv: Extensión de MPI_Gather, que toma un arreglo de valores desde cada proceso y los combina en un único arreglo en el nodo raíz.

Figura 9 — Ilustración de operaciones colectivas utilizadas para reunir información hacia un nodo: Las operaciones colectivas MPI_Gather y MPI_Gatherv toman elementos de cada nodo y los reúnen hacia un nodo determinado.



Después del procesamiento del evento, el proceso contiene una lista de hits con sus energías. Estas listas tienen un tamaño variable entre procesos, resultado del evento simulado. Debido a este tamaño variable, no es posible utilizar MPI_Gather o MPI_Gatherv directamente. Primero se debe conocer cuántos hits se deben recolectar desde cada proceso, lo cual se implementó con MPI_Gather recolectando la información reportada del número de hits desde cada nodo hacia el nodo con rango 0.

Conociendo el número de hits desde cada nodo, se computa un arreglo con los offsets correspondientes a cada lista, y se recolectan los arreglos con la información utilizando MPI_Gatherv hacia el nodo con rango 0. En el nodo 0, la

información se escribe hacia el histograma y finalmente el resultado se guarda como un archivo.

Figura 10 — Simulación sin operaciones colectivas: La modificación a Ultra sin operaciones colectivas guarda los resultados de la simulación en el sistema de archivos, generando un archivo separado para cada proceso.

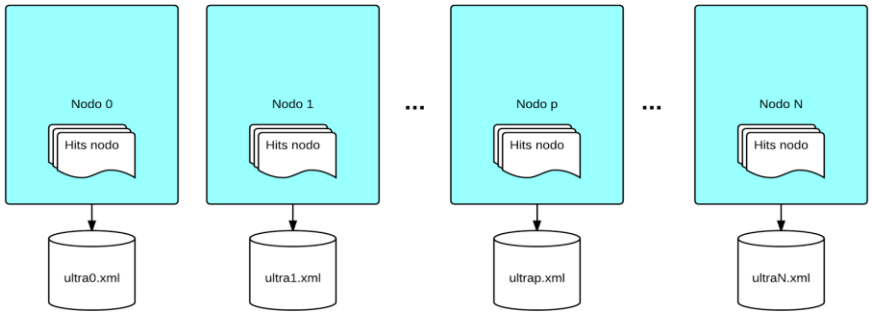
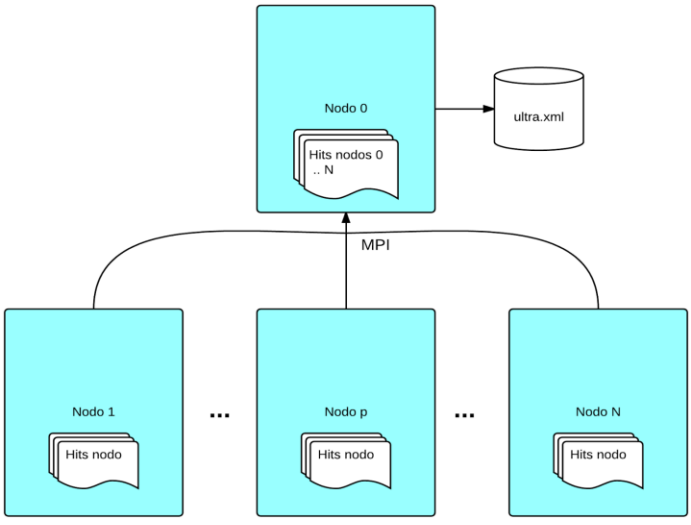


Figura 11 — Simulación con operaciones colectivas: La modificación con comunicación colectiva recolecta los resultados disponibles en cada nodo hacia el nodo 0, y al finalizar guarda los resultados centralizados en el sistema de archivos.

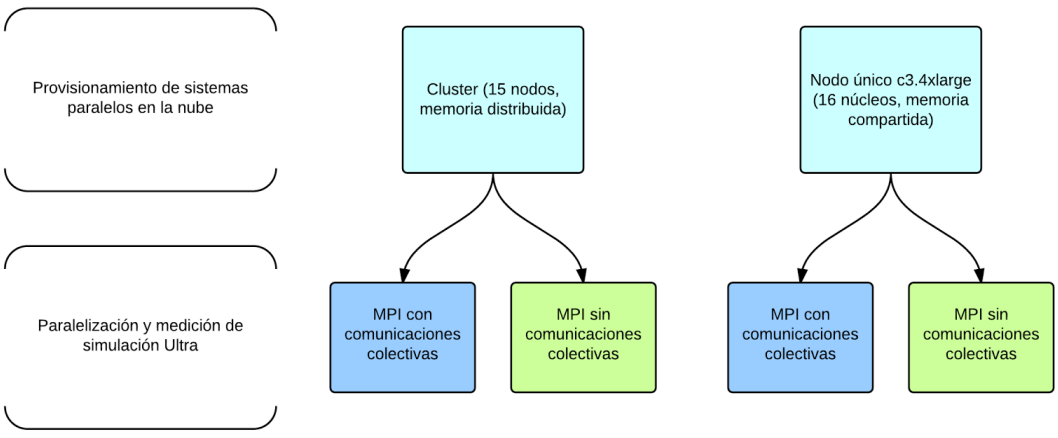


D. Medición del speedup de las simulaciones

Las simulaciones se corrieron bajo el usuario sgeadmin, compilando desde esta cuenta el código de la simulación disponible en el directorio /home. La medición de tiempo de corrida de la simulación incluye el tiempo de usuario, tiempo real y tiempo de sistema. Respectivamente, estas mediciones indican el tiempo que el código pasó ejecutándose en userspace, el tiempo de reloj desde la inicialización hasta la finalización del proceso y el tiempo de ejecución en espacio de kernel del sistema operativo. El tiempo real es el de interés para la medición del desempeño de la aplicación en la nube, ya que es el indicador del tiempo de corrida de la simulación.

Se midieron los tiempos de ejecución de la simulación veinte veces para cada caso desde un procesador hasta quince. Los resultados se resumen con la media, la mediana y la desviación estándar en segundos, y el speedup de la mediana con base en el máximo speedup teórico absoluto. Estas mediciones se realizaron para todas las combinaciones posibles de simulación con o sin comunicación colectiva, tanto en el cluster como en la instancia c3.4xlarge.

Figura 12 — Mediciones realizadas en los sistemas aprovisionados



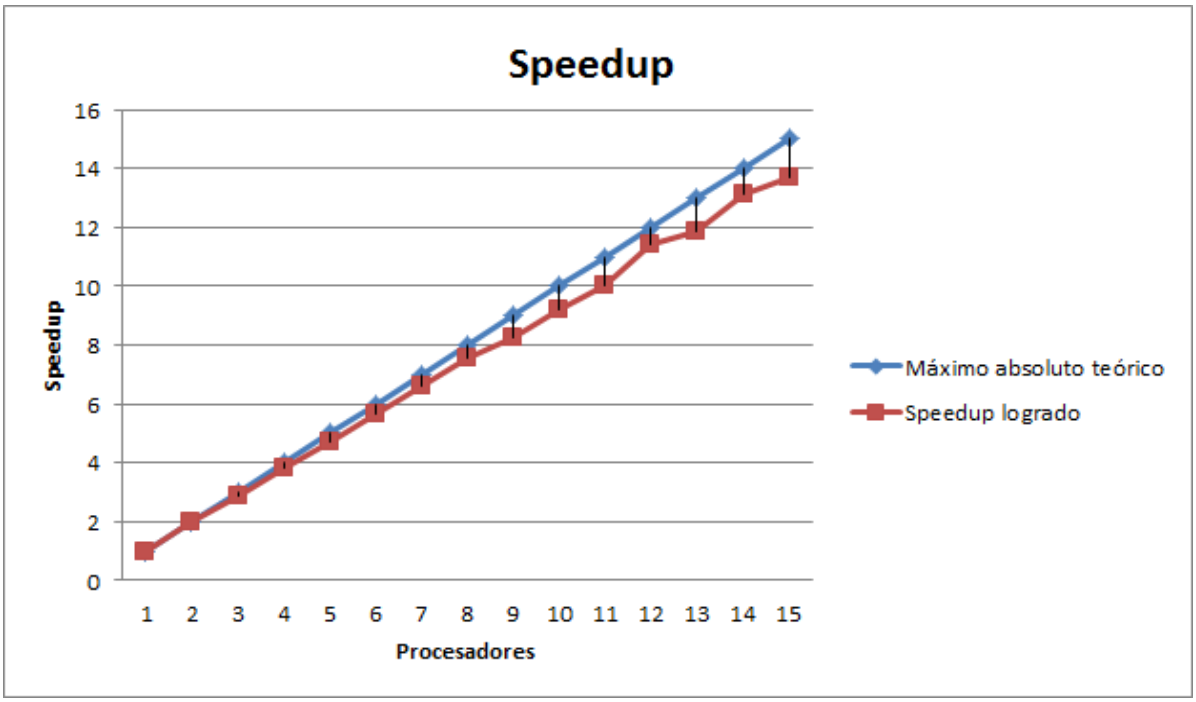
V. RESULTADOS

A. Simulación en cluster con comunicación colectiva

Cuadro 1 — Tiempos de ejecución y speedup para simulación en cluster con comunicación colectiva

Procesadores	Media (s)	Mediana (s)	Desviación estándar (s)	Speedup mediano
1	42.18	42.26	1.86	1
2	21.59	21.35	1.28	1.98
3	14.67	14.8	1.31	2.86
4	11.08	11.08	1.02	3.82
5	8.87	8.93	0.87	4.73
6	7.52	7.49	0.81	5.65
7	6.35	6.38	0.72	6.62
8	5.59	5.59	0.62	7.57
9	5.08	5.12	0.69	8.25
10	4.63	4.58	0.67	9.23
11	4.2	4.22	0.61	10.02
12	3.77	3.71	0.54	11.4
13	3.55	3.57	0.49	11.85
14	3.26	3.22	0.51	13.12
15	3.08	3.09	0.5	13.67

Figura 13 — Speedup para simulación en cluster con comunicación colectiva



B. Simulación en cluster sin comunicación colectiva

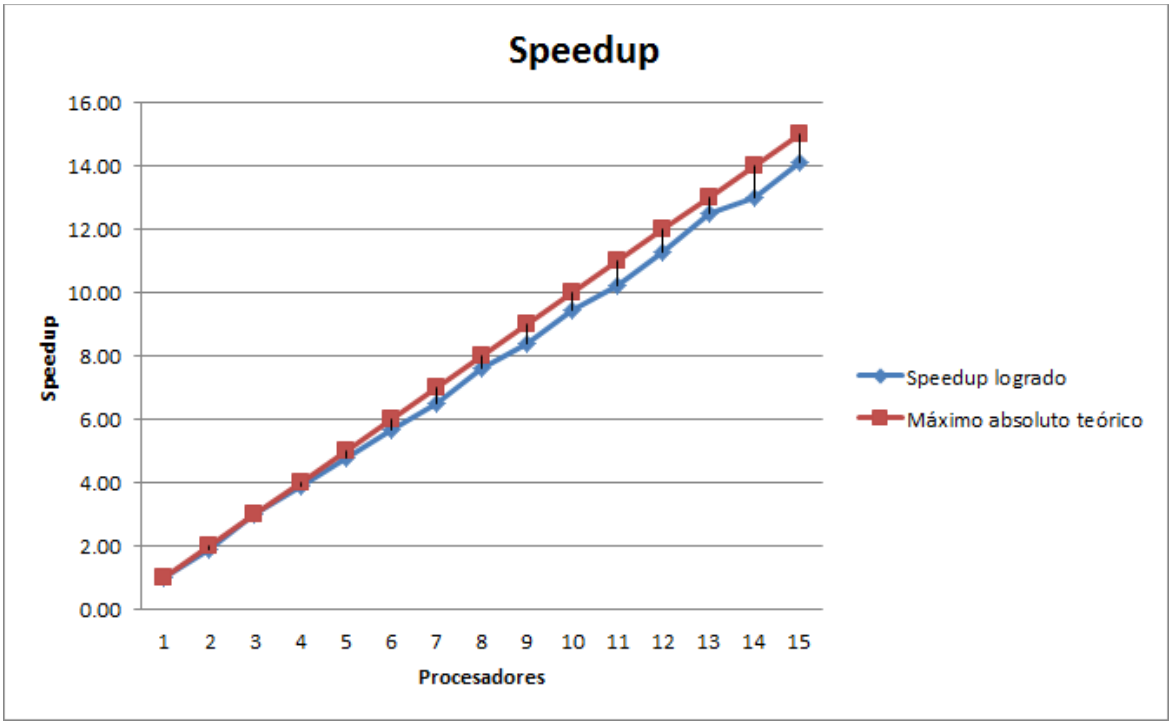
Cuadro 2 — Tiempos de ejecución y speedup para simulación en cluster sin comunicación colectiva.

Procesadores	Media (s)	Mediana (s)	Desviación estándar (s)	Speedup
1	42.44	42.68	1.44	1
2	22.22	22.2	1.6	1.92
3	14.31	14.2	1.45	3.01
4	10.85	10.9	0.86	3.92
5	8.85	8.92	0.93	4.78
6	7.51	7.5	0.66	5.69

Continuación cuadro 2

Procesadores	Media (s)	Mediana (s)	Desviación estándar (s)	Speedup
7	6.56	6.58	0.59	6.49
8	5.64	5.61	0.7	7.61
9	5.03	5.09	0.67	8.39
10	4.61	4.51	0.6	9.46
11	4.21	4.18	0.58	10.22
12	3.79	3.79	0.48	11.26
13	3.49	3.43	0.54	12.46
14	3.35	3.29	0.53	12.97
15	3.08	3.03	0.49	14.09

Figura 14 — Speedup para simulación en cluster sin comunicación colectiva

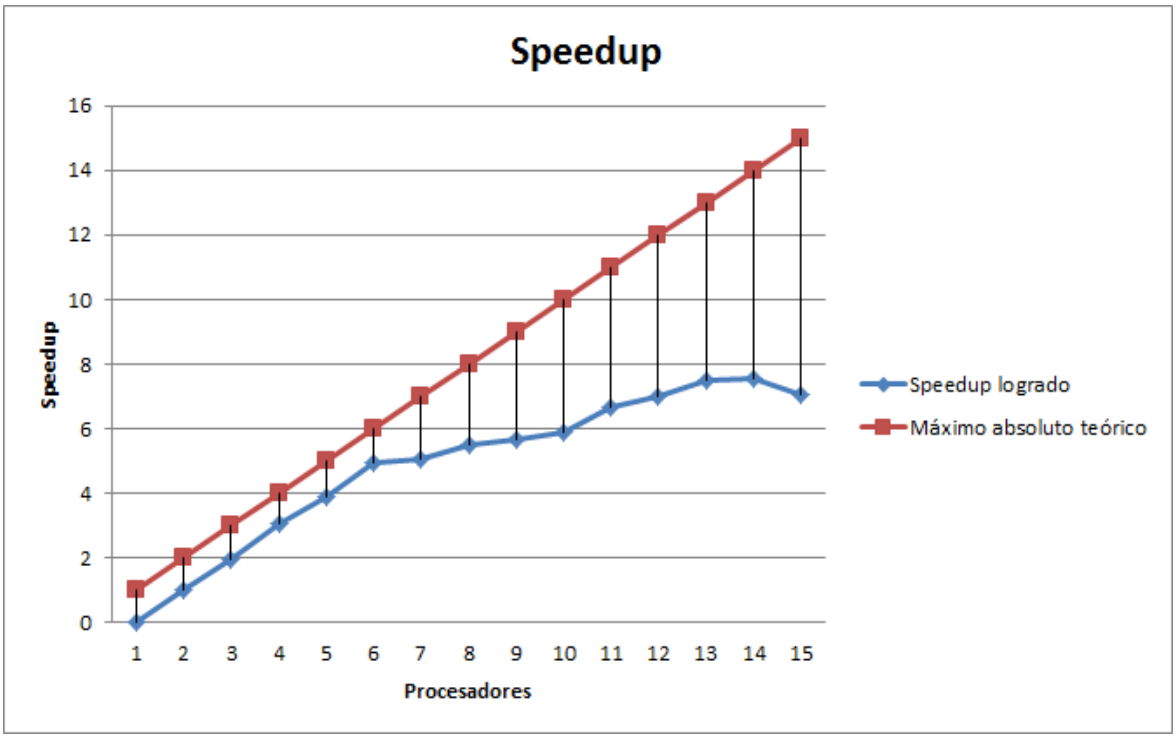


C. Simulación en instancia c3.4xlarge con comunicación colectiva

Cuadro 3 — Tiempos de ejecución y speedup para simulación en instancia c3.4xlarge con comunicación colectiva

Procesadores	Media (s)	Mediana (s)	Desviación estándar (s)	Speedup mediano
1	29.19	29.41	1.43	1
2	14.92	15.15	1.28	1.94
3	9.64	9.66	0.79	3.05
4	7.51	7.6	0.7	3.87
5	6.1	5.96	0.72	4.93
6	5.92	5.83	0.88	5.05
7	5.41	5.35	0.81	5.5
8	5.18	5.18	0.94	5.68
9	5	4.99	1.09	5.89
10	4.47	4.42	0.87	6.66
11	4.38	4.2	1.16	7.01
12	3.96	3.92	0.75	7.51
13	3.92	3.9	0.79	7.54
14	4.1	4.16	0.97	7.07
15	3.68	3.61	0.86	8.15

Figura 15 — Speedup para simulación en instancia c3.4xlarge con comunicación colectiva



D. Simulación en instancia c3.4xlarge sin comunicación colectiva

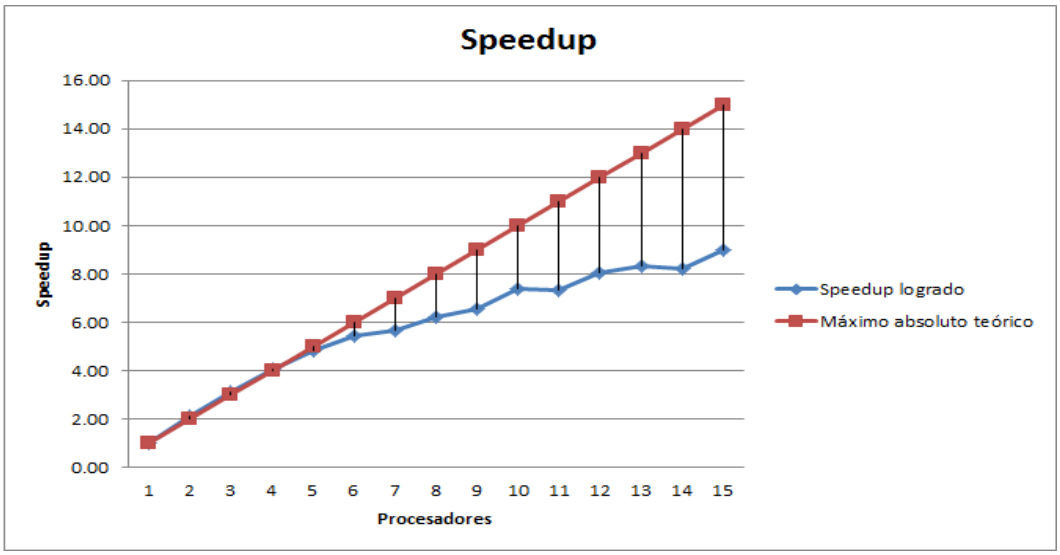
Cuadro 4 — Tiempos de ejecución y speedup para simulación en instancia c3.4xlarge sin comunicación colectiva

Procesadores	Media (s)	Mediana (s)	Desviación estándar (s)	Speedup mediano
1	30.7	30.97	1.43	1
2	14.54	14.48	0.93	2.14
3	9.99	10	1.1	3.1

Continuación cuadro 4

Procesadores	Media (s)	Mediana (s)	Desviación estándar (s)	Speedup mediano
4	7.59	7.59	0.94	4.08
5	6.41	6.39	0.73	4.85
6	5.81	5.66	0.96	5.47
7	5.51	5.48	0.97	5.66
8	4.97	4.99	0.79	6.21
9	4.66	4.72	0.83	6.56
10	4.26	4.19	0.77	7.4
11	4.4	4.22	1.14	7.35
12	3.9	3.84	0.83	8.08
13	3.9	3.72	0.91	8.33
14	3.78	3.77	0.76	8.21
15	3.52	3.44	0.77	9

Figura 16 — Speedup para simulación en instancia c3.4xlarge sin comunicación colectiva



VI. ANÁLISIS DE RESULTADOS

Los resultados obtenidos muestra que para todos los casos se obtuvieron tiempos de corrida más rápidos al utilizar más procesadores. Sin embargo, los speedups obtenidos en el cluster son mayores que en el sistema multiprocesador, debido a que MPI está diseñado para sistemas multicomputador, y las simulaciones probadas no utilizan ninguna optimización para sistemas de memoria compartida.

La escalabilidad obtenida en el cluster es mucho mayor que en la instancia c3.4xlarge. El speedup en el cluster crece con una tendencia casi lineal, logrando un speedup mediano de 13.67 para la simulación con comunicaciones colectivas y 14.09 para la simulación sin comunicación colectiva. Esto se contrasta con los speedups de 8.15 y 9.00 en la instancia c3.4xlarge, donde se observa que el crecimiento del speedup decae rápidamente al pasar de seis procesadores.

Las desviaciones estándar muestran la dispersión de los tiempos de corrida respecto a la media. La variabilidad se atribuye a factores como calendarización de procesos, tiempos variables de acceso a memoria, I/O, sincronización y otros, que toman lugar en el hardware y el sistema operativo; y a los factores estocásticos de la simulación de Monte Carlo.

Para todos los casos, la desviación estándar está por debajo de dos segundos. Sin embargo, los resultados en la instancia c3.4xlarge también muestran comportamientos más variables. Se puede observar que la desviación estándar de las mediciones en el cluster tiende a disminuir a medida que se aumenta el número de procesadores y se reduce el tiempo de corrida. En la instancia c3.4xlarge se observan aumentos ocasionales de la desviación estándar a medida que se aumenta el número de procesadores. Para 15 procesadores, la desviación estándar de los tiempos de ejecución es de 0.50

segundos para el caso con comunicaciones colectivas y 0.49 segundos para el caso sin comunicaciones colectivas en el cluster. En la instancia c3.4xlarge, las desviaciones estándar son 0.86 y 0.77 segundos, respectivamente.

En ocasiones, el speedup medido en la instancia c3.4xlarge es ligeramente mayor al número de procesadores P. Esto no es una contradicción del límite teórico de un máximo speedup igual a P, pues estas discrepancias se deben a la variabilidad en los tiempos de ejecución, los cuales se determinaron empíricamente y no son determinísticos.

VII. CONCLUSIONES

Con la realización de este trabajo se mostró la viabilidad de construir un cluster utilizando la infraestructura provista por Amazon Web Services para paralelizar una aplicación científica. La simulación Ultra, basada en Geant4, se paralelizó con MPI y se logró procesar sus eventos en paralelo utilizando calendarización estática.

El cluster creado se configuró exitosamente dentro de Amazon Web Services utilizando instancias de alto rendimiento para procesamiento, aprovechando la opción de instancias económicas tipo spot. En cumplimiento del objetivo general de crear un cluster científico, las imágenes creadas para el cluster incluyen OpenMPI y las librerías de Geant4. También incluyen librerías científicas de Python, Open Grid Scheduler, Network File System y configuraciones de seguridad para funcionalidad general o para aplicaciones que se le puedan dar en el futuro.

El objetivo de paralelización se cumplió realizando modificaciones al código de Ultra, programando explícitamente la distribución de los eventos con MPI. Para ilustrar la capacidad de solamente distibuir tareas con MPI o utilizarlo también para comunicación entre procesos se creó una versión que utiliza comunicaciones colectivas de MPI y otra que no las utiliza.

Finalmente, se demostró la efectividad de la paralelización midiendo su desempeño utilizando de uno a quince procesadores. Fue posible comparar el desempeño de la paralelización más allá del cluster contemplado originalmente, ejecutando también la simulación en varios núcleos de procesamiento de una misma computadora de alto rendimiento (la instancia c3.4xlarge) para comparar su desempeño con el del cluster. Los resultados muestran que la ejecución en el cluster fue más escalable, logrando mejores speedups que en la instancia

c3.4xlarge, la cual después del uso de seis procesadores mostró un decaimiento rápido en el crecimiento del speedup.

Los resultados obtenidos con el cluster son prometedores. Una simulación de Ultra que tardaría 15 horas siendo ejecutada con un solo procesador, se podría reducir a poco más de una hora. Con suficientes recursos, el tamaño del cluster puede ser variado para lograr una ejecución lo suficientemente rápida dentro de sus límites de escalabilidad.

VIII. RECOMENDACIONES

El propósito de este trabajo de graduación era fundamentalmente exploratorio. Está orientado al uso de MPI en un cluster, lo cual implica un enfoque en memoria distribuida y un modelo de message passing para el paralelismo. Sin embargo, el mundo del paralelismo y la computación de alto rendimiento es muchísimo más amplio y queda mucho por explorar.

La oferta de infraestructura de los proveedores de nube actuales es amplia. Es posible aprovisionar sistemas multiprocesador de alto rendimiento, que pueden tener hasta 32 núcleos de procesamiento y 60 GB de RAM. Amazon ofrece también instancias de GPU con 1536 núcleos CUDA y 4 GB de VRAM. Estas ofertas se pueden utilizar para explorar y desarrollar una gran variedad de proyectos basados en memoria compartida, que es la otra gran arquitectura que no se trató en este trabajo.

La simulación utilizada en este trabajo tiene requerimientos más altos de procesamiento que de memoria o I/O. Toda una clase de problemas, como los relacionados con Big Data, tienen requerimientos altos de I/O y/o memoria. Amazon ofrece instancias optimizadas para memoria o I/O que se pueden utilizar para este tipo de problemas, y es un área interesante para explorar por la relevancia actual alrededor del tema de Big Data.

Además de explorar los aspectos técnicos del uso de nubes públicas, computación de alto rendimiento y paralelismo, es necesario tratar también los aspectos humanos relacionados con su uso a través de la creación de productos, políticas y nuevos estudios orientados a estos temas; de manera que estas herramientas sean adoptadas y utilizadas para educación, investigación e incluso negocios en Guatemala y en la Universidad del Valle de Guatemala.

IX. REFERENCIAS

1. Abraham Silberschatz. 2009. Operating System Concepts (8th ed.). John Wiley & Sons Software.

2. Åke Edlund, Maarten Koopmans, Zeeshan Ali Shah, Ilja Livenson, Frederik Orellana, Jukka Kommeri, Miika Tuisku, Pekka Lehtovuori, Klaus Marius Hansen, Helmut Neukirchen, and Ebba Hvannberg. 2011. Practical cloud evaluation from a nordic eScience user perspective. In *Proceedings of the 5th international workshop on Virtualization technologies in distributed computing* (VTDC '11). ACM, New York, NY, USA, 29-38. DOI=10.1145/1996121.1996129 http://doi.acm.org/10.1145/1996121.1996129

3. Amazon Web Services. 2014. AWS | Amazon Elastic Compute Cloud. http://aws.amazon.com/ec2/

4. Andrew S. Tanenbaum and Maarten Van Steen. 2006. Distributed Systems: Principles and Paradigms (2nd ed.). Prentice Hall PTR, Upper Saddle River, NJ, USA.

5. Andrew S. Tanenbaum. 2005. Structured Computer Organization (5th Edition). Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

6. Aniruddha Marathe, Rachel Harris, David K. Lowenthal, Bronis R. de Supinski, Barry Rountree, Martin Schulz, and Xin Yuan. 2013. A comparative study of high-performance computing on the cloud. In *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing* (HPDC '13). ACM, New York, NY, USA, 239-250. DOI=10.1145/2462902.2462919 http://doi.acm.org/10.1145/2462902.2462919

7. Asai, M. 2006. Introduction to Geant4. Stanford Linear Accelerator Center. geant4.slac.stanford.edu/tutorial/mcgill06/Kernel1.pdf

8. Atanas Radenski. 2012. Integrating data-intensive cloud computing with multicores and clusters in an HPC course. In *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education* (ITiCSE '12). ACM, New York, NY, USA, 69-74. DOI=10.1145/2325296.2325316 http://doi.acm.org/10.1145/2325296.2325316

9. Barney, B. 2012. Introduction to parallel computing. https://computing.llnl.gov/tutorials/parallelcomp/

10. Emanuel Ferreira Coutinho, Gabriel Paillard, and José Neuman de Souza. 2014. Performance analysis on scientific computing and cloud computing environments. In *Proceedings of the 7th Euro American Conference on Telematics and Information Systems* (EATIS '14). ACM, New York, NY, USA, DOI=10.1145/2590651.2590656. http://doi.acm.org/10.1145/2590651.2590656

11. Fayez Gebali. 2011. Algorithms and Parallel Computing (1st ed.). Wiley Publishing.

12. Gideon Juve and Ewa Deelman. 2011. Wrangler: virtual cluster provisioning for the cloud. In *Proceedings of the 20th international symposium on High performance distributed computing* (HPDC '11). ACM, New York, NY, USA, 277-278.DOI=10.1145/1996130.1996173 http://doi.acm.org/10.1145/1996130.199617

13. Glas, M. & P. Andres. 2010. Achieving the Cloud Computing Vision. Oracle Corporation. Disponible en: http://www.oracle.com/technetwork/topics/entarch/architectural-strategies-for-cloud--128191.pdf

14. Lavanya Ramakrishnan, R. Shane Canon, Krishna Muriki, Iwona Sakrejda, and Nicholas J. Wright. 2012. Evaluating Interconnect and Virtualization Performance forHigh Performance Computing. *SIGMETRICS Perform. Eval. Rev.* 40, 2 (October 2012), 55-60. DOI=10.1145/2381056.2381071 http://doi.acm.org/10.1145/2381056.2381071

15. Lees, K. 2012. Organizing for the Cloud. VMware Cloud Operations Global Center of Excellence. VMware, Inc. Disponible en: http://doi.acm.org/10.1145/1996130.199617

16. M. R. Palankar, A. Iamnitchi, M. Ripeanu, & S. Garnkel. Amazon S3 for science Grids: a viable solution? In International Workshop on Data-aware Distributed Computing (DADC'08) in conjunction with HPDC 2008, p. 55-64, New York, NY, USA, 2008. ACM.

17. Marcos Dias de Assuncao, Alexandre di Costanzo, and Rajkumar Buyya. 2009. Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters. In *Proceedings of the 18th ACM international symposium on High performance distributed computing* (HPDC '09). ACM, New York, NY, USA, 141-150. DOI=10.1145/1551609.1551635 http://doi.acm.org/10.1145/1551609.1551635

18. National Institute of Standards. 2013. NIST Cloud Computing Program. National Institute of Standards. U.S. Department of Commerce. Disponible en: http://www.nist.gov/itl/cloud/

19. Qiming He, Shujia Zhou, Ben Kobler, Dan Duffy, and Tom McGlynn. 2010. Case study for running HPC applications in public clouds. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing* (HPDC '10). ACM, New York, NY, USA, 395-401. DOI=10.1145/1851476.1851535 http://doi.acm.org/10.1145/1851476.1851535

20. Snir, Marc; Otto, Steve y Steven Huss-Lederman. 1995. MPI: The Complete Reference. MIT Press Cambridge, MA, USA. ISBN 0-262-69215-5

21. Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. 2001. Introduction to Algorithms(2nd ed.). McGraw-Hill Higher Education.

22. Tome, B. & M.C. Espirito-Santo. 2008. Ultra Experiment. http://hepg.sdu.edu.cn/zhangxueyao/software_doc/Geant4/html/Ultra_8cc-source.html

23. Yan Zhai, Mingliang Liu, Jidong Zhai, Xiaosong Ma, and Wenguang Chen. 2011. Cloud versus in-house cluster: evaluating Amazon cluster compute instances for running MPI applications. In *State of the Practice Reports* (SC '11). ACM, New York, NY, USA, , Article 11 , 10 pages. DOI=10.1145/2063348.2063363 http://doi.acm.org/10.1145/2063348.2063363

X. APÉNDICE

A. Configuración de Starcluster

```
####################################
## StarCluster Configuration File ##
####################################
[global]

DEFAULT_TEMPLATE=tesis

############################################
## Credenciales  ##
############################################
[aws info]
AWS_ACCESS_KEY_ID = AKIAIIQYPN2PTYM324GA
AWS_SECRET_ACCESS_KEY = [key]

# User ID de cuenta de AWS
AWS_USER_ID= 2863-7094-2886

AWS_REGION_NAME = us-west-2
AWS_REGION_HOST = ec2.us-west-2.amazonaws.com

############################
## Llaves
############################

# Llave de la computadora usada para conectarse al cluster
[key tesis]
KEY_LOCATION=~/.ssh/tesis.rsa


#################################
## Plantilla ##
#################################
DEFAULT_TEMPLATE

[cluster tesis]
# Llave definida anteriormente
KEYNAME = tesis
# No. de nodos
CLUSTER_SIZE = 15
# Usuario para ejecutar tareas (se recomienda que no sea root)
CLUSTER_USER = sgeadmin
# Shell de Linux a utilizar
CLUSTER_SHELL = bash

# Imagen creada en Amazon para la tesis en region Oregon
NODE_IMAGE_ID = ami-edf3b0dd

# Tipo de instancia (p.ej. c3.large, c3.4xlarge, m2.medium, etc)
NODE_INSTANCE_TYPE = c3.large

# Precio max. a pagar para instancias spot (en USD)
```

SPOT_BID = 0.15

B. Corrida con comunicaciones colectivas

```
//
// ********************************************************************
// * License and Disclaimer                                           *
// *                                                                  *
// * The  Geant4 software  is  copyright of the Copyright Holders  of *
// * the Geant4 Collaboration.  It is provided  under  the terms  and *
// * conditions of the Geant4 Software License,  included in the file *
// * LICENSE and available at  http://cern.ch/geant4/license .  These *
// * include a list of copyright holders.                             *
// *                                                                  *
// * Neither the authors of this software system, nor their employing *
// * institutes,nor the agencies providing financial support for this *
// * work  make  any representation or  warranty, express or implied, *
// * regarding  this  software system or assume any liability for its *
// * use.  Please see the license in the file  LICENSE  and URL above *
// * for the full disclaimer and the limitation of liability.         *
// *                                                                  *
// * This  code  implementation is the result of  the  scientific and *
// * technical work of the GEANT4 collaboration.                      *
// * By using,  copying,  modifying or  distributing the software (or *
// * any work based  on the software)  you  agree  to acknowledge its *
// * use  in  resulting  scientific  publications,  and indicate your *
// * acceptance of all terms of the Geant4 Software license.          *
// ********************************************************************
//
//
// --------------------------------------------------------------
//                      GEANT 4 - ULTRA experiment example
// --------------------------------------------------------------
//
// Code developed by:
// B. Tome, M.C. Espirito-Santo, A. Trindade, P. Rodrigues
// Modified by: Victor Araujo 2014-09
//          *****************************************************
//          *           UltraRunAction.cc
//          *****************************************************
//
//          RunAction class for Ultra; it has also a Messenger Class
//

#include "mpi.h"

#include "UltraRunAction.hh"
#include "G4Run.hh"
#include "G4RunManager.hh"
#include "Randomize.hh"
#include "UltraAnalysisManager.hh"
#include <ctime>
#include "Randomize.hh"
#include "G4SystemOfUnits.hh"
#include <sstream>

// //....oooOO0OOooo.......oooOO0OOooo.......oooOO0OOooo.......oooOO0OOooo....
```

```
UltraRunAction::UltraRunAction()
{
   seed = -1;         // RANLUX seed
   luxury = 3;        // RANLUX luxury level (3 is default)
   saveRndm = 1;
}

// //....oooOO0OOooo.......oooOO0OOooo.......oooOO0OOooo.......oooOO0OOooo....

UltraRunAction::~UltraRunAction()
{;}

// //....oooOO0OOooo.......oooOO0OOooo.......oooOO0OOooo.......oooOO0OOooo....

void UltraRunAction::BeginOfRunAction(const G4Run* aRun)
{
  G4int rank = MPI::COMM_WORLD.Get_rank();
  if (rank != 0)
            {
            G4cout << "ooo Run " << aRun->GetRunID() << " starts on slave." << G4endl;
            }
  // Get/create analysis manager: need to do that in the master and in the workers
  G4AnalysisManager* man = G4AnalysisManager::Instance();
  // Open an output file
  man->OpenFile("ultra");
  man->SetFirstHistoId(1);

  // Create histogram(s)
  man->CreateH1("1","Optical photons energy (eV)", //histoID,histo name
             500,0.,5.); //bins' number, xmin, xmax
  man->CreateH1("2","Number of Detected Photons",
              1000,0.,1000.); //bins' number, xmin, xmax

  //Master or sequential
  G4cout << "ooo Run " << aRun->GetRunID() << " starts (global)." << G4endl;
  if (seed<0) //not initialized by anybody else
            {
            seed=time(0);
            G4Random::setTheSeed(seed*(rank+1),luxury);
            G4Random::showEngineStatus();
            }

 // Persist hit energies to file
  std::stringstream ss;
  ss << rank;
  std::string str = ss.str();
  std::string base = "hitEnergies";
  std::ofstream outf(base.append(str).c_str());

  // save Rndm status
  if (saveRndm > 0)
            G4Random::saveEngineStatus("beginOfRun.rndm");

  return;
}

// //....oooOO0OOooo.......oooOO0OOooo.......oooOO0OOooo.......oooOO0OOooo....
```

```
void UltraRunAction::EndOfRunAction(const G4Run* aRun)
{
  // Write histograms to file
  // Save histograms
  // Complete clean-up
  G4int rank = MPI::COMM_WORLD.Get_rank();
  G4int size=MPI::COMM_WORLD.Get_size();
  double *hitEnergies = new double[10000];

  G4AnalysisManager* man = G4AnalysisManager::Instance();

  // Load hit energies to send to node with rank 0
  std::stringstream ss;
  ss << rank;
  std::string str = ss.str();
  std::string base = "hitEnergies";
  std::ifstream inf(base.append(str).c_str(), std::ios_base::app);

  int counter = 0;
  while (inf)
            {
            // read from the file
            std::string strInput;
            inf >> strInput;
            hitEnergies[counter] = (atof(strInput.c_str()));
              counter++;
            }

  int *counts = new int[size];
  int nelements = counter;


 // Gather used to send information about number of hits to root
  MPI_Gather(&nelements, 1, MPI::INT, counts, 1, MPI::INT, 0, MPI_COMM_WORLD);

  int *displ = new int[size];

  int recsize = 0;
  G4cout << "Ranksize rank" << rank << "\n";
  if (rank == 0) {
            double sum = 0;
            for (int i = 0; i < size; ++i) {
            displ[i] = sum;
            G4cout << rank << " : " << i << " : " << sum << "\n";
            sum += counts[i];
    G4cout << "Rank Counts  " << i << " : " << counts[i] << "\n";
            }
            recsize = sum;
            G4cout << "Ranksize to receive at rank " << sum << "\n";
  }

  double *recvbuf = new double[recsize];

 // Gatherv used to send the vectors with hit energies to the root
  MPI_Gatherv(hitEnergies, nelements, MPI::DOUBLE, recvbuf,
            counts, displ, MPI::DOUBLE, 0, MPI_COMM_WORLD);
//MPI::COMM_WORLD.Barrier();

  if (rank == 0) {
            for (int i = 0; i < recsize; ++i) {
```

```
    man->FillH1(1, recvbuf[i]/eV);
    G4cout << "Real " << recvbuf[i] << G4endl;
            }
            man->Write();
            man->CloseFile();
  }

  G4cout << "### Run " << aRun->GetRunID() << " (global) ended." << G4endl;
  // save Rndm status
  if (saveRndm == 1)
            G4Random::saveEngineStatus("endOfRun.rndm");
  return;
}
```

C. Corrida sin comunicaciones colectivas

```
//
// ********************************************************************
// * License and Disclaimer                                           *
// *                                                                  *
// * The  Geant4 software  is  copyright of the Copyright Holders  of *
// * the Geant4 Collaboration.  It is provided  under  the terms  and *
// * conditions of the Geant4 Software License,  included in the file *
// * LICENSE and available at  http://cern.ch/geant4/license .  These *
// * include a list of copyright holders.                             *
// *                                                                  *
// * Neither the authors of this software system, nor their employing *
// * institutes,nor the agencies providing financial support for this *
// * work  make  any representation or  warranty, express or implied, *
// * regarding  this  software system or assume any liability for its *
// * use.  Please see the license in the file  LICENSE  and URL above *
// * for the full disclaimer and the limitation of liability.         *
// *                                                                  *
// * This  code  implementation is the result of  the  scientific and *
// * technical work of the GEANT4 collaboration.                      *
// * By using,  copying,  modifying or  distributing the software (or *
// * any work based  on the software)  you  agree  to acknowledge its *
// * use  in  resulting  scientific  publications,  and indicate your *
// * acceptance of all terms of the Geant4 Software license.          *
// ********************************************************************
//
//
// --------------------------------------------------------------
//                      GEANT 4 - ULTRA experiment example
// --------------------------------------------------------------
//
// Code developed by:
// B. Tome, M.C. Espirito-Santo, A. Trindade, P. Rodrigues
// Modified by: Victor Araujo 2014-09
//          ***************************************************
//          *           UltraRunAction.cc
//          ***************************************************
//
//          RunAction class for Ultra; it has also a Messenger Class
//

#include "mpi.h"

#include "UltraRunAction.hh"
```

```
#include "G4Run.hh"
#include "G4RunManager.hh"
#include "Randomize.hh"
#include "UltraAnalysisManager.hh"
#include <ctime>
#include "Randomize.hh"


// //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

UltraRunAction::UltraRunAction()
{
   seed = -1;         // RANLUX seed
   luxury = 3;        // RANLUX luxury level (3 is default)
   saveRndm = 1;
}

// //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

UltraRunAction::~UltraRunAction()
{;}

// //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

void UltraRunAction::BeginOfRunAction(const G4Run* aRun)
{
  G4int rank = MPI::COMM_WORLD.Get_rank();
  if (rank != 0)
            {
            G4cout << "ooo Run " << aRun->GetRunID() << " starts on slave." << G4endl;
            }
  // Get/create analysis manager: need to do that in the master and in the workers
  G4AnalysisManager* man = G4AnalysisManager::Instance();
  // Open an output file

 // Save file numbered according to rank
  std::stringstream su;
  su << rank;
  std::string stru = su.str();
  man->OpenFile("ultra"+stru);
  man->SetFirstHistoId(1);

  // Create histogram(s)
  man->CreateH1("1","Optical photons energy (eV)", //histoID,histo name
             500,0.,5.); //bins' number, xmin, xmax
  man->CreateH1("2","Number of Detected Photons",
             1000,0.,1000.); //bins' number, xmin, xmax

  G4cout << "ooo Run " << aRun->GetRunID() << " starts (global)." << G4endl;
  if (seed<0)
            {

            seed=time(0);
            G4Random::setTheSeed(seed*(rank+1),luxury);
            G4Random::showEngineStatus();
            }

  std::stringstream ss;
  ss << rank;
```

```
  std::string str = ss.str();
  std::string base = "hitEnergies";
  std::ofstream outf(base.append(str).c_str());

  // save Rndm status
  if (saveRndm > 0)
          G4Random::saveEngineStatus("beginOfRun.rndm");

  return;
}

// //....oooOO0OOooo.....oooOO0OOooo.....oooOO0OOooo.....oooOO0OOooo....

void UltraRunAction::EndOfRunAction(const G4Run* aRun)
{
  // Write histograms to file
  // Save histograms
  // Complete clean-up

  // Simply write the hit energies histogram. No collective communications
  G4AnalysisManager* man = G4AnalysisManager::Instance();
  man->Write();
  man->CloseFile();
  delete G4AnalysisManager::Instance();
          //G4cout << "### Run " << aRun->GetRunID() << " (slave) ended." << G4endl;

  G4cout << "### Run " << aRun->GetRunID() << " (global) ended." << G4endl;
  // save Rndm status
  if (saveRndm == 1)
          G4Random::saveEngineStatus("endOfRun.rndm");
  return;
}
```

D. Hook de eventos

```
//
// ********************************************************************
// * License and Disclaimer                                           *
// *                                                                  *
// * The  Geant4 software  is  copyright of the Copyright Holders  of *
// * the Geant4 Collaboration.  It is provided  under  the terms  and *
// * conditions of the Geant4 Software License,  included in the file *
// * LICENSE and available at  http://cern.ch/geant4/license .  These *
// * include a list of copyright holders.                             *
// *                                                                  *
// * Neither the authors of this software system, nor their employing *
// * institutes,nor the agencies providing financial support for this *
// * work  make  any representation or  warranty, express or implied, *
// * regarding  this  software system or assume any liability for its *
// * use.  Please see the license in the file  LICENSE  and URL above *
// * for the full disclaimer and the limitation of liability.         *
// *                                                                  *
// * This  code  implementation is the result of  the  scientific and *
// * technical work of the GEANT4 collaboration.                      *
// * By using,  copying,  modifying or  distributing the software (or *
// * any work based  on the software)  you  agree  to acknowledge its *
// * use  in  resulting  scientific  publications,  and indicate your *
// * acceptance of all terms of the Geant4 Software license.          *
```

```
// ********************************************************************
//
//
// ------------------------------------------------------------
//                      GEANT 4 - ULTRA experiment example
// ------------------------------------------------------------
//
// Code developed by:
// B. Tome, M.C. Espirito-Santo, A. Trindade, P. Rodrigues
// Modified by: Victor Araujo 2014-09
//          ***************************************************
//          *           UltraEventAction.cc
//          ***************************************************
//
//          Ultra EventAction class. The UltraAnalysisManager class is used for histogram
//          filling
//
#include "mpi.h"

#include "UltraEventAction.hh"
#include "UltraPrimaryGeneratorAction.hh"
#include "UltraOpticalHit.hh"

#include "G4RunManager.hh"
#include "G4Event.hh"
#include "G4EventManager.hh"
#include "G4SDManager.hh"
#include "G4HCofThisEvent.hh"
#include "G4VHitsCollection.hh"
#include "G4GeneralParticleSource.hh"
#include "UltraAnalysisManager.hh"
#include "G4SystemOfUnits.hh"
#include <fstream>
#include <iostream>
#include <sstream>

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

UltraEventAction::UltraEventAction()
  :OpticalHitsCollID(-1)
{;}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

UltraEventAction::~UltraEventAction(){;}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

void UltraEventAction::BeginOfEventAction(const G4Event* evt)
{
  G4int printModulo = 100;

  evtNb = evt->GetEventID();

  G4SDManager * SDman = G4SDManager::GetSDMpointer();


  if(OpticalHitsCollID==-1) {
          OpticalHitsCollID = SDman->GetCollectionID("OpticalHitsCollection");
```

```
  }


  if (evtNb%printModulo == 0)
        G4cout << "\n---> Begin of Event: " << evtNb << G4endl;

}

//....oooOO0OOooo.....oooOO0OOooo.....oooOO0OOooo.....oooOO0OOooo....



void UltraEventAction::EndOfEventAction(const G4Event* evt)
{
  //int size=MPI::COMM_WORLD.Get_size();
  int rank=MPI::COMM_WORLD.Get_rank();
  G4HCofThisEvent* HCE = evt->GetHCofThisEvent();
  UltraOpticalHitsCollection* OpticalHitsColl = 0;


  // Fill histograms
  G4AnalysisManager* man = G4AnalysisManager::Instance();


  if(HCE){
        if(OpticalHitsCollID != -1) OpticalHitsColl =
        (UltraOpticalHitsCollection*)(HCE->GetHC(OpticalHitsCollID));
  }


  G4int nOptHits = 0 ;


  if(OpticalHitsColl) {
        nOptHits = OpticalHitsColl->entries();


#ifdef ULTRA_VERBOSE
        if (nOptHits > 0){
        G4cout << " Optical Hit # " << " " << "Energy (eV)" <<  " " << "x,y,z (cm)" << G4endl ;
        }
#endif


        // Hit energies are stored in a file to be sent to root or written to histogram after end of run
        double *hitEnergies = new double[nOptHits];
std::stringstream ss;
ss << rank;
std::string str = ss.str();
std::string base = "hitEnergies";
        std::ofstream outf(base.append(str).c_str(), std::ios_base::app);
        for(G4int iHit=0; iHit<nOptHits; iHit++) {
        G4double HitEnergy = (*OpticalHitsColl)[iHit]->GetEnergy() ;
        G4cout << "Rank hitenergy  " << iHit << " : " << ((double)HitEnergy) << "\n";
        outf << HitEnergy << G4endl;
        hitEnergies[iHit] = ((double)HitEnergy);
        }
```

```
 }
  man->FillH1(2,nOptHits);

}
```

E. Calendarización estática

```
//
// ********************************************************************
// * License and Disclaimer                                           *
// *                                                                  *
// * The  Geant4 software  is  copyright of the Copyright Holders  of *
// * the Geant4 Collaboration.  It is provided  under  the terms  and *
// * conditions of the Geant4 Software License,  included in the file *
// * LICENSE and available at  http://cern.ch/geant4/license .  These *
// * include a list of copyright holders.                             *
// *                                                                  *
// * Neither the authors of this software system, nor their employing *
// * institutes,nor the agencies providing financial support for this *
// * work  make  any representation or  warranty, express or implied, *
// * regarding  this  software system or assume any liability for its *
// * use.  Please see the license in the file  LICENSE  and URL above *
// * for the full disclaimer and the limitation of liability.         *
// *                                                                  *
// * This  code  implementation is the result of  the  scientific and *
// * technical work of the GEANT4 collaboration.                      *
// * By using,  copying,  modifying or  distributing the software (or *
// * any work based  on the software)  you  agree  to acknowledge its *
// * use  in  resulting  scientific  publications,  and indicate your *
// * acceptance of all terms of the Geant4 Software license.          *
// ********************************************************************
//
//
// --------------------------------------------------------------
//                      GEANT 4 - ULTRA experiment example
// --------------------------------------------------------------
//
// Code developed by:
// B. Tome, M.C. Espirito-Santo, A. Trindade, P. Rodrigues
//
//          ******************************************************
//          *           UltraPrimaryGeneratorAction.cc
//          ******************************************************
//
//          Class used in the definition of the optical photons source
//          A plane, circular source is used. Depending on the source position, optical
//          photons may reach the UVscope directly or after reflection. By default direct
//          incidence is used. The source parameters can be set directly in this class
//          or through the GeneralParticleSource  messenger class.
//
#include "mpi.h"
#include "UltraPrimaryGeneratorAction.hh"
#include "UltraDetectorConstruction.hh"

#include "G4PhysicalConstants.hh"
#include "G4SystemOfUnits.hh"
#include "G4RunManager.hh"
#include "G4Event.hh"
#include "G4GeneralParticleSource.hh"
#include "G4SPSAngDistribution.hh"
```

```
#include "G4SPSEneDistribution.hh"
#include "G4SPSPosDistribution.hh"
#include "G4ParticleTable.hh"
#include "G4ParticleDefinition.hh"
#include "G4ThreeVector.hh"

//....oooOO0OOooo.....oooOO0OOooo.....oooOO0OOooo.....oooOO0OOooo....

UltraPrimaryGeneratorAction::UltraPrimaryGeneratorAction()
{

  particleGun = new G4GeneralParticleSource();


  // Define here the user default properties for the General Particle Source (GPS)
  // Can be modified through the GPS Messenger (/gps/... commands)

  G4ParticleTable* particleTable = G4ParticleTable::GetParticleTable();
  G4String particleName;

  G4ParticleDefinition* opticalphoton = particleTable->FindParticle(particleName="opticalphoton");

  //....PARTICLE DEFINITIONS
  particleGun->SetParticleDefinition(opticalphoton);

  G4ThreeVector Polarization = G4ThreeVector(1.,1.,0.) ;
  particleGun->SetParticlePolarization(Polarization);

  // DEFINE A MONO-ENERGETIC SOURCE
  G4SPSEneDistribution *eneDist = particleGun->GetCurrentSource()->GetEneDist() ;
  eneDist->SetEnergyDisType("Mono");
  eneDist->SetMonoEnergy(3.0*eV);

  // SET POSITION DISTRIBUTION
  G4SPSPosDistribution *posDist = particleGun->GetCurrentSource()->GetPosDist() ;
  posDist->SetPosDisType("Plane");
  posDist->SetPosDisShape("Circle");
  posDist->SetRadius(20.0*cm);

#ifdef ULTRA_MIRROR_USE
#define ULTRA_REFLECTION_USE
#endif

#ifdef ULTRA_GROUND_USE
#define ULTRA_REFLECTION_USE
#endif

  G4SPSAngDistribution *angDist = particleGun->GetCurrentSource()->GetAngDist() ;

#ifdef ULTRA_REFLECTION_USE
  angDist->SetParticleMomentumDirection(G4ThreeVector(0.0,-1.0,0.0)) ;
  posDist->SetPosRot1(G4ThreeVector(1.,0.,0.));
  posDist->SetPosRot2(G4ThreeVector(0.,0.,-1.));
  posDist->SetCentreCoords(G4ThreeVector(0.0*cm,90.0*cm,150.0*cm));

#else
  angDist->SetParticleMomentumDirection(G4ThreeVector(0.0,0.0,-1.0)) ;
  posDist->SetCentreCoords(G4ThreeVector(0.0*cm,0.0*cm,150.0*cm));
```

```
#endif

}

//....oooOO0OOooo.....oooOO0OOooo.....oooOO0OOooo.....oooOO0OOooo....

UltraPrimaryGeneratorAction::~UltraPrimaryGeneratorAction()
{
  delete particleGun;
}

//....oooOO0OOooo.....oooOO0OOooo.....oooOO0OOooo.....oooOO0OOooo....

void UltraPrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
{

// If the ID of the event mod the total number of nodes is different from the rank the event must be skipped
  if (((anEvent->GetEventID() % MPI::COMM_WORLD.Get_size())) != MPI::COMM_WORLD.Get_rank()) {
            G4cout << "Rank " << MPI::COMM_WORLD.Get_rank() << " skipping event ID " <<  anEvent-
>GetEventID() << "\n";
            return;
  }

  G4int iEvent = anEvent->GetEventID() ;
  if ( iEvent == 0 ){

            G4cout << particleGun->GetParticleDefinition()->GetParticleName()       << " " ;
            G4cout << particleGun->GetCurrentSource()->GetEneDist()->GetEnergyDisType() << " " ;
            G4cout << particleGun->GetCurrentSource()->GetPosDist()->GetPosDisType()    << G4endl ;


// Check if optical photon wavelength is within limits set for material optical properties tables.




}
  particleGun->GeneratePrimaryVertex(anEvent);

            if (particleGun->GetParticleDefinition()->GetParticleName() == "opticalphoton"){

                        const UltraDetectorConstruction * detector =
                        dynamic_cast<const UltraDetectorConstruction *>((G4RunManager::GetRunManager())-
>GetUserDetectorConstruction()) ;

    G4double lambda_min = detector->GetLambdaMin() ;
    G4double lambda_max = detector->GetLambdaMax() ;

                        G4double energy = particleGun->GetParticleEnergy() ;

    if (h_Planck*c_light/energy > lambda_max || h_Planck*c_light/energy < lambda_min){
            G4cerr << "Error ! Optical photon energy (" << energy/eV << " eV) out of limits set by material
optical properties tables. \n"
            << "Please check that photon wavelength is within the following interval: ["
            << lambda_min/nm << ","
            << lambda_max/nm << "] nm"
            << ", i.e., ["
```

```
            << h_Planck*c_light/lambda_max/eV << ","
            << h_Planck*c_light/lambda_min/eV << "] eV"
            << G4endl ;

            G4Exception("UltraPrimaryGeneratorAction::GeneratePrimaries()","AirSh005",
                                FatalException,"Wavelength outside the valid range") ;
    }
  }

}
```

F. Versiones de software utilizado

Geant4 10.0 patch-2
Cmake 2.8
Starcluster 0.95.5
OpenMPI 1.8.3
GCC 4.6