# Universidad de las Ciencias Informáticas

## Facultad 6


Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas.

**Título:** Biblioteca para el pre-procesamiento paralelo del modelo BDAM usando un *cluster* de computadoras.

**Autor:** Yurian Jimenez Rubalcaba

**Tutores:** MSc. Gilberto Arias Naranjo

**Co-tutores:** José Luis Castrillón Garrido

Nelson González Peñate

La Habana, 5 de Julio del 2016

¨Cualquier cosa que la mente pueda concebir, puede ser lograda¨.

W.Clement Stone

## Declaración de Autoría

Declaro ser autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos primordiales de la misma con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Yurian Jimenez Rubalcaba

_____

Firma del Autor

MSc. Gilberto Arias Naranjo

_____

Firma del Tutor

Ing. José Luis Castrillón Garrido

_____

Firma del Tutor

Ing. Nelson González Peñate

_____

Firma del Tutor

## Datos de contacto

**Autor:**

Yurian Jimenez Rubalcaba

**E-mail:** yjimenez@estudiantes.uci.cu

**Tutores:**

Msc. Gilberto Arias Naranjo

Categoría Docente: Asistente

**E-mail:** gilbertoa@uci.cu

Ing. José Luis Castrillón Garrido

**E-mail:** jlcastrillon@uci.cu

Ing. Nelson González Peñate

**E-mail:** npenate@uci.cu

## Dedicatoria

*A la memoria de mis abuelas y abuelos por apoyarme de alguna forma u otra en mi vida.*

*A mi familia y en especial a mis padres por ser mi motor impulsor, los quiero.*

## Agradecimientos

*A mis tutores Gilberto, José Luis y Nelson por guiarme en todo momento, por su apoyo, recomendaciones y esfuerzo para lograr que los resultados fueran los mejores.*

*A mis padres por todos los sacrificios que han hecho por mí en estos cinco años de carrera en beneficio de mi formación como profesional.*

*A mi hermano por creer en mí y darme su apoyo en todo momento, aunque se molestaba a veces con mi mamá por ser tan consentida conmigo.*

*A mi familia por estar pendientes de mí y los resultados alcanzados a lo largo de esta batalla de cinco años.*

*A mis amistades de Jovellanos por darme su apoyo incondicional desde el mismo comienzo de la carrera, los quiero mucho.*

*A todos los amigos conseguidos en estos cinco años, por haber sido como una familia muy grande.*

*A Luis Antonio García González por brindarme los conocimientos necesarios para montar un cluster de computadoras.*

*Al Movimiento de Programación Competitiva "Tomás López Jiménez" de la UCI.*

*A todos los profesores que me ayudaron durante el desarrollo de la tesis.*

*A Yunier René Pérez Valdés por comportarse como profesor, amigo, familia y brindar consejos en todo momento.*

*A todos los que de una forma u otra han contribuido a mi formación como profesional y humano en general.*

## Resumen

La representación de terrenos en tres dimensiones es de gran importancia en muchas áreas de la investigación científica. Un ejemplo es el análisis de patrones espaciales con el fin de conseguir predicciones a partir de los datos que se visualizan, simulación de fenómenos naturales o mostrar las diferentes propiedades de un terreno. Cuando la dimensión del terreno o mapa es grande se necesitan modelos computacionales que permitan un procesamiento eficiente de los datos. En la Universidad de las Ciencias Informáticas se desarrolla un componente para la visualización de varias capas de información geográfica en tres dimensiones. Entre las técnicas utilizadas por el componente con el objetivo de lograr la representación de terrenos de altas dimensiones se encuentra el modelo BDAM. Una de las desventajas de este modelo es que tiene un alto costo computacional en el pre-procesamiento de los datos.

Una de las variantes para aumentar el rendimiento en el pre-procesamiento es mediante paralelismo de memoria compartida. Sin embargo, surgen problemas en cuanto a uso de memoria y procesamiento. Una alternativa para disminuir los requerimientos, en memoria y tiempo de procesamiento es con la computación distribuida o la computación paralela en ambientes de tipo *cluster*.

En la presente investigación se implementa una biblioteca para el procesamiento paralelo de datos sobre un *Cluster Beowulf*. Los resultados obtenidos evidencian una reducción significativa del tiempo de ejecución en el pre-procesamiento de datos.

**Palabras claves:** *Cluster,* Computación de alto rendimiento, Procesamiento distribuido, Procesamiento paralelo.

## Abstract

The representation of terrain in three dimensions is of great importance in many areas of scientific research. One example is the analysis of spatial patterns in order to obtain predictions from the data displayed, simulating natural phenomena or show different properties of a field. When the size of the field or map is large computational models that allow efficient processing of data are needed. At the University of Information Science a component for viewing multiple layers of three-dimensional geographic information develops. Among the techniques used by the component with the aim of achieving the representation of high-dimensional terrain is the DBSA model. One disadvantage of this model is that it has a high computational cost in the pre-processing of data.

One of the variants to increase performance in the pre-processing is through shared memory parallelism. However, problems arise in terms of memory usage and processing. An alternative to reduce the requirements, memory and processing time is with distributed computing and parallel computing cluster type environments.

In this research a library for parallel data processing on a Beowulf Cluster is implemented. The results show a significant reduction in execution time in the pre-processing of data.

**Keywords:** Cluster, Distributed processing, High Performance Computing, Parallel processing.

# Índice

# Índice de Figuras

# Índice de Tablas

# Introducción

La visualización interactiva y eficiente de terrenos es de gran utilidad en varias áreas, como es el caso de los Sistemas de Información Geográfica (GIS por sus siglas en inglés), la realidad virtual y la visualización científica. Con el creciente avance en las tecnologías satelitales se ha hecho posible la existencia de terrenos e imágenes con elevadas resoluciones y dimensiones, lo que en muchas ocasiones conlleva a un elevado costo de almacenamiento que imposibilita su visualización directa. Una de las alternativas existente para dar solución a este problema es mediante el uso de modelos multi-resolución (Pajarola and Gobbetti, 2007), estos consisten en el filtrado de los datos que en un instante dado son relevantes para la visualización en dependencia de los parámetros de visión y de métricas de error. El costo computacional para la obtención de estos modelos es elevado cuando los terrenos están compuestos por grandes volúmenes de datos (De Floriani et al., 2005).

En este sentido, en los últimos años y con la finalidad de incrementar el rendimiento en velocidad de las aplicaciones, los procesadores han migrado hacia el paralelismo como un cambio importante en busca de este factor (Fraire et al., 2013). Por su parte, los paradigmas de programación paralela están estrechamente relacionados por la arquitectura de hardware y la infraestructura de software con la que se cuenta. Esta necesidad ha provocado que se haya potenciado la investigación y el desarrollo de nuevos métodos y estrategias para poder ejecutar aplicaciones con altos requerimientos de procesamiento. Una de las alternativas para resolver este problema es la utilización de supercomputadores, los cuales brindan un desempeño elevado; pero estos son caros y por tal motivo no abundan en organizaciones o proyectos pequeños.

Con el aumento de las prestaciones de las computadoras de trabajo, o *workstations,* surgió el interés de aprovechar estos recursos en conjunto, sirviendo esto como una alternativa más económica a los llamados supercomputadores o *mainframes*. Con esto, surgió una nueva arquitectura denominada *Cluster*, que permite conectar mediante un enlace de red diferentes nodos, y aprovechar la capacidad de cómputo del conjunto. En la actualidad existe una gran cantidad de recursos computacionales, repartidos en diferentes departamentos de una misma organización, empresa o universidad. Esta disponibilidad brinda la posibilidad a los investigadores de estudiar y desarrollar aplicaciones paralelas para aprovechar estos recursos al máximo.

En el año 2013 se inscribe el proyecto de investigación: “**Representación multicapas de información geográfica, tomando como base Modelos Digitales de Elevación, para el desarrollo y análisis de**

**simulaciones en 3D.**" suscrito al Centro de Matemática y Computación (CEMC) y que se realiza en conjunto con el Centro Internacional de Métodos Numéricos en la Ingeniería (CIMNE), con sede en Barcelona, España. En este proyecto se desarrolló un componente para la visualización en tres dimensiones de información geográfica (Visor de Terrenos). El objetivo fundamental del proyecto es la representación de forma eficiente, de varias capas de información geográfica, incluyendo aquellas que formen parte de resultados de ejecutar simulaciones sobre determinado terreno.

Actualmente en el Visor de Terrenos se utiliza el modelo BDAM (Cignoni et al., 2003) para la representación multi-resolución de las capas de información geográfica. En una primera etapa, el modelo BDAM realiza un pre-procesamiento intensivo de los datos, mediante la aplicación de algoritmos de elevado costo computacional. La forma en que el modelo divide los datos para el pre-procesamiento permite el uso de técnicas de programación paralela, sin embargo, en el Visor de Terrenos solamente se aprovechan los procesadores del ordenador donde se esté ejecutando el componente, pudiéndose aumentar el rendimiento del modelo usando un *cluster* con mayor cantidad de procesadores. Actualmente el Visor de Terrenos no posee ningún mecanismo para la ejecución de las tareas de pre-procesamiento de datos en un *cluster* de computadoras.

Partiendo de lo antes expuesto se define como **problema de investigación:**

¿Cómo disminuir el tiempo de procesamiento de datos del modelo BDAM en el Visor de Terrenos utilizando un *cluster* de computadoras?

Del problema anterior se define como **objeto de estudio:** Procesamiento paralelo de datos en sistemas distribuidos.

Se establece como **campo de acción:** Técnicas de computación paralela aplicadas al procesamiento de datos en arquitecturas de cómputo heterogéneas y dedicadas.

El **objetivo general** de la presente investigación es: Desarrollar una biblioteca para el procesamiento paralelo del modelo BDAM sobre un *cluster* de computadoras para disminuir el tiempo de respuesta en el procesamiento de datos.

**Objetivos específicos**

- Implementar un mecanismo de comunicación con un *cluster* de computadoras.
- Implementar la aplicación que realiza el procesamiento paralelo de los datos sobre un *cluster*.

- Diseñar las funcionalidades de la biblioteca.
- Implementar las funcionalidades de la biblioteca.
- Validar el funcionamiento de la biblioteca a través de pruebas.

A partir del marco teórico desarrollado se formula la siguiente **hipótesis:**

Si se utiliza un *cluster* de ordenadores para el pre-procesamiento del modelo BDAM aumentará el rendimiento del mismo.

Para dar cumplimiento al objetivo planteado se definieron las siguientes **tareas de investigación**.

- Estudio de las tecnologías para el procesamiento paralelo de tareas en un *cluster*.
- Estudio de las diferentes herramientas para la gestión de recursos en *clusters*.
- Estudio de varios protocolos de comunicación utilizados en arquitecturas *clusters*.
- Selección de las tecnologías a utilizar.
- Realización de experimentos para comparar el rendimiento del modelo ejecutando las tareas en el *cluster*.
- Evaluación de los resultados obtenidos en los experimentos.

Durante el desarrollo de la investigación fueron utilizados los siguientes **métodos de investigación** científica:

- Método analítico-sintético: Se estudiaron los diferentes sistemas distribuidos y después fueron sintetizados en la selección de un *cluster Beowulf*.
- Método hipotético-deductivo: Para la formulación de la hipótesis y arribar a conclusiones.
- Método de modelado: para representar los diagramas correspondientes a las etapas de análisis, diseño e implementación del servicio.
- Análisis documental: Se realizó una revisión de la literatura sobre Sistemas distribuidos para consultar la información necesaria en el proceso de investigación.

**Resultados esperados**

- Incluir al Visor de Terrenos la funcionalidad de ejecutar tareas en un *cluster.*
- Disminuir el tiempo de procesamiento de los datos en el Visor de Terrenos.

El presente trabajo consta de la estructura siguiente:

**Capítulo 1:** Fundamentación teórica.

En este capítulo se describen las principales características del modelo BDAM y el proceso de visualización en el Visor de Terrenos, así como las limitaciones del procesamiento secuencial de las capas de información geográfica a través de este modelo. Se identifican los principales elementos de computación paralela de interés para la investigación y un análisis de las capacidades de cómputo de los *clusters*.

**Capítulo 2:** Análisis y Diseño.

En este capítulo se encuentran los elementos resultantes del análisis y diseño de la solución. Listado de requisitos funcionales y no funcionales. Se muestra la arquitectura de la solución a través de un diagrama de componentes. Se muestra el diagrama de clases y los patrones de diseño empleados. Por último, se propone el proceso para realizar el procesamiento paralelo de los datos del modelo BDAM.

**Capítulo 3:** Implementación y pruebas.

En este capítulo se describen la implementación de la solución, proporcionando detalles de la misma y se especifica los estándares de codificación utilizados. Se le realizan pruebas a la solución para validar su correcto funcionamiento. Se muestra además los experimentos realizados para validar el aumento del rendimiento del Visor de Terrenos en el procesamiento de datos.

# Capítulo 1: Fundamentación teórica

En este capítulo se abordan los conceptos fundamentales para el domino del problema, los cuales contribuyen a la compresión de su contenido. Se caracteriza los principales elementos de computación paralela de interés para el desarrollo de la investigación, así como las tecnologías a utilizar en el desarrollo de la solución.

## 1.1 Conceptos asociados al dominio del problema

### *1.1.1 Modelo Digital de Terreno y Modelo Digital de Elevación*

Un modelo digital de terreno (DTM, por sus siglas en inglés) es un conjunto ordenado de puntos que representa la distribución espacial de varios tipos de información en el terreno. Puede definirse también como la representación digital de la distribución espacial de información de terreno representada por localizaciones en dos dimensiones junto a una representación matemática de la información del terreno. Un modelo digital de elevación (MDE) es un subconjunto de DTM y se refiere a los datos de elevación organizados en forma de matriz (Li et al., 2004), a este modelo se le conoce también como *raster*.

### *1.1.2 Proceso de visualización en el Visor de Terrenos*

En el Visor de Terrenos cada capa es representada mediante un MDE y para llevar a cabo las animaciones se realiza una superposición de las capas que conforman la animación. Cada capa para ser visualizada en el Visor de Terrenos debe pasar por un proceso que consta de cuatro fases:

1. Carga de los metadatos del modelo.
2. Aplicación de técnicas de optimización y generación de la geometría.
3. Obtención de la información que se encuentra dentro del campo de visión de la cámara.
4. Representación de la escena.

En la segunda fase es donde se realiza el pre-procesamiento de los datos usando el modelo BDAM. El pre-procesamiento de varias capas se realiza de forma secuencial, o sea una a la vez, siendo independiente el procesamiento de cada una de ellas. Las tareas que tienen lugar en la construcción del modelo BDAM se ejecutan de manera independiente, lo que permite la paralelización de las mismas (Cignoni et al., 2003). Sin embargo, la implementación de este modelo en el Visor de Terrenos explota esta última característica haciendo uso solamente de los núcleos del microprocesador de la estación de trabajo donde se esté ejecutando. Una de las principales desventajas en la utilización de este modelo es que tiene un alto costo computacional durante el pre-procesamiento de los datos.

### *1.1.3 Arquitecturas paralelas*

A partir del interés de resolver problemas que exigen un alto costo computacional para su resolución, surge el concepto de paralelismo, que como plantea (Gove, 2010), su esencia consiste en usar múltiples recursos computacionales simultáneamente para resolver un problema dado, con el fin de lograr menor tiempo de procesamiento. Existen múltiples tipos de computadoras paralelas, cada una con modelos de programación asociados a su jerarquía de memoria y componentes físicos. Si bien la mayoría de las computadoras modernas permiten programar con herramientas casi independientes de la plataforma, es conveniente conocer la arquitectura del hardware paralelo con el fin de hacer un uso óptimo de los recursos disponibles. En función de los conjuntos de instrucciones y datos, frente a la arquitectura secuencial que denominaríamos SISD (Single Instruction, Single Data), podemos clasificar las diferentes arquitecturas paralelas en diferentes grupos (a esta clasificación se le suele llamar taxonomía de Flynn) (McBryan, 1994) (Almasi and Gottlieb, 1988) (SPARSKIT, 1994) (Solar et al., 2011) (Van der Steen and Dongarra, 1995):

- **SIMD (*Single Instruction, Multiple Data*):** En esta arquitectura unos procesos homogéneos (con el mismo código) ejecutan sincrónicamente la misma instrucción sobre sus datos, o bien la misma operación se aplica sobre unos vectores de tamaño fijo o variable (Márquez Pérez et al., 2011). Por ejemplo: GPU's[1].
- **MISD (*Multiple Instruction, Single Data*):** El mismo conjunto de datos es tratado de forma diferente por los procesadores. Son útiles en caso donde sobre el mismo conjunto de datos se deban realizar muchas operaciones diferentes (Márquez Pérez et al., 2011).
- **MIMD (*Multiple Instruction, Multiple Data*):** En este enfoque no sólo distribuimos datos sino también las tareas a realizar entre los procesadores. Varios flujos (posiblemente diferentes) de ejecución son aplicados a diferentes conjuntos de datos (Márquez Pérez et al., 2011). Estos sistemas paralelos se pueden construir a partir de componentes SIMD, como es el caso de muchos *clusters* de computadoras.
- **SPMD (*Single Program, Multiple Data*):** Un SPMD es un MIMD con algunas variaciones y es el más usado para desarrollar aplicaciones paralelas en *clusters* de computadoras. En este modelo el mismo código del programa de la aplicación es ejecutado en todos los procesadores. Los datos de la aplicación son divididos y distribuidos en los procesadores donde fue ejecutado el programa.

[1] *Graphical Processing Unit*. Usualmente conocidos como "tarjetas de video" pero en la actualidad cuentan con ambientes de desarrollo para propósito general. Su valor radica en la rapidez de su memora y componentes internos, y la gran cantidad de núcleos de CPU SIMD de bajo consumo eléctrico, apto para aplicaciones numéricas que puedan correr con gran cantidad de hilos.

Así, cada procesador ejecuta básicamente la misma pieza de código, pero sobre una parte diferente de los datos (Márquez Pérez et al., 2011).

Las computadoras paralelas MIMD pueden clasificarse en algunas de las siguientes categorías (Dongarra et al., 2003), como muestra la Figura 1:

a) **Memoria compartida:** Conocido típicamente como Multiprocesamiento Simétrico (SMP, por sus siglas en inglés), cada procesador posee su propia memoria *cache*, y está interconectado junto con otros procesadores a la memoria principal mediante un medio físico (por ejemplo, un *bus*), Figura 1(a). Los programas para estas máquinas deben controlar las operaciones como lectura y escritura de modo que estas no produzcan inconsistencias. Un inconveniente del SMP es la cantidad limitada de procesadores y el alto costo en conectividad interna.
b) **Memoria distribuida (no compartida):** Resuelve algunos de los problemas de crecimiento de las máquinas SMP pues utiliza procesadores con memoria local interconectados mediante una red, Figura 1(b). El acceso a datos locales es rápido, sin embargo, no es posible leer la memoria de otros nodos de la red directamente, por lo que se debe usar algún mecanismo de paso de mensajes. Esto consume ancho de banda y produce retrasos en el procesamiento, por lo que una buena práctica es enviar pocos mensajes largos.
c) **Sistemas híbridos:** Los dos enfoques anteriores pueden ser combinados de diferentes formas. Algunas máquinas de memoria distribuida (memoria-compartida distribuida, DSM) permiten que un procesador lea un dato remoto de forma directa. La latencia aumenta con la distancia entre las máquinas, y el mantenimiento de la coherencia de *cache* usualmente involucra redes sofisticadas. Para sistemas de paralelismo masivo, uno de los enfoques híbrido más populares son los *clusters* SMP, donde se cuenta con un sistema de memoria distribuida donde cada nodo es una máquina SMP.

De manera general, los procesos de una aplicación paralela requieren compartir los datos entre ellos, por lo que deben comunicarse y sincronizarse para acceder a los datos. Para realizar la comunicación entre los procesos de un *cluster* comúnmente se usa el ambiente de programación de paso de mensajes. Este ambiente de programación está basado en una biblioteca que se ejecuta en el espacio de dirección del proceso. Además, la biblioteca provee un conjunto de funciones o primitivas que permiten al programador crear aplicaciones paralelas.

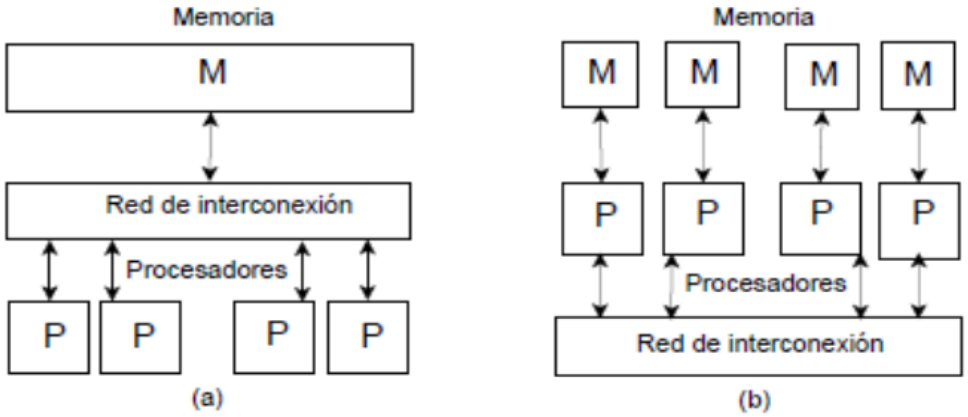


Fig 1: Clasificaciones de las computadoras paralelas MIMD. a) Arquitectura de memoria compartida. b) Arquitectura de memoria distribuida.

### *1.1.4 Estrategias de paralelización*

Se suelen aplicar dos estrategias básicas para dividir la solución a un problema, de forma que este pueda ser resuelto en paralelo. Una estrategia se enfoca en tareas (*task partitioning)* y otras en datos (*data partitioning*) (Dongarra et al., 2003).

Si se identifican las fases o tareas dentro del programa y las dependencias entre ellas, aquellas que no son interdependientes pueden ser procesadas en paralelo. De esta forma, diferentes procesadores ejecutan diferentes funciones. Este enfoque se conoce como paralelización por tareas.

Por otra parte, el dominio del problema puede ser dividido en subdominios o regiones mapeadas a diferentes procesos o procesadores. Este acercamiento puede favorecer la escalabilidad del programa, pues permite controlar directamente la relación entre el tamaño del problema y la cantidad de procesadores. Esta estrategia, llamada paralelización por datos, es usada ampliamente ya que mantiene más procesadores activos.

Para contribuir al diseño de la solución de la presente investigación se seleccionó la estrategia de paralelización por datos o descomposición del dominio del problema, debido a sus ventajas y las características expuestas anteriormente del proceso de visualización en el Visor de Terrenos.

### *1.1.5 Cluster de cómputo*

Un sistema distribuido consiste en grupos de ordenadores conectados a través de una red, y la computación distribuida se refiere a la forma en que una aplicación es ejecutada en más de un ordenador, de forma simultánea. Se pueden utilizar los siguientes criterios para distinguir un sistema paralelo de uno distribuido:

- En un sistema paralelo, todos los procesadores tienen acceso a una memoria compartida. Esta memoria compartida puede ser utilizada para intercambiar información entre procesadores.
- En un sistema distribuido, cada procesador tiene su propia memoria privada. La información es intercambiada entre procesadores mediante el uso de mensajes.

Se entiende por sistemas de cómputo distribuido aquellos en que sus componentes, situados en computadores en red, se comunican y coordinan únicamente a través del paso de mensaje (Blanco de Frutos, 2012). Dentro de las categorías de sistemas distribuidos se puede encontrar un amplio abanico de sistemas diferentes, entre ellos se encuentra la arquitectura *cluster*. Se define un *cluster* como una colección de recursos de computación formada normalmente por ordenadores comunes de escritorio, conectados entre sí y que se comportan como una única computadora, alcanzando en ocasiones, y según su configuración, rendimientos comparables al de los súper-computadores (Carri, 2004).

Si se quisiera clasificar a los *clusters* de cómputo según algún “Modelo de Arquitectura Paralela”, estos son sistemas MIMD con memoria distribuida, aunque la interconexión puede no ser dedicada, y utilizar mecanismos de interconexión simples de red local, o incluso a veces no dedicados exclusivamente al conjunto de máquinas. Básicamente es un modelo DM-MIMD (*Distributed Memory*-MIMD) en donde las comunicaciones entre procesadores son habitualmente más lentas (Márquez Pérez et al., 2011).

Una ventaja importante que brindan los *clusters* es que son bastante escalables en cuanto al hardware. Podríamos decir que es fácil construir sistemas con centenares o miles de máquinas, donde comúnmente las tecnologías de red son las que limitan la capacidad de escalabilidad. Aunque no es habitual disponer de hardware tolerante a fallos, normalmente se incorporan mecanismos por software que invalidan (y/o substituyen por otro) el recurso que ha fallado.

En la Figura 2 se muestra un diagrama que representa la distribución básica de un sistema *cluster*. En ella se puede observar los distintos computadores, encargados de llevar a cabo las tareas de cómputo (*Node* 1, *Node* 2,…, *Node n*) y un servidor encargado de controlar y ejecutar aplicaciones en los nodos conectados a través de una red de comunicaciones que puede ser dedicada. A continuación, se describe un *cluster* que

cumple con las características mencionadas anteriormente, el cual se utilizará para el desarrollo de la solución.

***Cluster Beowulf*:** Es un sistema de cómputo masivamente paralelo de altas prestaciones donde cada nodo es una computadora personal y están conectados a través de redes informáticas *estándares* (Sterling, 2002), sin el uso de equipos desarrollados específicamente para la computación paralela. El sistema consiste básicamente en un servidor y uno o más clientes conectados generalmente a través de Ethernet y sin la utilización de ningún hardware específico. Para explotar esta capacidad de cómputo, es necesario que los programadores tengan un modelo de programación distribuido lo cual se puede conseguir a través de software diseñados para este fin. La capa de software aportada por sistemas tales como *Parallel Virtual Machine* (PVM) y *Message Passing Interface* (MPI) facilita notablemente la abstracción del sistema y permite desarrollar aplicaciones paralelas/distribuidas de modo sencillo y simple; los mismos son ampliamente utilizados en este tipo de *cluster*.

Hay que considerar que *Beowulf* no es un software que transforma el código del usuario en distribuido ni afecta al *kernel*[2] del sistema operativo. Simplemente es una forma de agrupación de computadoras que ejecutan GNU/Linux y actúan como un supercomputador; por ejemplo: la construcción de un *cluster Beowulf* de dos nodos, se puede llevar a cabo simplemente con dos computadoras conectadas por Ethernet mediante un *switch,* una distribución de GNU/Linux estándar (como Debian), el sistema de archivo compartido (NFS, por sus siglas en inglés) y tener habilitados los servicios de red como SSH. La principal ventaja de un *cluster Beowulf* es su bajo costo monetario e implementación en una determinada organización, empresa, universidad o institución con determinados recursos computacionales ociosos.

[2] El *kernel* es el módulo central de un sistema operativo y es el primero en cargarse en memoria. Típicamente el *kernel* es el responsable de la gestión de la memoria, procesos, administración de tareas entre otras responsabilidades imprescindibles para la ejecución de aplicaciones.

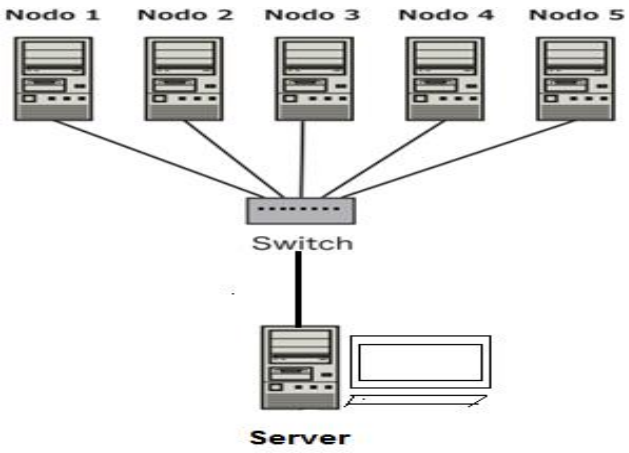


Fig 2. Distribución básica de un sistema cluster.

### *1.1.6 Paradigmas de programación*

Los paradigmas de programación paralelos son clases de algoritmos que solucionan diferentes problemas bajo las mismas estructuras de control (Hansen, 1993). Cada uno de estos paradigmas define modelos de como paralelizar una aplicación mediante la descripción general de cómo distribuir los datos y de cómo es la interacción entre las unidades de cómputo. Al momento de diseñar una aplicación paralela es importante conocer que alternativas de paradigmas existen, así como conocer sus ventajas y desventajas, puede ser que dependiendo del problema sea innecesario utilizar un tipo de paradigma o bien la utilización de un determinado paradigma disminuirán las prestaciones en comparación a otro (Márquez Pérez et al., 2011). Existen diferentes clasificaciones de paradigmas de programación, pero un subconjunto habitual (Buyya, 1999) en todas ellas es:

- ***Divide and Conquer*:** En este paradigma el problema es dividido en una serie de subproblemas. Se realizan tres operaciones: dividir, computar y unir. La estructura del programa es del tipo árbol, siendo los subproblemas las hojas que ejecutarán los procesos o *threads*[3]. Cada uno de estos subproblemas se soluciona independientemente y sus resultados son combinados para producir el resultado final.
- ***Pipeline:*** Este paradigma está basado en una aproximación por descomposición funcional. La aplicación se subdivide en subproblemas o procesos y cada subproblema se debe completar para comenzar el siguiente proceso, uno tras otro. En un pipeline los procesos se denomina etapas y

[3] En informática, un hilo de ejecución es la secuencia más pequeña de las instrucciones programadas que se pueden gestionar de forma independiente por un programador, que es típicamente una parte del sistema operativo.

cada una de estas etapas resuelve una tarea en particular. También es importante destacar que en este paradigma se logra una ejecución concurrente si las diferentes etapas del pipeline están llenas o existe en ellas un flujo de datos.

- ***Master-Worker:*** El paradigma *master-worker* también es conocido como *task-farming* o *master-slave*. Este paradigma se compone de dos tipos de entidades: un *master* y varios *workers*. El *master* es el encargado de descomponer el problema en trabajos más pequeños (o se encarga en dividir los datos de entrada en subconjuntos) y distribuir las tareas de la aplicación a los diferentes *workers*. Posterior a esto se realiza una recolección de los resultados parciales para procesar el resultado final. Los *workers* solo cumplen la función de recibir la información, procesarla y enviar los resultados al *master*.
- ***SPMD:*** En el paradigma *Single Program Multiple Data* (SPMD), para un determinado número de unidades de computo se ejecutan sobre estas un mismo código, pero sobre un subconjunto distinto de los datos de entrada. Es decir, partir los datos de la aplicación entre los nodos (o maquinas) disponibles en el sistema. El paradigma SPMD describe un comportamiento iterativo en el cual antes de finalizar una iteración se realiza una fase de sincronización, en esta fase el intercambio de información se hace entre los nodos vecinos. La información de intercambio dependerá de la naturaleza del problema. En la literatura también se conoce paralelismo geométrico, descomposición por dominios o paralelismo de datos. En la Figura 3 se muestra una representación esquemática de este paradigma.

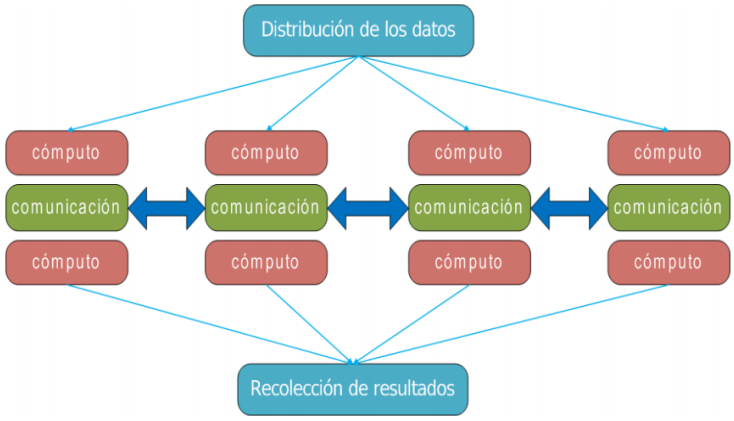


Fig 3. Estructura básica de un programa SPMD, tomado de (Márquez Pérez et al., 2011).

La estructura básica es geométrica regular, con interacción limitada espacialmente. Esta estructura permite que los datos se distribuyan uniformemente entre los procesos, donde cada uno será responsable de un área definida del total de los datos de entrada. Las aplicaciones SPMD suele ser muy eficientes si el entorno

es homogéneo y si los datos están bien distribuidos sobre los procesos (Márquez Pérez et al., 2011). Si la carga de trabajo o la capacidad de computo entre los diferentes procesos es heterogénea, entonces el paradigma requiere el apoyo de alguna estrategia de *load balancing* (balanceo de carga) que sea capaz de adaptar la distribución de los datos durante la ejecución de la aplicación.

En la presente investigación se tomó la decisión de utilizar el paradigma de programación SPMD para el diseño de la aplicación que realizará el procesamiento de los datos. Este paradigma fue seleccionado una vez analizada sus ventajas, desventajas, su rendimiento, su capacidad de escalabilidad y la naturaleza paralela presente en el Visor de Terrenos en el pre-procesamiento de datos.

## 1.2 Protocolos de comunicación

Un protocolo se define como las reglas para la transmisión de la información entre dos puntos. Un protocolo de red de comunicación de datos es un conjunto de reglas que gobierna el intercambio ordenado de datos dentro de la red (Peterson and Davie, 2011). A continuación, se muestran algunos protocolos y sus características.

### *1.2.1 HTTP*

El Protocolo de Transferencia de Hipertexto (HTTP por sus siglas en inglés) es un protocolo de red que se utiliza para publicar páginas web o HTML. HTTP es la base sobre la cual se fundamenta Internet o la WWW[4]. Este funciona a través de solicitudes de respuestas entre un cliente y un servidor (Peterson and Davie, 2011). A una secuencia de estas solicitudes se les conoce como HTTP. Algunas de las características de este protocolo son:

- Opera en la capa de aplicación del modelo OSI[5].
- El puerto estándar para este protocolo es el 80.
- No se requieren certificados, lo que disminuye su seguridad.
- La información que se transmite no es cifrada.

### *1.2.2 HTTPS*

El Protocolo Seguro de Transferencia de Hipertexto Seguro (HTTPS por sus siglas en inglés) es un protocolo de red basado en el protocolo HTTP, destinado a la transferencia segura de datos de hipertexto (Peterson

[4] Sigla de la expresión inglesa *World Wide Web*, “red informática mundial”, sistema lógico de acceso de búsqueda de información disponible en internet, cuyas unidades informáticas son las páginas *web*.
[5] El Modelo OSI es un marco de referencia para la definición de arquitecturas de interconexión de sistemas de comunicaciones.

and Davie, 2011). Es ampliamente utilizado por bancos, tiendas en línea y cualquier tipo de servicio que requiera el envió de datos personales o contraseñas; por ejemplo: transacciones bancarias, comercio electrónico. Algunas de las características de HTTPS son:

- Utiliza un cifrado basado en SSL/TLS[6] para crear un canal cifrado para el tráfico de información sensible.
- El puerto estándar para este protocolo es el 443.
- Opera en una subcapa de la capa de aplicación del modelo OSI, cifrando un mensaje HTTP previo a la transmisión y descifrando un mensaje una vez recibido.

### *1.2.3 SSH*

SSH (o *Secure Shell*) es un protocolo que facilita las comunicaciones seguras entre dos sistemas usando una arquitectura cliente/servidor y que permite a los usuarios conectarse a un *host* remotamente. SSH encripta la sesión de conexión, haciendo imposible que alguien pueda obtener contraseñas no encriptadas (Barrett and Silverman, 2001). Este protocolo está diseñado para reemplazar los métodos más viejos y menos seguros para registrarse remotamente en otro sistema a través de la *Shell* de comando, tales como ***telnet*** o ***rsh***. También permite copiar archivos entre *hosts* mediante el programa **scp**. Algunas de las características de este protocolo son:

- Después de la conexión inicial, el cliente puede verificar que se está conectando al mismo servidor al que se conectó anteriormente.
- El cliente transmite su información de autenticación al servidor usando encriptación robusta de 128 bits.
- Todos los datos enviados y recibidos durante la sesión se transfieren por medio de encriptación de 128 bits, lo cual lo hacen extremadamente difícil de descifrar y leer.
- El cliente tiene la posibilidad de reenviar aplicaciones X11 desde el servidor. Esta técnica, llamada reenvío de X11, proporciona un medio seguro para usar aplicaciones gráficas sobre una red.

### *1.2.4 SFTP*

SFTP es un protocolo de transferencia de archivos que utiliza el protocolo SSH para asegurar los comandos y los datos que se transfieren entre el cliente y el servidor, por lo que dejan de ser vulnerables a escuchas fugitivas, interferencias o falsificaciones (Barrett and Silverman, 2001). Con SFTP, los datos transferidos

[6] SSL y TLS son protocolos criptográficos que proporcionan comunicaciones seguras por una red, comúnmente internet.

entre el cliente y el servidor están cifrados, lo que evita que usuarios no autorizados tengan acceso a ellos. *SSH File Transfer Protocol* (SFTP), es completamente diferente del protocolo FTP (*File Transfer Protocol*). SFTP fue construido desde cero y añade la característica de FTP a SSH. Sólo usa un canal de comunicación, envía y recibe los mensajes en binario (y no en formato texto como lo hace FTP).

De los protocolos de comunicación mencionados anteriormente se decidió utilizar **SSH** y **SFTP** en la presente investigación, ya que encripta toda transferencia de datos entre el cliente y el servidor, lo cual es de vital importancia para aumentar el nivel de seguridad en sistemas complejos. Además, mediante este protocolo es posible ejecutar comandos en el *host* remoto a través de la *Shell* de comando; siendo útil esta característica a la hora de administrar servidores.

## 1.3 Herramientas para el manejo de recursos en un *cluster* de computadoras

Para la correcta administración de un *cluster* es necesario la existencia de algún manejador de recursos computacionales; tales como cantidad de nodos, memoria, manejo de espacio de disco. En este epígrafe se abordan las características y funcionalidades de algunas de estas herramientas.

### *1.3.1 Torque*

Torque (*Tera-scale Open-source Resource and Queue Manager*) es un manejador de recursos que permite manejar trabajos encolados en nodos de cómputo distribuidos, es decir permite enviar tareas a ejecutarse en el *cluster,* en colas previamente creadas en su instalación o posterior a la misma. Posee todo un conjunto de características, comandos y configuraciones para el manejo de las colas, calendarización, planificación, respuesta a fallos del sistema, administrar múltiples trabajos, usuarios y grupos. Junto con el planificador Maui[7] puede realizar toda una serie de políticas que permiten el uso eficiente de los recursos computacionales en un *cluster* de computadoras (Montero Montero, 2009). Torque a través de una interfaz de comandos de usuario y una biblioteca API[8], permite:

- Gestionar más de 5000 trabajos activos o en espera.
- Parar o reanudar un trabajo.
- Cancelar un trabajo.
- Mostrar el estado y las características de un trabajo.

[7] Funciona en conjunto con Torque. Es un planificador para *clusters* y supercomputadores, el cual es capaz de definir un conjunto de políticas, prioridades dinámicas entre otras características.
[8] La Interfaz de Programación de Aplicaciones es un conjunto de subrutinas, funciones y procedimientos que ofrece cierta biblioteca para ser utilizada por otro *software* como una capa de abstracción.

- Enviar un trabajo.
- Tolerancia a fallos.
- Escalable.

### *1.3.2 LSF*

LSF (*Load Sharing Facility*) es una *suite* comercial de productos para administración de recursos que planifica, monitoriza y analiza la carga de trabajos de una red de computadoras. En la *suite* se encuentra un sistema de colas llamado LSF *batch*. LSF es un sistema propietario desarrollado por la compañía *Plataform* (Martínez, 2014). Algunas de las características de LSF son:

- El usuario, al igual que el administrador de LSF posee herramientas gráficas, además de la línea de comandos.
- El administrador de LSF tiene la posibilidad de configurar el sistema de forma sencilla a través de la herramienta gráfica.
- El usuario, una vez comenzada la ejecución de su trabajo, puede controlar el mismo (detener su ejecución, eliminarlo, consultar la salida).
- Soporta migración de procesos y también balanceo de carga.
- Soporte para aplicaciones paralelas: ejemplo MPI; comportándose como un asignador de recursos
- Es un sistema soportado por casi todos los sistemas operativos Unix y Windows NT/2000.

### *1.3.3 Oracle Grid Engine (OGE)*

*Oracle Grid Engine* es actualmente una de las herramientas comerciales para la administración de carga de trabajo más utilizadas. Ofrece la posibilidad de adaptarse a distintos entornos computacionales y distintas cargas de trabajo. OGE se utiliza normalmente en un *cluster* de alto rendimiento y es responsable de aceptar, enviar, cancelar trabajos y gestionar la ejecución remota y distribuida de un gran número de recursos computacionales, tales como procesadores, memoria, espacio en disco, GPU entre otros.

Esta herramienta permite implementar políticas de *sheduling* (planificación) avanzadas, reserva de recursos y adapta automáticamente el tamaño del *cluster* basándose en objetivos de nivel de servicios. OGE también permite compartir recursos entre dos *cluster* OGE con el fin de aumentar la utilización del centro de datos, pudiendo inclusive hacer uso de servicios de una nube pública o privada cuando sea necesario (Martínez, 2014).

La última versión de OGE mejora el mecanismo de preferencias de ejecución de trabajos, resultando en un aumento en la utilización del centro de datos en la organización. Esta nueva actualización incluye la posibilidad de especificar límites en la ejecución de trabajos que emiten los usuarios. Para usuarios que emiten grandes trabajos paramétricos, el planificador de OGE les permite compartir recursos de manera más equitativa, lo que resulta en una mayor eficiencia y menores tiempos de ejecución. OGE soporta varias plataformas de ejecución, como Windows, Linux y varias versiones de Solaris (Martínez, 2014).

### *1.3.4 SLURM (Simple Linux Utility for Resource Management)*

SLURM es un planificador de tareas de código abierto usado actualmente por muchos de los súper computadores a nivel mundial y por *clusters* de computación. Ofrece tres funcionalidades principales. En primer lugar, permite el acceso exclusivo / no exclusivo de los usuarios a recursos computacionales durante una cierta cantidad de tiempo con el fin de que ejecuten sus trabajos. SLURM también brinda un entorno para inicializar, ejecutar y monitorear trabajos en un conjunto de nodos. Además dirige las peticiones de recursos mediante la administración de colas de trabajo pendientes (Martínez, 2014). SLURM está disponible para determinadas versiones de los sistemas operativos FreeBSD, OS X y Solaris y no para Windows. Algunas de sus características más notables son:

- Altamente escalable (permite planificar hasta 100 000 trabajos independientes).
- Alto rendimiento (hasta 1000 peticiones de trabajo y 500 ejecuciones por segundo).
- Altamente configurable con más de 100 *plugins*.
- Planificación compartida sobre la base de jerarquías de usuarios.
- Los nodos ociosos pueden ser apagados.
- Diferentes sistemas operativos pueden ser usados para cada tarea.
- Software libre.

Una vez analizadas las características y funcionalidades de las herramientas mencionadas anteriormente se decidió implementar los mecanismos de comunicación con Torque; pues permite encolar, controlar y monitorear los recursos computacionales de un *cluster* a través de una interfaz de comando y una API de forma sencilla. Además, es una de las herramientas más usadas en los *cluster* de alto rendimiento. Por un lado, OGE y LSF son herramientas privativas, por lo cual no es una opción adecuada para el desarrollo de la biblioteca, por temas de restricción de licencias en la implementación de la presente investigación.

## 1.4 Tecnologías para el desarrollo de aplicaciones distribuidas

El auge que ha tenido últimamente el uso de las redes ha provocado un crecimiento de los entornos distribuidos y heterogéneos. El desarrollo de aplicaciones distribuidas enfrenta diferencias de arquitectura, tales como el hardware, el sistema operativo el ambiente de desarrollo, el lenguaje de programación e incluso el paradigma de programación, por todo esto ha sido necesario utilizar diferentes tecnologías y mecanismos de desarrollo. La programación distribuida hace uso de distintas tecnologías e incluso puede mezclarlas para generar otras. A continuación, se muestran algunas de las tecnologías para el desarrollo de aplicaciones distribuidas.

### *1.4.1 Hadoop*

***Apache Hadoop*** es un *framework* que permite el procesamiento de grandes volúmenes de datos a través de *clusters*, usando un modelo simple de programación. Además, su diseño permite pasar de pocos nodos a miles de nodos de forma ágil. *Hadoop* es un sistema distribuido que utiliza una arquitectura Maestro-Esclavo (*Master-Slave*, por su traducción al inglés), usando para almacenar datos su Sistema de Ficheros Distribuidos *Hadoop* (HDFS por sus siglas en inglés) y algoritmos de *MapReduce*[9] para hacer cálculos y operaciones complejas (White, 2012). El sistema de archivos está diseñado para mezclar la tolerancia a fallas con un alto rendimiento. Se cargan los bloques para mantener la transmisión constante y generalmente no se almacenan en memoria caché para minimizar la latencia. Ofrece una abstracción fina sobre el almacenamiento local y la sincronización de datos, permitiendo que los programadores se concentren en la escritura de código para analizar los datos. *Hadoop* es basado en el lenguaje de programación Java, por lo que la implementación de aplicaciones es sencilla. Algunas de las ventajas que ofrece *Hadoop* son las siguientes:

No es necesario conocer y definir de una forma exacta cómo son los datos antes de incorporarlos a los sistemas. Esta característica aporta flexibilidad, permite ser ágil en la incorporación de cualquier dato y no desechar datos por su mutabilidad o desconocimiento.

- Además de poder almacenar cualquier tipo de información, permite acceder a esta y procesarla independientemente de su tipo, utilizando distintos paradigmas (*batch*, interactivo, *real-time*) o tecnologías (*MapReduce*, *Spark*, R).
- Se basa en hardware *commodity* y código abierto, por lo que su coste es inferior a otras soluciones de gestión de datos.

[9] *MapReduce* es un *framework* que proporciona un sistema de procesamiento de datos paralelo y distribuido. Su nombre se debe a las funciones principales; que son *Map* y *Reduce*

- No hay que modificar el código o las herramientas que procesan la información si se crece en volumen o número de nodos.
- Utiliza un enfoque distribuido (divide y vencerás).

### *1.4.2 Apache Spark*

Es un motor para el procesamiento distribuido de datos a gran escala; que combina un sistema de computación distribuida a través de *clusters* de ordenadores con una forma sencilla y elegante de escribir programas (Yadav, 2015). Es considerado el primer software de código abierto que hace la programación distribuida realmente accesible a los científicos de datos. Spark fue diseñado para soportar en memoria algoritmos iterativos que se pudiesen desarrollar sin escribir un conjunto de resultados cada vez que se procesaba un dato (Yadav, 2015). Esta habilidad para mantener todo en memoria es una técnica de computación de alto rendimiento aplicado al análisis avanzado, la cual permite que Spark tenga unas velocidades de procesamiento que sean 100 veces más rápidas que las conseguidas utilizando *MapReduce*. Debido a su arquitectura logra una escalabilidad lineal, ya que si los datos crecen es posible añadir más ordenadores y tardar lo mismo en la ejecución de tareas.

*Spark* tiene un *framework* integrado para implementar análisis avanzados que incluye la biblioteca *MLib*, el motor de procesamiento de grafos: *GraphX*, Spark *Streaming*, y la herramienta de consulta *Shark*. Esta plataforma asegura a los usuarios la consistencia en los resultados a través de distintos tipos de análisis. Desarrollado en *Scala*, pudiéndose utilizarse sobre *Java*, *R*, o *Python*. Gracias a la completa API que posee, es posible programar complejos hilos de ejecución paralelos en unas pocas líneas de código.

### *1.4.3 MPI-2*

La Interfaz de Paso de Mensaje *(*MPI por sus siglas en inglés*)* es una interfaz estandarizada para la construcción de aplicaciones paralelas basadas en paso de mensajes (Message Passing Interface (MPI) Forum Home Page, 2015). El modelo de programación que subyace tras MPI es: Flujo de Múltiples Instrucciones, Flujo de Múltiples Datos (MIMD por sus siglas en inglés), habiendo también facilidades para el modelo: Programa simple, Múltiples datos (SPMD por sus siglas en inglés). Un caso particular de SPMD es en el que todos los procesos ejecutan el mismo programa con datos diferentes, aunque no necesariamente al mismo tiempo. MPI fue diseñado para computación de alto rendimiento (HPC, por sus siglas en inglés) sobre computadoras masivamente paralelas y *cluster (*Memoria distribuida*)*. Códigos escritos en MPI son portables a una amplia variedad de computadoras (De Floriani et al., 2005). MPI-2 presenta las siguientes características:

- Paralelismo explícito (definido y controlado en su totalidad por el programador).
- Mecanismo de comunicación: Pase de Mensaje (MP por sus siglas en inglés).
- Se puede usar C/C++ y Foltran para programar las aplicaciones.
- Modular, portable, heterogéneo, comunicaciones seguras, comunicaciones punto a punto y comunicación colectiva.
- Proporciona replicación de datos en *clusters*.
- Soporta Lectura/Escritura de ficheros en paralelo.

### *1.4.4 Hive*

*Hive* es un *framework* para *data warehouse* que facilita la consulta y administración de grandes conjuntos de datos residentes en almacenamiento distribuido (White, 2012). Provee un mecanismo para la estructura del proyecto sobre estos datos y consultar los datos usando un lenguaje SQL llamado *HiveQL* (White, 2012). Al mismo tiempo este lenguaje también permite a los programadores que utilizan el modelo de programación *MapReduce* tradicionales conectar sus *mappers* personalizados y *reducers* cuando es inconveniente o ineficiente esta lógica en HiveQL, o sea, ofrece una capa de abstracción sobre *Hadoop*, *MapReduce*, con lo que claramente obtenemos las ventajas de cada uno de ellos.

Algunas cuestiones negativas de *Hive* es que el lenguaje SQL no está orientado a flujo de datos sino a consulta, no hay un Entorno Integrado de Desarrollo (IDE por sus siglas en inglés), el código de una aplicación es difícil de probar y difícil de encontrar bugs, tiempos de ejecución prolongados y variantes, por lo que en ocasiones se hace difícil predecir cuanto tiempo tomará la realización una tarea.

### *1.4.5 Flink*

Se trata de un motor de procesamiento de *streams* o flujos de datos que proporciona capacidades de distribución de datos, comunicaciones y, muy importante, tolerancia a fallos a las computaciones (Apache Flink: Scalable Batch and Stream Data Processing, 2015). El modelo seguido por Apache *Flink* fue pensado desde el principio como alternativa a *MapReduce*, lo que no significa que no pueda acceder y hacer uso tanto de HDFS (*Hadoop Distributed* File *System*) como de YARN (*MapReduce v2*).

En la superficie, *Flink* es muy parecido a *Spark*, sin embargo, internamente se trata de un modelo de programación enteramente centrado en el procesado de flujos o *stream processing*, a diferencia de *Spark* que consigue esta funcionalidad gracias a un enfoque de pequeños procesos por lotes procesados rápidamente y de forma ligera, lo que los desarrolladores llaman *micro-batching*. Otras diferencias

fundamentales entre ambos modelos son la abstracción con la que trabaja cada uno de ellos: *Spark* procesa *RDDs* o bloques de datos, mientras que *Flink* realiza un procesado registro a registro en tiempo real. Algunas de las características de Apache *Flink* son:

- Soporta técnicas de *machine learning* o procesado de grafos de forma nativa sobre su motor de *streaming*.
- Hace más fácil para entornos ya implantados el paso de *streaming* a *batch* que al revés.
- Posee una completa API en Java para el desarrollo de las aplicaciones.
- Contiene una API de creación para la creación y análisis de grafos.
- Permite expresar expresiones con una sintaxis SQL.
- Entorno de ejecución en el que las transformaciones se ejecutan sobre conjuntos de datos tomados de fuentes más estáticas (como ficheros o bases de datos locales).

En la presente investigación se decidió utilizar MPI-2 de las tecnologías mencionadas anteriormente, pues soporta el procesamiento distribuido de datos mediante un sistema de computación distribuida a través de *clusters* de computadoras. Permite la posibilidad de ejecutar un mismo programa con diferentes datos. Además, al ser un estándar se asegura la portabilidad de las aplicaciones a otros sistemas que implementen el estándar MPI-2, manteniendo el mismo diseño. Las aplicaciones se pueden programar usando los lenguajes de programación C/C++, que son del dominio de los integrantes del proyecto.

## 1.5 Aplicación informática y Biblioteca

Una aplicación informática es un tipo de software que permite al usuario utilizar varios tipos de operaciones para realizar tareas complejas sobre dispositivos informáticos. También puede verse como programas independientes que resuelven una necesidad de negocio específica (Pressman, 2014)**.** Algunos ejemplos de estas aplicaciones son los procesadores de textos, hojas de cálculo, y base de datos.

En computación una biblioteca es un conjunto de herramientas de software pequeño y autónomo que ofrece una funcionalidad muy específica al usuario. Normalmente se usa en conjunto con otras herramientas para hacer una aplicación completa, ya que por lo general las bibliotecas no son ejecutables, pero si pueden ser usadas por ejecutables que las necesiten para poder funcionar. La utilización de bibliotecas en la construcción de aplicaciones complejas presenta las siguientes ventajas:

- No tener que volver a escribir el código que resuelve un problema.
- Se ahorra el tiempo de compilar cada vez un conjunto de códigos que ya está compilado.

- El código ya compilado estará probado y será fiable. Puede ser que no sea a la primera vez, pero sí cuando se haya usado en muchos programas distintos y le hayamos ido corrigiendo los errores.

Aunque tanto las bibliotecas como las aplicaciones informáticas son módulos de programas ejecutables. Se diferencian en varios aspectos. Para el usuario final, la diferencia más obvia es que las bibliotecas no son programas directamente ejecutables. Desde el punto de vista del sistema hay dos diferencias fundamentales:

1. Una aplicación puede tener varias instancias ejecutándose simultáneamente en el sistema, mientras que sólo se puede ejecutar una instancia de una biblioteca.
2. A diferencia de una biblioteca, una aplicación puede ser propietaria de elementos como una pila, una memoria global, identificadores de archivo y una cola de mensajes.

Para dar solución al problema planteado se decidió realizar una biblioteca de clases, ya que puede resolver el mismo problema que una aplicación. Su mantenimiento, ampliación y modificación es más sencilla, pues solo se modifican las clases o métodos necesarios sin necesidad de afectar todo el sistema. Además, puede ser utilizada por cualquier aplicación informática que necesite las funcionalidades implementadas en ella, con solo añadirla a su código fuente.

## 1.6 Tecnologías, herramientas y metodología para el desarrollo

Para un correcto desarrollo de la solución se utilizarán un conjunto de herramientas que permitirán implementar de modo rápido y seguro la solución propuesta. En este epígrafe se describen cada una de las herramientas que serán utilizadas.

### *1.6.1 Metodología de desarrollo*

Las metodologías de desarrollo de software son un conjunto de procedimientos, técnicas y ayudas a la documentación para la producción de software. En la actualidad su uso es imprescindible para obtener mejores resultados en la organización y planificación del trabajo, contribuyendo a la obtención de un producto de calidad, que satisfaga tanto a los clientes como a los desarrolladores.

Las metodologías de desarrollo se dividen en dos vertientes:

- Metodologías tradicionales o pesadas.
- Metodologías agiles.

Las metodologías clásicas tradicionales no facilitan el desarrollo rápido de aplicaciones, pues requieren de una excesiva cantidad de documentación y no se adaptan a los cambios por lo que no son métodos adecuados cuando se trabaja en un entorno donde los requisitos pueden variar fácilmente. Las metodologías no agiles, son aplicables cuando el equipo de desarrollo es numeroso, mientras que las metodologías agiles son usadas en equipos pequeños. Estas se centran en el factor humano y el producto de software (Pressman, 2014); es decir, le dan mayor valor al individuo, a la colaboración entre el cliente y el equipo de desarrollo y definen un proceso incremental de desarrollo con iteraciones cortas.

Para el desarrollo de la presente solución no se utilizará una metodología pesada pues el equipo de desarrollo de la presente solución es pequeño, por tanto, se decide utilizar una metodología ágil. Entre las metodologías ágiles más populares se encuentran XP (Lindstrom and Jeffries, 2004) y OpenUP (Balduino, 2007), se decidió implementar el modelo usando esta última metodología debido a la familiaridad que tiene el desarrollador con la misma y a que esta es la utilizada en el desarrollo del Visor de Terrenos.

### *1.6.2 Metodología de desarrollo OpenUp*

Esta metodología define las fases, actividades y artefactos que se generan durante el ciclo del software. La finalidad de esta metodología de desarrollo es garantizar la eficacia mediante el cumplimiento de los requisitos iniciales y minimizar las pérdidas de tiempo en el proceso de generación del software. Entre las ventajas de OpenUP se encuentran:

- Es extensible, pues los procesos se pueden agregar o adaptar según lo vayan requiriendo los sistemas.
- Es ligero y proporciona una comprensión detallada del proyecto, beneficiando a clientes y desarrolladores sobre productos a entregar y su formalidad.
- Se centra en una arquitectura temprana para reducir al mínimo los riesgos y organizar el desarrollo.
- Maneja el ciclo de vida del desarrollo de software de manera apropiada al ofrecer una buena administración de las áreas del proyecto.

### *1.6.3 Lenguajes de programación*

*C Sharp* (*conocido comúnmente como C#*) es un lenguaje de programación de alto nivel y orientado a objetos, el cual se diseñó en específico para la plataforma .Net de Microsoft como un lenguaje que permitiera a los programadores migrar hacia la plataforma. Tiene sus raíces en los lenguajes de programación C, C++ y Java; adapta las mejores características de cada uno de estos lenguajes y agrega nuevas características propias. Algunas de las principales funcionalidades del lenguaje se mencionan a continuación:

- **Nombre de espacios:** C# funciona en el modelo jerárquico de nombres de espacios para mantener la organización en los programas
- **Recolección de Basura:** Al igual que otros lenguajes de programación como Python, Java; C# realiza dinámicamente la liberación memoria cuando un objeto termina su ciclo de vida.
- **Sintaxis:** C# hereda su sintaxis de Java y C/C++ tomando los puntos fuertes de ambos.
- **Métodos:** Los métodos en C# por defecto no son virtuales, por tanto, mejora el rendimiento de las aplicaciones en comparación con otros lenguajes como Java.

*C*++ es un lenguaje de programación de alto nivel y orientado a objetos. Basado en los sólidos fundamentos del lenguaje de programación C (permitiendo también programación estructurada) y otras nuevas características, sin perder la capacidad, estilo y flexibilidad del mismo. En la actualidad ya C++ posee un estándar, lo cual ayuda a la portabilidad de las aplicaciones desarrolladas.

Para la implementación de la solución se utilizarán los lenguajes de programación C# 5.0 y C++ empleando el estándar C11[10]. El primero en vistas de las funcionalidades que este provee y para lograr homogeneidad con el desarrollo del Visor de Terrenos, el cual fue implementado en su totalidad con C#, y el segundo por ser uno de los lenguajes soportados para desarrollar aplicaciones paralelas usando MPI.

### *1.6.4 Lenguaje de Modelado*

Los cambios de empleados en empresas productoras de software es a menudo un gran problema, sino se tiene una forma de comunicación estandarizada de ideas y procesos. Para esto se necesita un lenguaje estandarizado que comunique las ideas a otros desarrolladores y que sirva de apoyo para el análisis de un problema. Como lenguaje de modelado, se selecciona el Lenguaje Unificado de Modelado (UML por sus siglas en inglés) en su versión 2.0. Es el lenguaje gráfico para modelado de sistemas con tecnología Orientada a Objetos que permite especificar, visualizar, construir y documentar. Permite apoyar todo el ciclo de vida de desarrollo de software: especificaciones de analistas, arquitectura, diseño, implementación e implantación. Es un lenguaje de modelado visual fácil de aprender, estándar, estable y configurable.

### *1.6.5 Entorno de Desarrollo Integrado*

[10] C11 es el nombre informal para la norma ISO/IEC 9899:2011, el estándar actual para el lenguaje C que fue ratificado por la ISO en diciembre de 2011.

Un Entorno de Desarrollo Integrado (IDE por sus siglas en inglés) es un software que provee un conjunto de funcionalidades a los programadores para escribir, testear, compilar, interpretar y diseñar interfaces de una aplicación; proporcionándole así rapidez y fluidez en la creación de programas complejos.

Se seleccionó como Entorno Integrado de Desarrollo Visual Studio 2015, ya que el Visor de Terrenos está siendo desarrollado con este IDE. Algunas de las características más importantes de este IDE son:

- Soporta múltiples lenguajes de programación, tales como: C++, C#, F#, Visual Basic .NET.
- Posee herramientas de evaluación y depuración lo cual ayuda a crear aplicaciones en el menor tiempo posible.
- Permite integración con la nube.
- Ofrece un rápido acceso a todos los servicios virtuales, lo cuales podrán ser modificados según las necesidades de la aplicación.
- Posee herramientas de modelado para visualizar la arquitectura del software, tales como diagramas UML, el mapa de código y el explorador de arquitectura.

### *1.6.6 Herramienta CASE*

Las Herramientas de Ingeniería de Software Asistida por Ordenador (CASE por sus siglas en inglés) supone la automatización del desarrollo de software, contribuyendo a mejorar la productividad en el desarrollo de sistemas de información. Automatiza además la documentación, la generación de código, el chequeo de errores y la gestión del proyecto, permitiendo así la reutilización y la portabilidad del software y la estandarización de la documentación.

La herramienta CASE seleccionada es Visual Paradigm 8.0. La misma soporta todo el ciclo de vida del desarrollo de software: análisis y diseño orientado a objetos, construcción, pruebas y despliegue. Permite modelar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. Presenta licencia gratuita y comercial. Es fácil de instalar y actualizar y compatible entre ediciones. Algunas de sus características principales son:

- Permite el control de versiones.
- Consta de un entorno para la especificación de detalles de los casos de uso, incluyendo la especificación del modelo general y de las descripciones de los casos de usos.
- Construcción de diagramas de flujo de datos.
- Posee un generador de informes.

## Conclusiones del capítulo

En este capítulo se abordaron y analizaron un conjunto de conceptos, tecnologías y herramientas. A partir de este análisis se llegó a las siguientes conclusiones:

- Dentro del proceso de visualización del Visor de Terrenos se identificó la fase donde se requieren altas prestaciones computacionales.
- La selección de la estrategia de paralelización por datos permite obtener beneficios de los entornos paralelos debido a que mantiene más procesadores activos y capacidad de escalabilidad.
- La selección de un *Cluster Beowulf* para la ejecución de aplicaciones con altos requerimientos computacionales permite obtener un alto rendimiento sin un gasto monetario elevado.
- Para la implementación de la aplicación encargada de distribuir el procesamiento de datos en un *cluster* se seleccionó la tecnología MPI-2 con el objetivo de dotar a la solución de portabilidad.
- La selección del software Torque para la ejecución de la aplicación desarrollada permite hacer un uso eficiente de los recursos computacionales disponibles en un *Cluster Beowuf.*

.

# Capítulo 2: Análisis y Diseño

## 2.1 Modelo de dominio

Un modelo de dominio es un modelo conceptual de todos los temas relacionados con un problema específico. En él se describen las distintas entidades, sus atributos, papeles y relaciones. Además de las restricciones que rigen el dominio del problema, es el artefacto más importante que se crea durante el análisis orientado a objetos (Larman, 2005). En la Figura 4 se muestra el modelo de domino del sistema.

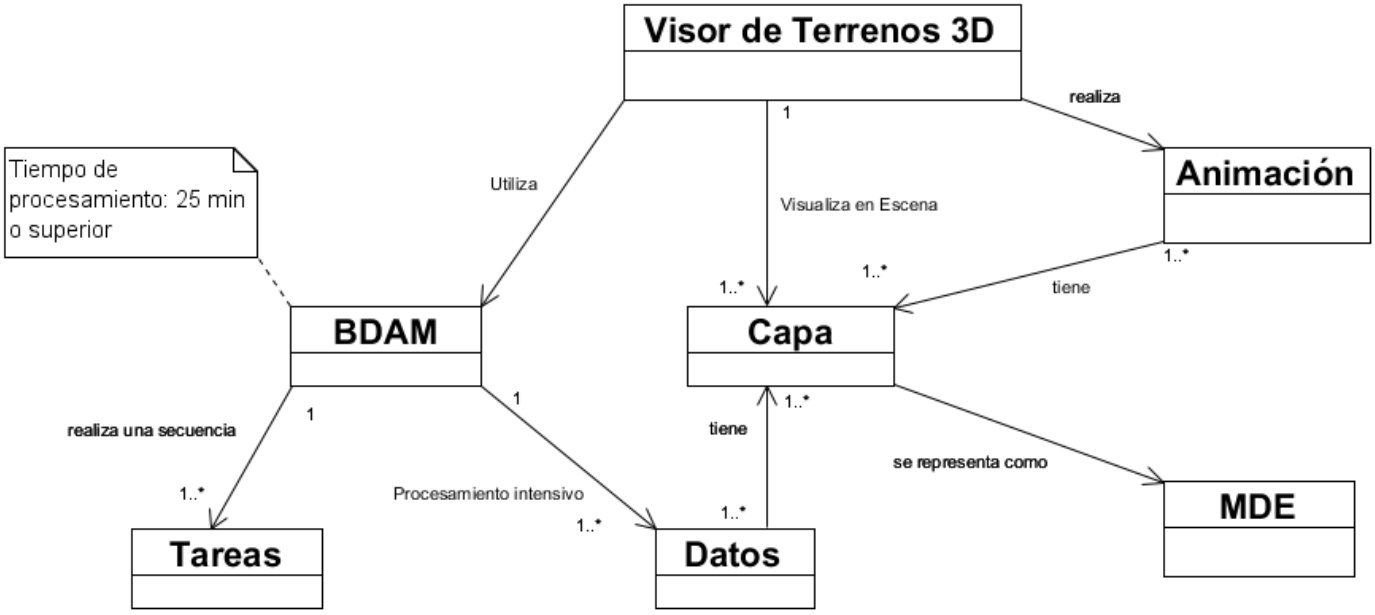


Fig 4. Modelo de dominio del Visor de Terrenos.

Descripción de los objetos del Modelo de Dominio.

- **BDAM:** modelo encargado de realizar el pre-procesamiento de datos en el Visor de Terrenos 3D.
- **Tareas:** conjunto de procesamientos independientes en la etapa de construcción del modelo BDAM.
- **Datos:** conjunto de información de terrenos.
- **MDE:** representa un conjunto de datos de elevación organizados generalmente en forma de matriz.
- **Capa:** representa un MDE para el Visor de Terrenos 3D.
- **Animación:** es una superposición de conjunto de capas.
- **Visor de Terrenos 3D:** se encarga de representar la información en la pantalla.

## 2.2 Especificación de los requisitos del sistema

La especificación de los requisitos del sistema (ERS) es una descripción completa del comportamiento del sistema que se va a desarrollar. El objetivo principal de este es servir como medio de comunicación entre clientes, ingenieros de requisitos y desarrolladores. En la ERS deben recogerse tanto las necesidades de clientes y usuarios (necesidades del negocio, también conocidas como requisitos de usuario*,* requisitos de cliente*,* necesidades de usuario, etc.) como los requisitos que debe cumplir el Sistema software a desarrollar para satisfacer dichas necesidades (requisitos del producto, también conocidos como requisitos de sistema *o* requisitos de software). En esta se facilita el mecanismo para comprender lo que quiere el cliente, analizando necesidades, confirmando su viabilidad, negociando una solución razonable, especificando la solución sin ambigüedad, validando la especificación y gestionando los requisitos para que se transformen en un sistema operacional (Sommerville, 2011).

### *2.2.1 Requisitos funcionales (RF)*

Los requisitos funcionales definen una función del sistema de software o sus componentes, este describe la forma que el sistema debe comportarse ante un conjunto de entrada y situaciones particulares (Sommerville, 2011). Los requisitos funcionales identificados para la presente solución son:

El **RF1 Ejecutar tarea** permite ejecutar una tarea que realiza el procesamiento paralelo de datos con las configuraciones proporcionadas por el usuario en un *Cluster Beowulf.*

El **RF2 Consultar el estado de la tarea** permite comprobar si una tarea que ha sido enviada a un *cluster Beowulf* ha finalizado su ejecución.

El **RF3 Obtener el fichero de error** permite obtener el fichero de error que contiene información de los errores ocurridos durante la ejecución de una tarea.

El **RF4 Obtener el fichero de salida** permite obtener el fichero generado por la ejecución de instrucciones que utilizaban la salida estándar (usualmente la pantalla) para mostrar resultados.

El **RF5 Obtener ficheros procesados** permite recibir los ficheros procesados del *Cluster Beowulf* una vez haya finalizado la ejecución de la tarea*.*

El **RF6 Eliminar tarea** permite eliminar una tarea que esté en estado de ejecución, en cola o de error con todos los datos asociados a ella.

El **RF7 Eliminar directorios creados para la tarea** permite eliminar toda la información asociada a una tarea.

El **RF8 Enviar ficheros** permite copiar ficheros hacia un *host* remoto.

El **RF9 Ejecutar comando** permite ejecutar comandos en un *host* remoto.

### *2.2.2 Requisitos no funcionales (RNF)*

Los requisitos no funcionales, son requisitos que imponen restricciones en el diseño, en la implementación o estándares de calidad. Son propiedades o cualidades que el producto debe tener. Mediante estos requisitos se busca disponer de un sistema manejable y gestionable que ofrezca la funcionalidad requerida de manera fiable, ininterrumpida o con el tiempo mínimo de interrupción, incluso ante situaciones inusuales (Sommerville, 2011).

**Requisitos de software**

**RNF1:** Existencia de alguna biblioteca que implemente el estándar MPI-2 o superior en el *cluster Beowulf*.

**RNF2:** Existencia del software Torque 4.2.9 en el *cluster Beowulf*.

**RNF3:** Existencia de los compiladores g++ y gcc en su versión 4.9.2 o superior.

**RNF4:** Existencia del servidor OpenSSH en su versión 6.7p1 o superior y tener en ejecución el servidor SFTP.

**RNF5:** Existencia de la plataforma .Net en su versión 4.0 o superior en la computadora donde se vaya a utilizar la biblioteca desarrollada.

**Requisitos de hardware**

**RNF6:** Procesador: Dual Core 2.5 GHz como mínimo.

**RNF7:** Memoria RAM: 2GB como mínimo.

**RNF8:** Existencia de un *Cluster Beowulf*.

**Requerimientos de Restricción del Diseño y la Implementación**

**RNF9.** La solución debe ser implementada utilizando los lenguajes de programación C# y C++.

**RNF10.** No deben utilizarse bibliotecas o componentes externos a la solución.

## 2.3 Modelado del sistema

El modelado de sistema es el proceso de desarrollo de modelos abstractos de un sistema, con cada modelo presentando una vista diferente o perspectiva de un sistema (Sommerville, 2011). A continuación, se muestran los modelos generados para la implementación de la biblioteca propuesta.

### *2.3.1 Arquitectura del sistema*

La arquitectura del software se refiere a las estructuras de un sistema compuestas de distintas entidades relacionadas con el sistema con propiedades visibles de forma externa y las relaciones que existen entre ellos (Pressman, 2014).

### *2.3.2 Estilos arquitectónicos*

El software que se construye para sistemas basados en computadoras también exhibe uno de los muchos estilos arquitectónicos. Cada estilo describe una categoría de sistema que abarca un conjunto de componentes (por ejemplo; una base de datos, módulos computacionales) que realizan una función requerida por un sistema, un conjunto de conectores que permiten “la comunicación, coordinación y cooperación entre los componentes”, las limitaciones que definen los componentes, cómo se pueden integrar para formar el sistema, y los modelos semánticos que permiten a un diseñador comprender las propiedades generales de un sistema mediante el análisis de las propiedades conocidas de sus partes constituyentes (Pressman, 2014).

Un estilo arquitectónico es una transformación que se impone en el diseño de todo un sistema. La intención es establecer una estructura para todos los componentes del sistema. En el caso donde exista una arquitectura esta debe ser rediseñada, la imposición de un estilo arquitectónico que dará lugar a cambios fundamentales de la estructura de software incluyendo una reasignación de las funcionalidades de los componentes.

Un patrón arquitectónico, como un estilo arquitectónico, impone una transformación en el diseño de una arquitectura. Sin embargo un patrón difiere de un estilo en un número de maneras fundamentales (Pressman, 2014):

1. El alcance de un patrón es menos amplio, centrándose en un aspecto de la arquitectura en lugar de la arquitectura en su totalidad.
2. Un patrón impone una regla en la arquitectura, que describe cómo el software se encargará de algún aspecto de su funcionalidad a nivel de estructura (por ejemplo, la concurrencia).
3. Un patrón arquitectónico tiende a abordar los problemas de comportamiento específicos dentro del contexto de la arquitectura (por ejemplo, cómo manejar las aplicaciones en tiempo real sincronización o interrupciones). Los patrones se utilizan en conjunción con un estilo arquitectónico para dar forma a la estructura global de un sistema.

Existen varios tipos de estilos arquitectónicos entre los que se encuentran el Flujo de datos, Llamada y retorno, Centrado de datos, Código móvil y *Peer To Peer*. Dentro de estos estilos se eligió Llamada y retorno, ya que permite construir una estructura de programa relativamente fácil para modificar y escalable (Pressman, 2014). Son los estilos más generalizados en sistemas a gran escala. Miembro de estos estilos son los patrones Orientados a Objetos y Estructuras Jerárquica en Capas.

### *2.3.3 Arquitectura en capas*

Dentro del estilo llamada y retorno se eligió el patrón arquitectónico N-capas para el desarrollo de la biblioteca. El objetivo primordial de este patrón es separar la lógica del negocio de la lógica de diseño; definiendo una organización jerárquica en capas, donde cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. A continuación, se mencionan algunas de las ventajas y desventajas del uso de este patrón arquitectónico:

**Ventajas**

- El desarrollo se puede llevar a cabo en varios niveles y, en caso que sobrevenga algún cambio, sólo se ataca el nivel requerido sin tener que revisar entre código mezclado.
- Permite distribuir el trabajo de creación de una aplicación por niveles.
- Permite trabajar en varios niveles de abstracción.
- Facilita la estandarización y la reutilización de capas.

**Desventajas**

- La comunicación a través de las diferentes capas puede hacer ineficiente el sistema en términos de rendimiento.

- No todos los sistemas se pueden estructurar fácilmente en capas.
- Trabajo innecesario por parte de capas más internas redundante entre varias capas.

Para la implementación de la presente solución se definieron dos capas:

**Capa de Acceso a datos:** Esta capa es la responsable del acceso a los datos externos a la biblioteca.

**Capa de Modelación:** En esta capa están alojadas las estructuras de datos responsables para la configuración y ejecución de una tarea en un *cluster Beowulf*.

### *2.3.4 Diagrama de clases del diseño*

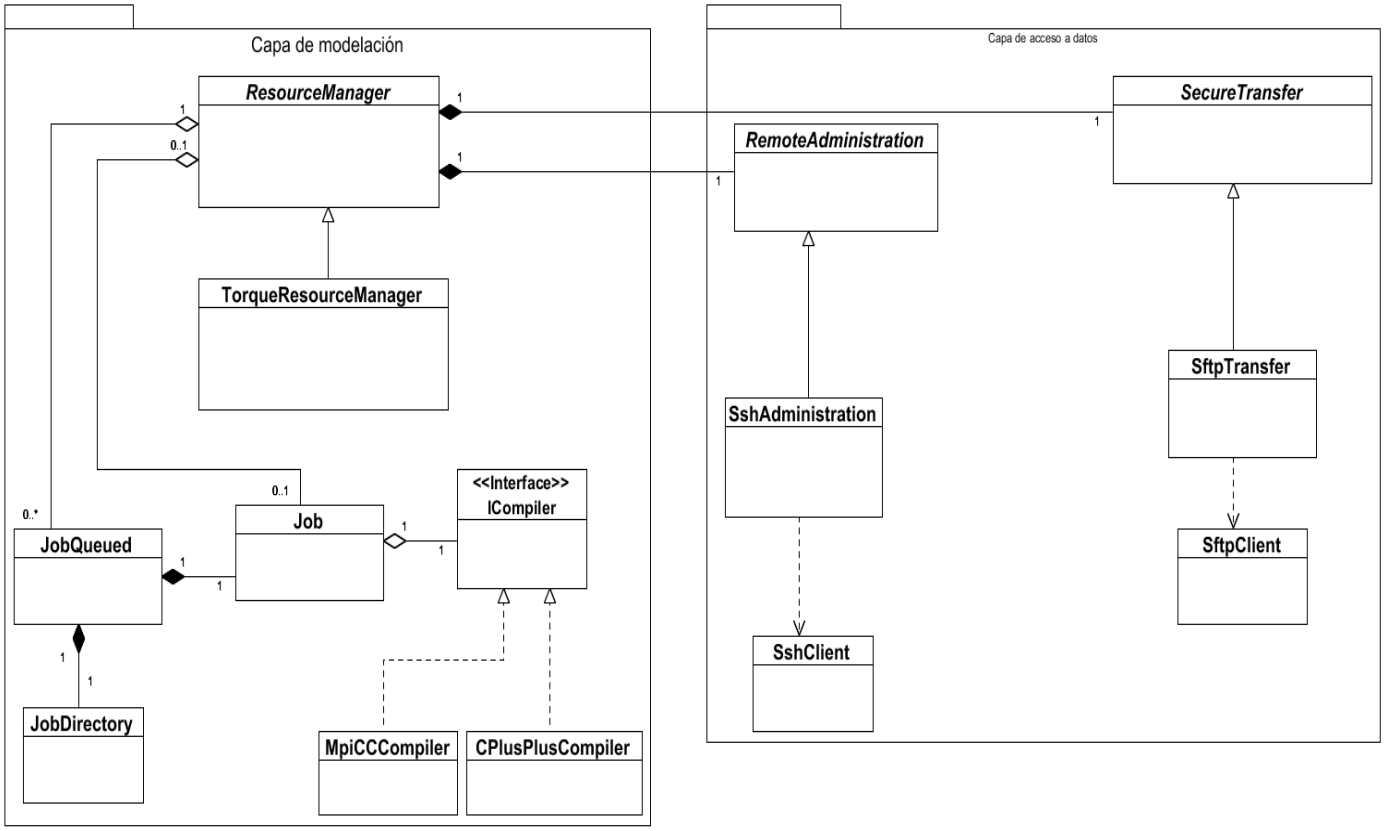


Fig 5. Diagrama de clases del diseño

### *2.3.5 Descripción de las clases del diseño*

Para tener un mejor entendimiento de las clases representadas en el diagrama antes expuesto, se realiza la descripción de las clases del diseño y se muestran las clases con sus atributos y métodos; las restantes descripciones pueden ser consultadas en el Anexo B.

***ResourceManager***: La implementación de esta clase abstracta permite la ejecución de tareas en un cluster de computadoras a través de algún software responsable de manejar los recursos computacionales en un cluster.

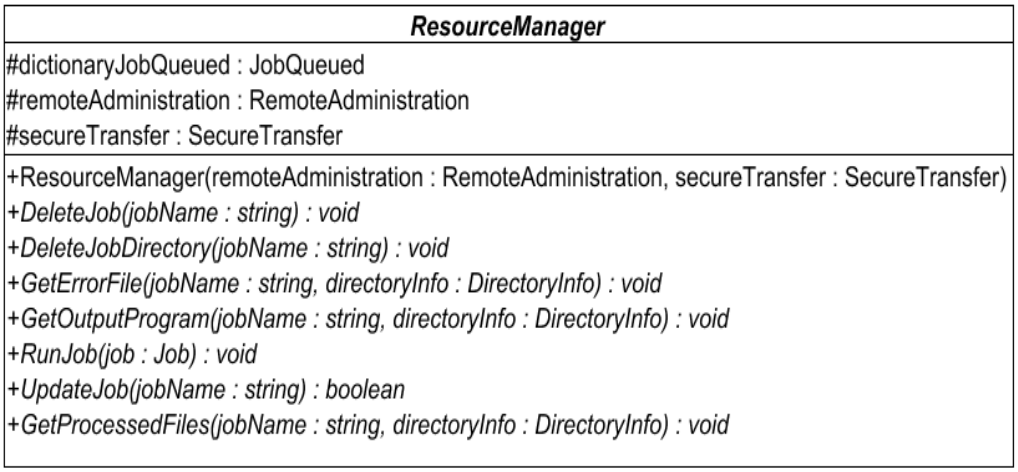


Fig 6. Clase abstracta ResourceManager

***TorqueResourceManager*:** Es la clase encargada de implementar los métodos de la clase abstracta ResourceManager. Esta contiene los métodos RunJob(), DeleteJob(), UpdateJob(). GetProcessedFiles(), DeleteJobDirectory(), GetErrorFile() y GetOutputProgram(). Mediante estos métodos permite ejecutar, eliminar, obtener estado, obtener ficheros procesados, eliminar directorios creados, obtener información de errores durante la ejecución y la información generada por la salida estándar de una tarea de procesamiento de datos en un *Cluster Beowulf,* a través del software Torque instalado en el *cluster*.

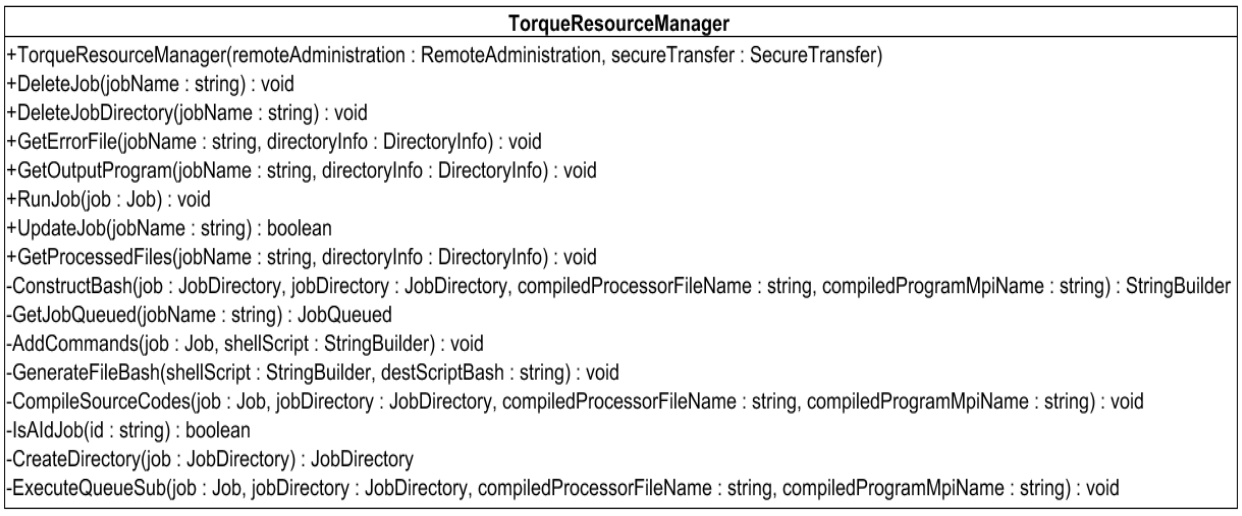


Fig 7. Clase TorqueResourceManager

***RemoteAdministration*:** La implementación de esta clase abstracta permite ejecutar comandos en un *host* remoto.

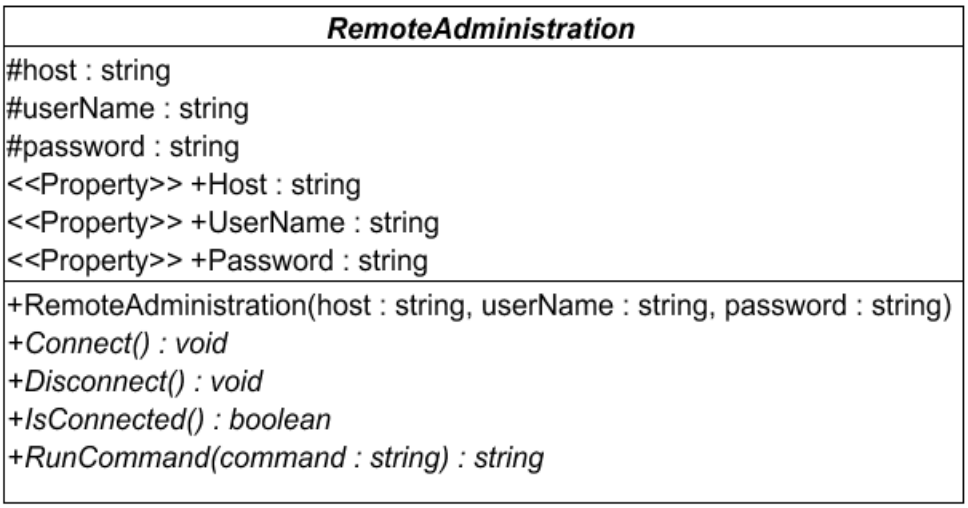


Fig 8. Clase abstracta RemoteAdministration

***SshAdministration*:** Es la clase encargada de implementar la clase abstracta RemoteAdministration, permitiendo ejecutar comandos en un *host* remoto a través de una conexión ssh. Esta posee los métodos Connect(), Disconnect(), IsConnected() y RunCommand() para conectar, desconectar, comprobar si existe conexión y ejecutar comandos respectivamente.

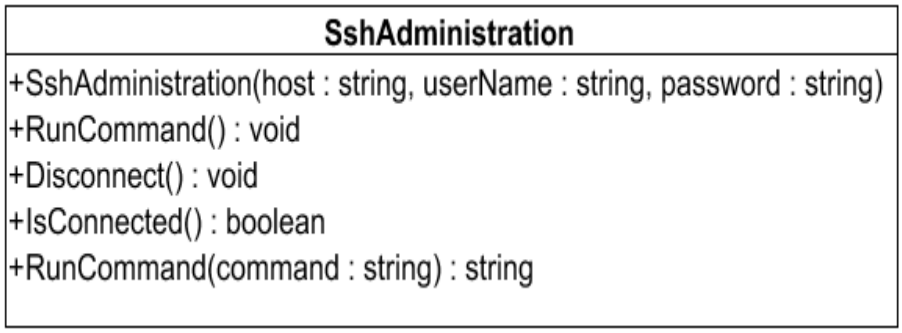


Fig 9. Clase SshAdministration

***SecureTransfer*:** La implementación de esta clase abstracta permite transferir datos a un *host* remoto a través de un protocolo seguro de comunicación. La Figura 10 muestra las características de la clase.

***SftpTransfer*:** Esta clase es la encargada de implementar la clase abstracta SecureTransfer, permitiendo copiar ficheros y manipular directorios en un *host* remoto a través del protocolo de comunicación SFTP. A través de los métodos CreateDirectory(), DeleteDirectory(), DeleteFile(), Download(), Upload() se puede

crear un directorio, eliminarlo, eliminar un fichero, descargar directorios y enviar ficheros respectivamente entre otras funcionalidades que ofrece la misma. En la Figura 11 muestra las funcionalidades que ofrece la clase antes descrita.

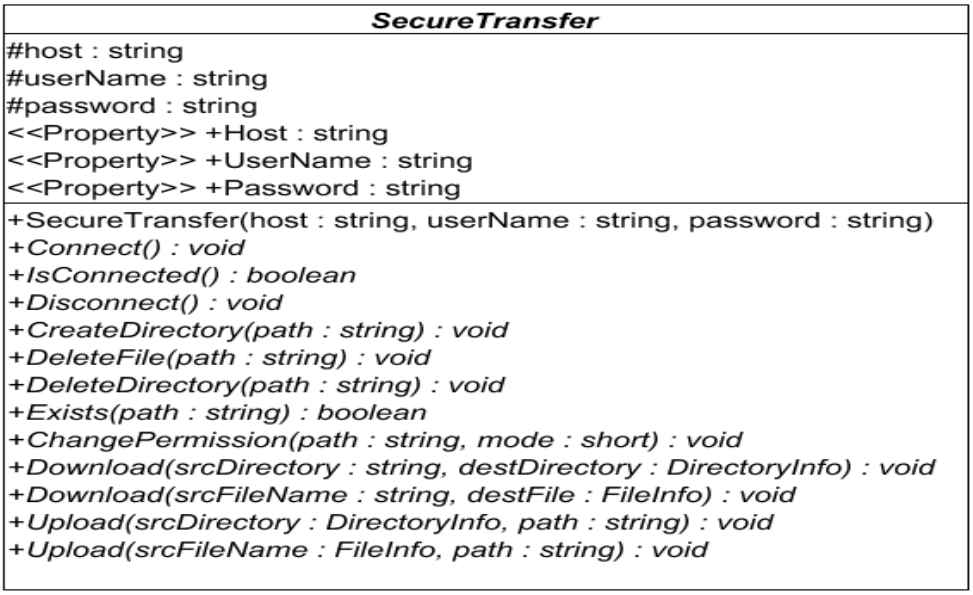


Fig 10. Clase abstracta SecureTransfer

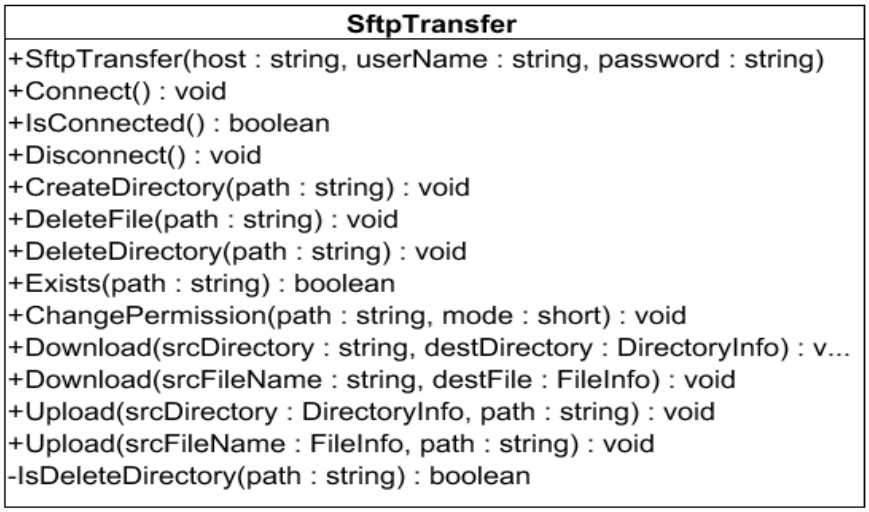


Fig 11. Clase SftpTransfer

***Job*:** Es la clase que representa una tarea a ser ejecutada en un *cluster Beowulf.* A través de la misma se especifica las características con la que queremos ejecutar cualquier tarea como: nombre, cantidad de nodos, cantidad de procesos por nodos, cantidad de memoria RAM, arquitectura del sistema operativo en

el cual deberá ejecutarse, directorio donde se encuentran los ficheros a ser procesados, programa ejecutable que procesa cada fichero entre otras opciones. En la Figura 12 se muestra todas las especificaciones para la clase Job.

### *2.3.5 Patrones de diseño*

Un patrón de diseño constituye una solución a un problema de diseño recurrente en programación orientada a objetos. Este nomina, abstrae e identifica los aspectos claves de una estructura de diseño común, lo que los hace útiles para la crear un diseño orientado a objetos reutilizable. El patrón de diseño identifica las clases e instancias participantes, sus roles, colaboraciones, y la distribución de responsabilidades. Cada patrón de diseño se centra en un problema concreto, describiendo cuando aplicarlo y si tiene sentido hacerlo teniendo en cuenta otras restricciones de diseño, así como las consecuencias, ventajas e inconvenientes de su uso (Larman, 2005).

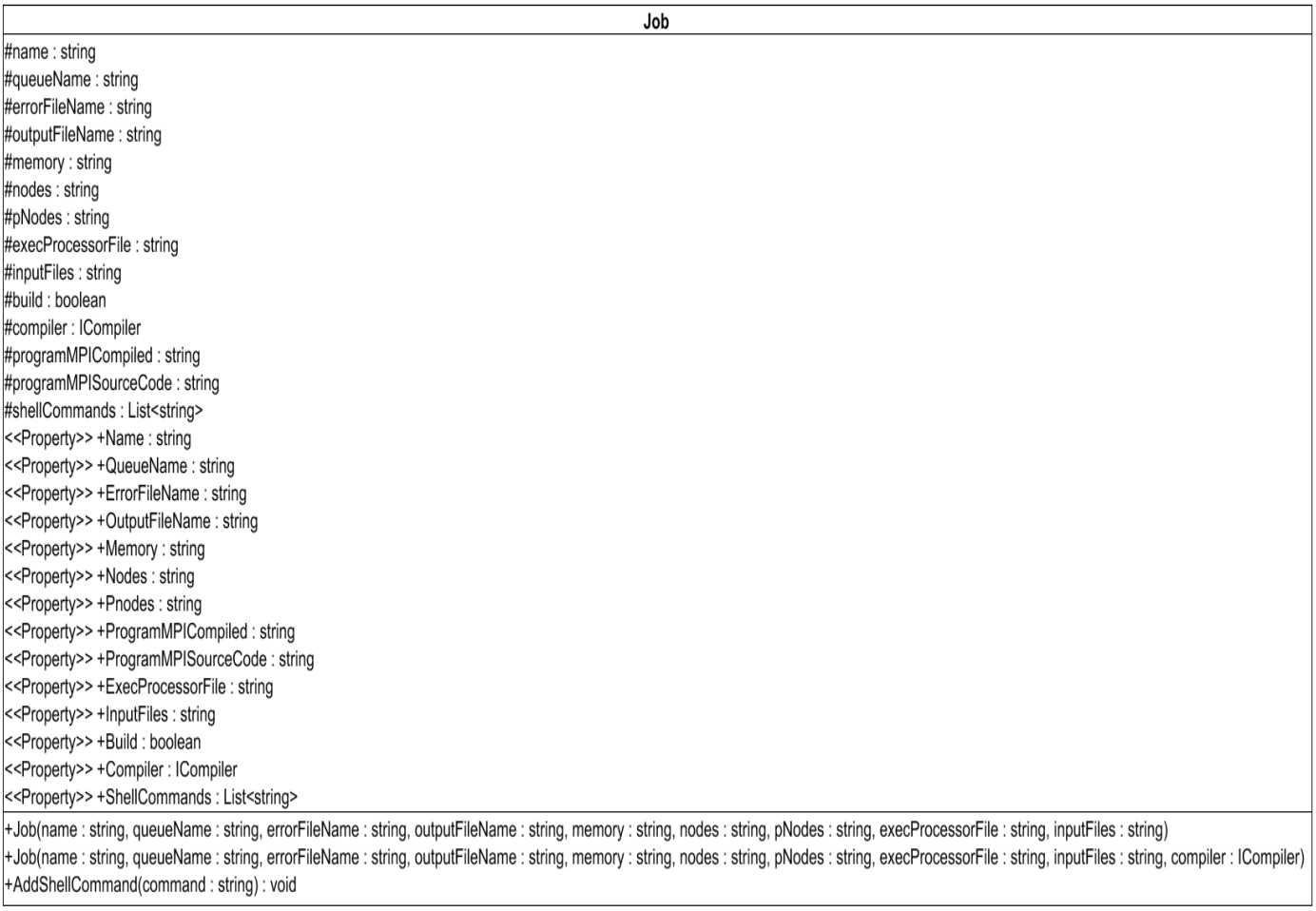


Fig 12. Clase Job

### *2.3.5 Patrones de diseño*

Un patrón de diseño constituye una solución a un problema de diseño recurrente en programación orientada a objetos. Este nomina, abstrae e identifica los aspectos claves de una estructura de diseño común, lo que los hace útiles para crear un diseño orientado a objetos reutilizable. El patrón de diseño identifica las clases e instancias participantes, sus roles, colaboraciones, y la distribución de responsabilidades. Cada patrón de diseño se centra en un problema concreto, describiendo cuando aplicarlo y si tiene sentido hacerlo teniendo en cuenta otras restricciones de diseño, así como las consecuencias, ventajas e inconvenientes de su uso (Larman, 2005).

Los Patrones de Principios Generales para Asignar Responsabilidades (GRASP por sus siglas en inglés) describen los principios fundamentales de la asignación de responsabilidades a objetos expresado en forma de patrones (Larman, 2005). Los diagramas de interacción definen gráficamente estas operaciones a partir de los objetos en interacción, que se responsabilizan de una actividad determinada. En la presente solución se utilizaron los siguientes patrones GRASP:

**Patrón Experto:** Se le asigna la responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir la necesidad. A continuación, se muestran varios ejemplos del uso del patrón:

- La clase *TorqueResourceManager* tiene la responsabilidad de ejecutar tareas en *clusters*.
- La clase *JobQueued* tiene la responsabilidad de almacenar la información de una tarea lanzada al cluster.

**Patrón Creador:** Se le asigna la responsabilidad relacionadas con la instanciación de objetos. El propósito fundamental de este patrón es encontrar un creador que se debe conectar con el objeto producido en cualquier evento (Larman, 2005). En la presente solución este patrón se evidencia en la clase *ResourceManager* la cual crea una instancia de la clase *RemoteAdministration* y *SecureTransfer,* la cuales permiten la ejecución de comandos, envió y recepción de datos en un host remoto.

**Patrón Bajo Acoplamiento:** Este patrón estimula asignar una responsabilidad de modo que su colocación no incremente el acoplamiento tanto que produzca los resultados negativos propios de un alto acoplamiento. Soporta el diseño de clases más independientes, que reducen el impacto de los cambios, y también más reutilizables, que acrecienten la oportunidad de una mayor productividad (Larman, 2005). Este patrón se evidencia en todas las clases del diseño.

**Patrón Alta Cohesión:** Es la meta principal que ha de tenerse en cuenta en cada momento en todas las decisiones de diseño. Es un patrón evaluativo que el desarrollador aplica al valorar sus decisiones de diseño. Una clase de alta cohesión posee un número relativamente pequeño, con una importante funcionalidad relacionada y poco trabajo que hacer. Colabora con otros objetos para compartir el esfuerzo si la tarea es grande. Este patrón se evidencia principalmente en la clase *TorqueResourceManager* la cual divide sus responsabilidades con las clases *JobQueued, Job* y las instancias de las clases *RemoteAdministration* y *SecureTransfer* que son obtenidas a través de la herencia.

Grupo de cuatro (Vlissides et al., 1995) define un conjunto de patrones de diseño que se evidencian en el proceso de construcción de software. De este conjunto se utilizó en la solución el siguiente:

**Patrón Adapter:** Este patrón tiene como objetivo convertir la interfaz de una clase existente en la interfaz esperada por los clientes también existentes para que puedan trabajar de forma conjunta. Se trata de conferir a una clase existente una interfaz para responder a las necesidades de los clientes (Vlissisdes et al., 1995). La clase abstracta *SecureTransfer* define la firma de los métodos de la familia de objetos referentes al envío y recepción de datos del *cluster*. La clase *SftpTransfer* representa un adaptador el cual implementa la firma de los métodos, utilizando las funcionalidades de la clase adaptada *SftpClient* para responder a las necesidades de la clase abstracta. La clase *SshAdministration* también representa un adaptador, pero esta utiliza las clases adaptadas *SshClient* para responder a las necesidades de la clase abstracta.

## 2.4 Proceso propuesto para el pre-procesamiento paralelo del modelo BDAM

Una vez solicitada la representación multi-resolución de un modelo digital de elevación por parte del Visor de Terrenos, el MDE es enviado al modelo BDAM donde se da inicio a la etapa de construcción. Para la construcción del modelo BDAM se utilizarán un conjunto de ficheros, los cuales guardan la información que es necesaria pre-procesar. El procesamiento de los ficheros es realizado por un programa, el cual toma como entrada un fichero y da como salida otro con la información procesada. A continuación, se muestra los pasos que se siguen durante la construcción:

1. Se cargan los ficheros a procesar, el programa que realiza el procesamiento de los ficheros y el programa MPI encargado de paralelizar el pre-procesamiento de los datos en el *cluster*.
2. Se copia toda la información anterior al *cluster* mediante una conexión ssh para preparar los datos.

3. Se ejecuta el programa MPI para realizar el procesamiento de los ficheros mediante el programa especificado por el usuario; donde cada proceso MPI realiza el pre-procesamiento de un conjunto de ficheros.
4. Una vez que se termina el procesamiento de los ficheros se procede a su organización.
5. Se copian hacia el Visor de Terrenos los ficheros procesados para su posterior utilización.

## Conclusiones del capítulo

Durante el presente capítulo se realizó el análisis y el diseño de la biblioteca para el pre-procesamiento del modelo BDAM sobre un *cluster* de computadoras. Se mostró el modelo de dominio de la aplicación a desarrollar, mostrando los principales conceptos y su relación. Después, durante la especificación de requisitos se obtuvieron nueve requisitos funcionales y ocho no funcionales. La arquitectura propuesta y los patrones de diseño utilizados, permiten la extensibilidad de la solución, obteniendo un diseño entendible para otros desarrolladores.

Al final del capítulo se ofrece una explicación del proceso para la construcción del modelo BDAM en paralelo utilizando la biblioteca. El resultado del pre-procesamiento de los datos será utilizado por el modelo BDAM para pasar a su próxima etapa.

# Capítulo 3: Implementación y Pruebas

## 3.1 Estándares de codificación

El uso de un estándar de codificación o de programación, es de gran relevancia al entregar un producto de software, lo cual repercute en la calidad de este. Al recoger todos los aspectos de la generación de código, permite mejorar las prácticas de programación, la eliminación de ambigüedades en el código, logrando un conocimiento y un seguimiento más fácil para el equipo de desarrollo. La presente investigación sigue el estándar de codificación de .Net detallado en (Hunt, 2007).

### *3.1.1 Terminología y definiciones*

**Estilo Camel:** Una cadena con la primera letra minúscula y la primera letra de cada palaba subsiguiente mayúscula.

**Estilo Pascal:** forma de escribir una palabra donde el primer carácter de todas las palabras, se escribe en mayúscula y los otros caracteres en minúsculas.

**Identificador:** define un nombre único para declarar un objeto o instancia de alguno.

**Modificador de acceso:** define la accesibilidad de los miembros de las clases y métodos. Las palabras claves de C# utilizadas con este propósito son *public*, *protected*, *internal*, *private*.

### *3.1.2 Convenciones y estándares de nombres*

En esta sección se describen algunas convenciones para nombrar el nombre del proyecto, archivos del código fuente, e identificadores, incluyendo variables, propiedades, métodos, parámetros, clases, interfaces y espacios de nombre.

Tabla 1. Convenciones y estándares de nombres en el formato identificador/Convención

| Identificador | Convención |
|---|---|
| **Clase (*Class*)** | **Estilo Pascal.**<br>Se usa un sustantivo para el nombre de la clase. En case de heredar de otra, se le agrega un sufijo apropiado.<br>Ejemplo:<br>public DataBdamCl : BdamCl {….} |
| **Archivo fuente** | **Estilo Pascal.**<br>El nombre de los archivos coincide con el nombre de la clase.<br>Ejemplo: |

| | |
|---|---|
| | Para la clase DataBdamCl el nombre del archivos es DataBdamCl.cs |
| **Interfaz** | **Estilo Pascal.**<br>Siempre se adiciona la letra mayúscula I.<br>Ejemplo:<br>interface IConnectionBdamCl |
| **Método** | **Estilo Pascal.**<br>El nombre viene en el formato verbo o verbo-sustantivo<br>Ejemplo:<br>public void Connect() {….} |
| **Propiedad (Property)** | **Estilo Pascal.**<br>Ejemplo:<br>public string SourceFiles ()<br>{<br>get {return _sourceFiles;}<br>set {_sourceFiles = value;}<br>} |
| **Atributo (public, protected, internal)** | **Estilo Pascal.**<br>Usar propiedades en vez de atributos públicos.<br>Ejemplo:<br>protected destFilesProcessed; |
| **Atributo ( private )** | **Estilo Camel.**<br>Al hombre del atributo se le concatena un guion bajo (_). Si el tipo del atributo es un boolean, agregar (is)<br>Ejemplo:<br>private int _numberProcess;<br>private bool _isConnected; |
| **Variable ( local )** | **Estilo Camel.**<br>Al nombre de la variable concatenarle el sufijo auxiliary<br>Ejemplo:<br>string auxiliaryCommand; |
| **Parámetro** | **Estilo Camel.**<br>Al nombre de la variable concatenarle el sufijo auxiliary<br>Ejemplo:<br>public string RunCommand(string command)<br>{<br>} |

### *3.1.3 Estilo de código*

En el estilo de codificación están la mayoría de las incoherencias y las controversias entre desarrolladores, pues cada uno tiene sus preferencias y rara vez son dos iguales. Sin embargo, el diseño coherente, el formato y la organización son claves para crear código fácil de mantener (Hunt, 2007). El estilo utilizado para la implementación es el que se detalla a continuación:

1. Siempre iniciar las llaves en una nueva línea.
2. En las sentencias condicionales siempre usar corchetes.
3. Siempre usar margen de tamaño cuatro.
4. El orden a seguir para los elementos en una clase es el siguiente:
   a) Atributos.
   b) Constructores y Finalizadores.
   c) Enumeradores, Estructuras y Clases.
   d) Propiedades.
   e) Métodos.
5. El orden a seguir para los elementos en una clase es el siguiente:
   *a) public*
   *b) protected*
   *c) internal*
   *d) private*
6. Cada variable se declara independientemente, no en la misma línea.
7. Todos los comentarios se escriben en el mismo idioma sin errores gramaticales.

## 3.2 Implementación del pre-procesamiento paralelo del modelo BDAM

En esta sección se abordarán los detalles de implementación del proceso propuesto para la ejecución del procesamiento paralelo del modelo BDAM.

### *3.2.1 Programas utilizados para el procesamiento paralelo de los ficheros*

Para ejecutar el procesamiento paralelo de los datos (conjunto de ficheros) del modelo, se utiliza un programa para realizar el procesamiento paralelo de los datos (en lo adelante *DistributorMPI*); el cual utiliza la biblioteca *MPICH2* en su versión 3.1, y otro que realiza el procesamiento del contenido de un fichero (en lo adelante *ProcessorFile*) el cual es proporcionado por el usuario de la biblioteca. El *DistributorMPI* recibe tres argumentos: el primero es la dirección de un *ProcessorFile,* el segundo es el directorio donde se

encuentran todos los ficheros a procesar y el tercer argumento es el directorio donde estarán los ficheros procesados. *ProcessorFile* recibe dos argumentos para su correcta ejecución: el primero es la dirección en el *cluster* de un fichero y el segundo es un directorio, el cual almacenará otro fichero con los datos procesados. Un ejemplo de como se le debe pasar los argumentos a *ProcessorFile* es el siguiente*:* *ProcessorFile /home/yurian/Tarea1/input_files/fichero1 /home/yurian/Tarea1/output_files/fichero1.*

### *3.2.2 Creación de la estructura de directorios en el cluster*

Para lograr una organización del contenido que se necesita en el *cluster,* se utiliza la estructura de directorio que se muestra en la Figura 13. Esta estructura de directorio estará en el directorio principal del usuario cuando se autentica mediante una conexión ssh; usualmente /home/nombre_de_usuario/. El directorio **Nombre de la tarea** constituye el identificador especificado por el usuario para su tarea. Para almacenar los errores y la salida producida después de la ejecución del procesamiento de los ficheros; se utiliza el directorio **error_file** y **output_program:** en el directorio **error_file** se genera un fichero de error llamado *NombreDeLaTarea.err* y uno de salida llamado *NombreDeLaTarea.out***.** Los directorios **input_files** y **output_files** se utilizan para almacenar los ficheros a procesar y los procesados respectivamente. Los programas **DistributorMPI** y **ProcessorFile** son utilizados para ejecutar el procesamiento paralelo de los ficheros. El script tarea.sh tiene toda la configuración que el usuario especificó para su tarea; como nombre de la tarea, cola que se encargará de ejecutar la tarea, cantidad de nodos, cantidad de procesos por nodo, cantidad de memoria RAM entre otras opciones. Un ejemplo de un script para ser ejecutado con Torque es el que se muestra en la Figura 15.

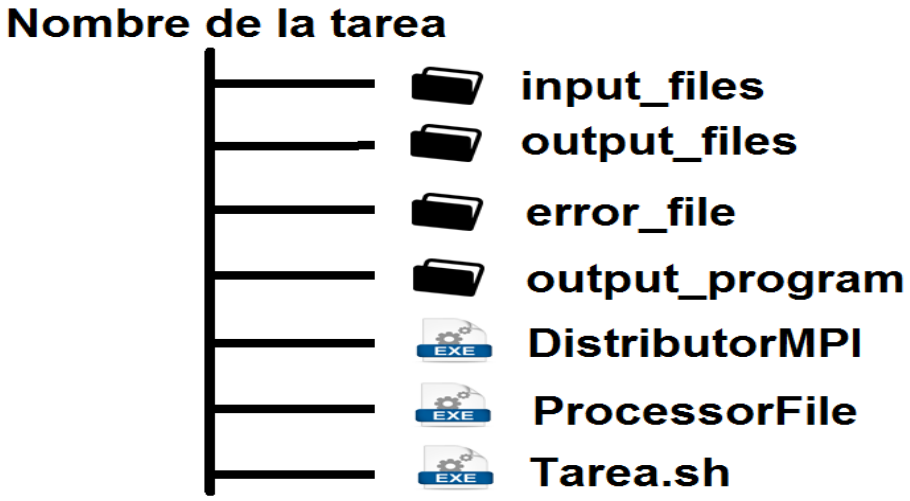


Fig 13. Estructura de directorio para almacenar los datos en el *cluster*

### *3.2.3 Ejecución del procesamiento paralelo de los ficheros*

Con el propósito de obtener un conjunto de ficheros procesados a partir de otros, se copian hacia el *cluster* los ficheros a procesar, el *DistributorMPI,* el *ProcessorFile* y el *script* Tarea.sh en lenguaje *bash.* Para ejecutar el *script* construido se utiliza el comando *qsub nombre_script_construido.sh,* entonces la herramienta de administración de recursos Torque, mediante una prioridad establecida por el administrador del *clúster* procederá a ejecutar la tarea.

Una vez que es ejecutado el *DistributorMPI* con una cantidad de nodos *n* o una cantidad *n* de procesos (preferiblemente igual a la cantidad de nodos disponibles en el *cluster*) el proceso maestro (aquel con rango 0) carga todas las direcciones de los ficheros a procesar. Después este envía a los demás procesos las direcciones de los ficheros y la cantidad mediante la operación de comunicación colectiva *MPI_Bcast*. Luego cada proceso procede a procesar los ficheros mediante el *ProcessorFile* de la siguiente forma: inicialmente procesa el fichero *i*, donde *i* inicialmente es el rango del proceso y va cambiando de la siguiente forma: *i = i + cantidad_de_procesos* mientras no sobrepase la cantidad de ficheros a procesar; con lo que se consigue que cada proceso realice las transformaciones de un grupo de ficheros a través del *ProcessorFile* proporcionado por el usuario. En la Figura 14 se muestra un ejemplo de una distribución de los ficheros a procesar por cada uno de los procesos (suponiendo que el usuario eligió dentro de su configuración 4 nodos y 2 procesos por nodo para ejecutar su tarea).

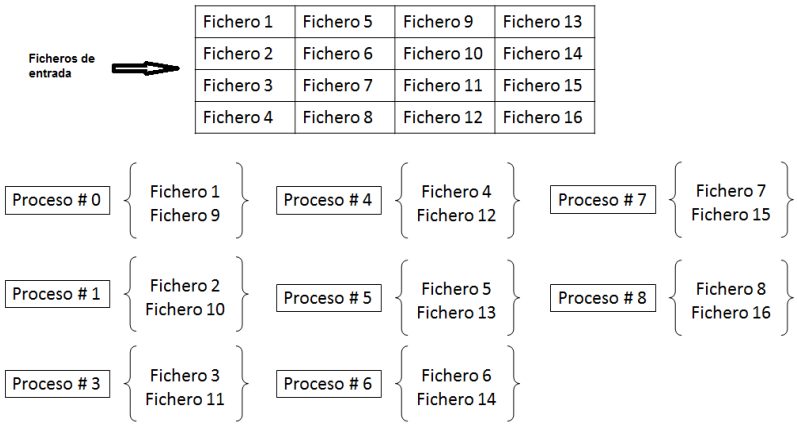


Fig 14.Ejemplo de distribución del procesamiento de los ficheros para 8 procesos

### *3.2.4 Implementación del recibimiento de los ficheros procesados*

Una vez que haya sido enviada la tarea al *cluster*, se debe comprobar la correcta finalización de la misma, comprobando que se hayan generado el fichero de error y de salida correspondientes. Después de validar

la correcta ejecución de la tarea (haya procesado los ficheros) se copian los ficheros procesados ubicados en el directorio **output_files** en el *cluster* hacia el Visor de Terrenos para su posterior utilización.

```
#!/bin/bash
#PBS -N ProcessorRaster
#PBS -l nodes=2:ppn=4
#PBS -l mem=1536Mb
#PBS -e /home/yurian/ProcessorRaster/error_file/ProcessorRaster.err
#PBS -o /home/yurian/ProcessorRaster/output_program/ProcessorRaster.out
#PBS -q cemcbatch

mpirun /home/yurian/ProcessorRaster/DistributorMPI /home/yurian/ProcessorRaster/ProcessRaster
        /home/yurian/ProcessorRaster/input_files /home/yurian/ProcessorRaster/output_files
```

Fig 15. Ejemplo de un script para ser ejecutado en Torque.

## 3.3 Diagrama de componentes

Un diagrama de componente es una parte física de un sistema (módulo, base de datos, programa ejecutable, bibliotecas, entre otros). Se puede decir que un componente es la materialización de las funcionalidades que debe cumplir un software y las relaciones que existen con otros componentes para dar solución a un problema existente. Los paquetes generados (ver Figura 16) en la presente solución son los siguientes:

**Administración de recursos**: Contiene todos los componentes referentes para realizar el procesamiento paralelo de cualquier aplicación que haya elegido la estrategia de paralelización de descomposición de dominio.

- ResourceManager: Este componente define una plantilla para todo el software administrador de recursos en *clusters Beowulf*.
- TorqueResourceManager: Componente que implementa la comunicación con el administrador de recursos Torque, el cual permite ejecutar tareas en paralelo.

**Estructura de datos**: Contiene todos los componentes necesarios para la creación de tareas a ejecutar en *clusers*.

- Job: Componente encargado de almacenar los parámetros de configuración de la tarea a ser ejecutada en el *cluster.*

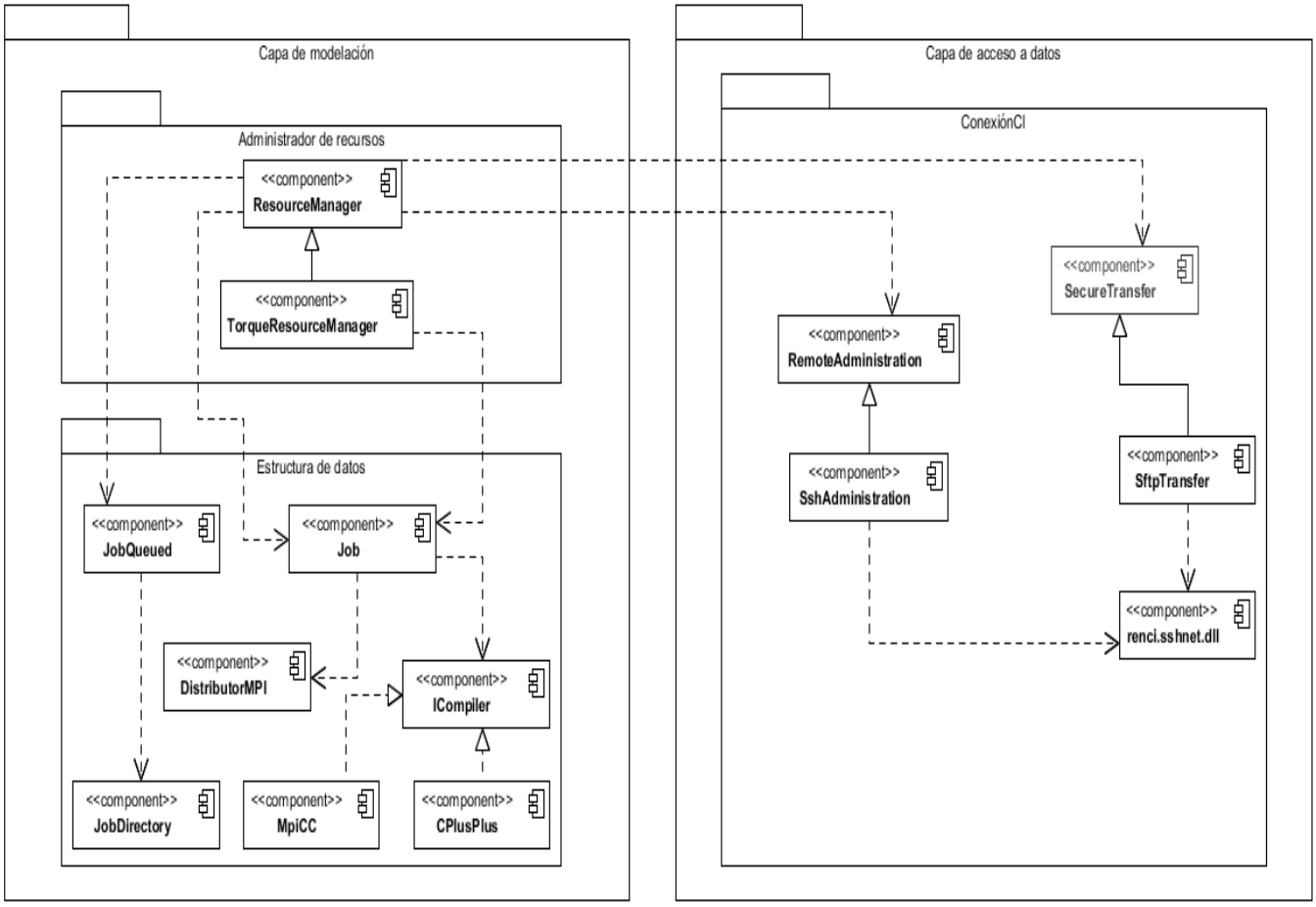


Fig 16: Diagrama de componentes de la biblioteca

- DistributorMPI: Componente que representa a el programa ejecutable encargado de distribuir el procesamiento de los datos mediante un programa ejecutable proporcionado por el usuario; ambos tienen que estar escritos con el estándar C11 del lenguaje de programación C/C++ y compilados en alguna distribución GNU/Linux.
- ICompiler: Este componente define una plantilla para los compiladores de lenguajes de programación.
- MpiCC: Este componente define el comando a utilizar para compilar un programa que haga uso de estándar MPI.
- CPlusCplus: Este componente define el comando a utilizar para compilar un programa escrito en C++.

- JobDirectory: Este componente almacena la estructura de directorio a utilizar para cada tarea a ejecutar en el *cluster*.
- JobQueued: Componente que representa a una tarea que ha sido encolada en alguna de las colas definidas en el *cluster*.

**ConexiónCl**: Contiene todos los componentes necesarios para establecer una conexión con un *cluster*.

- SecureTransfer: Este componente define una plantilla para cualquier protocolo que pueda enviar y recibir datos de forma segura hacia un host remoto.
- SftpTransfer: Componente que implementa la comunicación con un host remoto para enviar y recibir datos hacia un host remoto a través del protocolo sftp.
- RemoteAdministration: Este componente define una plantilla para la ejecución de comandos en un host remoto.
- SshAdministration: Contiene todo lo necesario para establecer una conexión a través del protocolo SSH-2, permitiendo ejecutar comandos remotos.
- renci.sshnet.dll: Es una biblioteca de vínculos dinámicos que permite establecer una conexión vía SSH-2 con un servidor remoto, permitiendo enviar y recibir datos entre el cliente y el servidor de forma segura. En esta se encuentran las clases SshClient y SftpClient entre otras.

## 3.4 Pruebas realizadas a la biblioteca

Las pruebas muestran que un programa hace lo que debe hacer y descubrir los defectos del mismo antes de que sea puesto en vigor. Al realizar la prueba de software, se ejecuta un programa a partir de datos artificiales. Se comprueba los resultados de la prueba de errores, anomalías, o información acerca de los atributos no funcionales del programa (Sommerville, 2011).

(Sommerville, 2011) dice que el proceso de prueba tiene dos objetivos distintos:

1. Demostrar al desarrollador y al cliente que el software cumple los requerimientos. Para el software personalizado, esto significa que debe haber al menos una prueba para todos los requerimientos en el documento de requerimientos. Para productos de software genéricos, significa que debe haber pruebas para todas las funciones del sistema, más combinaciones de estas funciones, que serán incorporadas en la liberación del producto.
2. Descubrir situaciones en las cuales el software se comporta de manera incorrecta, no deseada o no cumple con sus especificaciones. Estas son una consecuencia de defectos en el software. Las pruebas de defectos se refieren a erradicar comportamientos indeseables en el sistema, tales como

fallo del sistema, las interacciones no deseadas con otros sistemas, cálculos incorrectos y la corrupción de datos.

(Pressman, 2014) define cuatro niveles de abstracción para las pruebas realizadas al software durante el desarrollo del mismo.

1. Pruebas unitarias, donde unidades individuales del programa o clases de objetos son probadas.
2. Pruebas de Integración, donde varias unidades son integradas para crear componentes compuestos, esta se centra en el diseño y en la arquitectura del sistema.
3. Pruebas de validación, donde los requisitos establecidos durante el diseño son validados.
4. Pruebas de sistema, en esta el software y los otros elementos del sistema se prueban como un todo.

Entre los métodos de pruebas más utilizados se encuentran las pruebas de caja blanca. Estas pruebas se basan en un examen minucioso de los detalles de procedimientos, y son una filosofía para el diseño de juego de datos centrados en los detalles de codificación. Las rutas lógicas a través del software y colaboraciones entre los componentes son probadas mediante el ejercicio de un conjunto específico de condiciones y/o bucles. Entre las técnicas utilizadas para las pruebas de caja blanca se encuentra la técnica del camino básico, esta permite obtener una medida de la complejidad lógica del diseño de los procedimientos y utilizar esta medida como guía para derivar un conjunto de caminos de ejecución, los juegos de datos diseñados con el propósito de ejecutar dichos caminos garantizan la ejecución de cada sentencia en el programa al menos una vez (Pressman, 2014). En la presente investigación se realizaron pruebas unitarias utilizando el método de pruebas de caja blanca a través de la técnica del camino básico.

### *3.4.1 Aplicación de las pruebas unitarias.*

La prueba centra el proceso de verificación en la menor unidad de diseño del software. En el caso de un contexto orientado a objetos el concepto de unidad cambia. La menor unidad que se prueba es la clase (Pressman, 2014). La prueba de las clases es dirigida por las operaciones que esta encapsula y el estado de comportamiento de las mismas. A continuación, se describe una de las pruebas realizadas el método público *RunJob* mostrado en la Figura 17 de la clase TorqueResourceManage*r*, asociado al requisito funcional ‘‘Ejecutar tarea’’. Este método es responsable de ejecutar una tarea con las configuraciones proporcionadas por el usuario en un *cluster Beowulf* a través del software Torque instalado en el *cluster*. Una vez enviada una tarea al software Torque; mediante la ejecución del comando “qstat -a” en el nodo del

*cluster* habilitado para encolar tareas; debe aparecer información de la misma: como el nombre que le haya dado el usuario, el estado en el cual se encuentra, los recursos que solicitó para la misma, entre otras informaciones de interés.

```
1  public override void RunJob(Job job)
2  {
3      JobDirectory jobDirectory = CreateDirectory(job);
4
5      string compiledProcessorFileName;
6      string compiledProgramMpiName;
7      secureTransfer.Upload(new DirectoryInfo(job.InputFiles), jobDirectory.PathInputFiles);
8
9      if (job.Build)
10     {
11         CompileSourceCodes(job, jobDirectory, out compiledProcessorFileName, out compiledProgramMpiName);
12         remoteAdministrator.RunCommand("chmod 0700 " + compiledProcessorFileName);
13         remoteAdministrator.RunCommand("chmod 0700 " + compiledProgramMpiName);
14     }
15     else
16     {
17         secureTransfer.Upload(new FileInfo(job.ProgramMPICompiled), jobDirectory.RootPath);
18         secureTransfer.Upload(new FileInfo(job.ExecProcessorFile), jobDirectory.RootPath);
19         compiledProcessorFileName = jobDirectory.RootPath + "/" + Path.GetFileName(job.ExecProcessorFile);
20         compiledProgramMpiName = jobDirectory.RootPath + "/" + Path.GetFileName(job.ProgramMPICompiled);
21         secureTransfer.ChangePermission(compiledProcessorFileName, 0700);
22         secureTransfer.ChangePermission(compiledProgramMpiName, 0700);
23     }
24
25     ExecuteQueueSub(job, jobDirectory, compiledProcessorFileName, compiledProgramMpiName);
26 }
```

(1) (2) (3) (4) (5)

Fig 17. Método *RunJob* asociado al requisito funcional ‘‘Ejecutar tarea’’

En la Figura 18 se muestra el grafo de flujo correspondiente al método *RunJob* mencionado anteriormente. La cantidad de juegos de datos necesarios para ejecutar todas las sentencias posibles es al menos la complejidad ciclomática del grafo del flujo. El valor de la complejidad ciclomática de V(G) del grafo de flujo G se obtiene mediante la ecuación:

(1) $V(G) = E – N + 2$

donde *E* y *N* definen la cantidad de aristas y la cantidad de nodos respectivamente de *G.*

Utilizando la ecuación 1 se obtuvo como resultado que la complejidad ciclomática del grafo de flujo del método *RunJob* es dos. Con este propósito se diseñaron dos juegos de datos para cubrir todos los caminos posibles en el grafo.

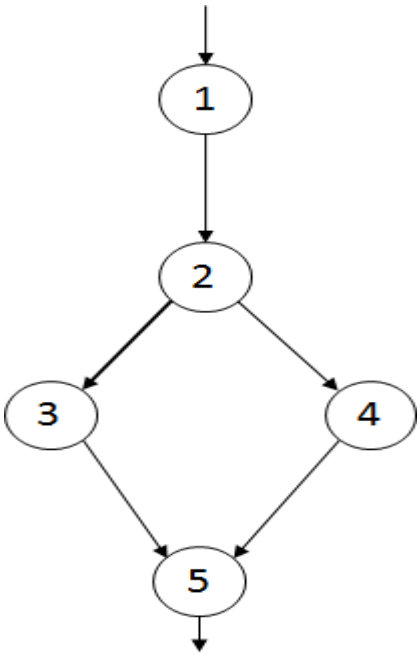


Fig 18. Grafo de flujo del método *RunJob.*

En la Tabla 7 se reflejan los 2 caminos básicos posibles dado la complejidad ciclomática obtenida.

Tabla 2. Caminos del grafo de flujo del método *RunJob.*

| No | Camino |
|---|---|
| 1 | 1-2-3-5 |
| 2 | 1-2-4-5 |

Luego de la obtención de la cantidad de caminos básicos del grado de flujo se procedió a la confección de los juegos de datos. La Tabla 3 muestra el juego de datos correspondientes al segundo camino básico. El resto de los casos pueden ser consultados en los anexos del documento. Las configuraciones de la tarea son válidas para el *cluster Beowulf* descrito en la sección de Validación de la biblioteca.

Tabla 3. Caminos del grafo de flujo

| Camino | Descripción | Entrada | Resultados esperados |
|---|---|---|---|

| | | | |
|---|---|---|---|
| 1-2-4-5 | Recibe una tarea a ejecutar con una determinada configuración. Entre las configuraciones de la tarea no se especifica que se quiera compilar el código fuente de la aplicación que procesa un fichero y da como resultado otro. Esta aplicación ya se encuentra compilada para ser ejecutada en cualquier distribución de GNU/Linux. Se indica la dirección de la aplicación compilada. | name = "ProcessorRaster"<br>queueName = "cemcbatch"<br>errorFileName = "ProcessorRaster.err"<br>outputFileName = "ProcessorRaster.out"<br>memory = "1536Mb"<br>nodes = "3"<br>pNodes = "2"<br>execProcessorFile = "@"D:\ProcessorRaster"<br>inputFiles = "@"D:\input_files" | Una vez ejecutado el comando "qstat -a" en el nodo que puede encolar tareas; se observa toda la información respecto a la tarea. |

Se realizaron cuatro iteraciones de las pruebas unitarias. Al finalizar cada iteración se procedió a la corrección de las no conformidades. En el proceso de ejecutar una tarea del usuario en el *cluster* se detectaron cuatro no conformidades de las cuales fueron resueltas el 50% al finalizar la primera iteración y el resto en la segunda iteración.

## 3.5 Validación de la biblioteca

Con el objetivo de realizar una comparación con respecto al rendimiento del modelo BDAM en el procesamiento de los datos utilizando técnicas de programación paralela en un *cluster Beowulf* y procesamiento secuencial en una computadora multi-core, se realizaron un conjunto de experimentos. La actual implementación del procesamiento de los datos del Visor de Terrenos utilizando el modelo BDAM se realiza de forma secuencial, o sea, se procesa un Modelo Digital de Elevación (MDE) a la vez. En la Tabla 4 se muestra el nombre, la cantidad de ficheros y capacidad de los juegos de datos utilizados, estos contienen información variada de los relieves de distintas porciones de la superficie terrestre. La variable capacidad define el espacio (en Megabyte) que ocupa el conjunto de ficheros en disco.

Tabla 4. Fuente de datos utilizados en los experimentos.

| Nombre | Cantidad de ficheros (capas) | Capacidad (sin compresión) |
|---|---|---|
| Monay Results | 62 | 62 MB |

### *3.5.1 Métricas de rendimiento*

Las métricas son muy útiles en la evaluación de algoritmos paralelos, estas facilitan una indicación de cuán bien está paralelizado el algoritmo o cuán bien este realiza las tareas en paralelo. A continuación, se resume según (Grama et al., 2003) las métricas más utilizadas para evaluar el rendimiento de algoritmos paralelos.

**Tiempo de ejecución**

El tiempo de ejecución *Tp* de un algoritmo paralelo con *p* procesadores es el tiempo que transcurre desde el momento en el que el primer procesador comienza la ejecución del programa hasta que el último procesador culmina. El mismo está dado por la siguiente razón:

$$(2)\ Tp = Ta + Tc$$

donde *Ta* es el tiempo de procesamiento de la aplicación (el cual puede estar condicionado por los datos a procesar) y *Tc* es el tiempo de comunicación entre los procesadores. El tiempo de comunicación no suele tomarse en cuenta cuando el intercambio de datos entre los procesadores es prácticamente nulo.

**Ganancia de velocidad**

La ganancia de velocidad (*Speed-Up* por su traducción al inglés) compara el tiempo secuencial con el tiempo paralelo para resolver un problema en particular, o sea, expresa cuánto se ha reducido el tiempo de procesamiento al ejecutar el algoritmo en *p* procesadores. El *speed-up* se calcula a través de la siguiente razón:

$$(3)\ Sp = \frac{Ts}{Tp}$$

*Ts*: es el tiempo del mejor algoritmo secuencial.
*Tp*: es el tiempo del algoritmo paralelo.

$$(4)\ \lim_{n \to \infty} Sp = p$$

Si se considera que, tomando como base que el tiempo de un algoritmo paralelo *Tp* ejecutado en *p* procesadores es, como máximo, *p* veces inferior al tiempo secuencial *Ts* del algoritmo, entonces el límite

del *speed-up Sp* está definido según la ecuación 4, donde *p* es el óptimo teórico, el cual es el más deseado cuando se implementa la versión paralela de un algoritmo secuencial.

**Eficiecia**

La eficiencia está relacionada con el *speed-up,* normaliza su valor teniendo en cuenta el número de procesadores. Esta se refiere al porcentaje de aprovechamiento de los procesadores empleados en la ejecución del algoritmo, y se expresa mediante el cociente de *Sp* y *p.*

$$(4)\ Ep = \frac{Sp}{p}$$

### *3.5.1 Experimentación en el procesamiento secuencial del modelo BDAM.*

En los experimentos para el procesamiento secuencial se utilizaron dos configuraciones de hardware para medir los tiempos de CPU. En la Tabla 5 se muestran las principales características de cada hardware.

Tabla 5. Características de las configuraciones de hardware para procesamiento secuencial

| Características | Hardware (H-1) | Hardware (H-2) |
|---|---|---|
| Procesador | Intel(R) Core(TM) i3-4160 | Intel(R) Core(TM) i3-4130 |
| RAM | 4096 MB | 4096 MB |
| Velocidad | 3.60 GHz | 3.40 GHz |
| Núcleos CPU | (2 físicos y 2 lógicos) | (2 físicos y 2 lógicos) |

La descripción de la ‘’fuente de datos’’ utilizada para la experimentación se muestra en la Tabla 4. En la Tabla 6 se muestra los tiempos (en segundos) de CPU obtenidos en la ejecución del procesamiento secuencial en las configuraciones de hardware **H-1** y **H-2**.

Tabla 6. Tiempo de procesamiento del modelo BDAM en cada hardware para cada fuente de datos.

| Configuración | Fuente de datos | Tiempo (s) |
|---|---|---|
| H - 1 | Monay_Results | 1160.46 |
| H - 2 | Monay_Results | 1210.36 |

La gráfica de la Figura 19 muestra los resultados obtenidos en la Tabla 6, donde se evidencia las mejoras del tiempo de procesamiento en dependencia de las capacidades de cómputo.

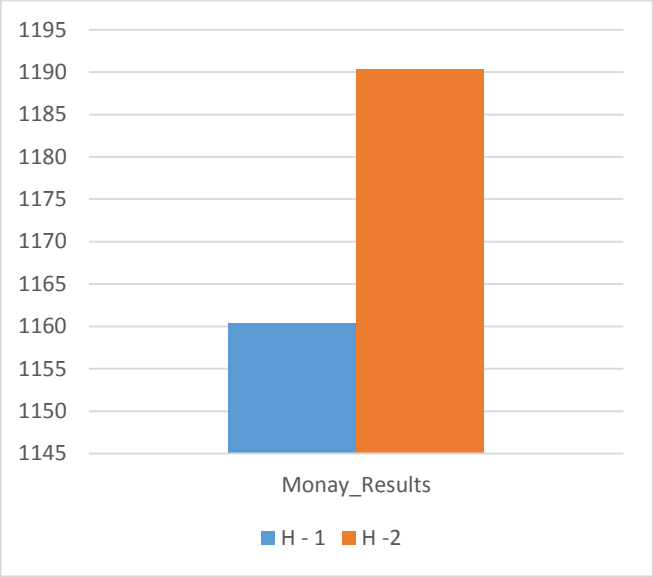


Fig 19. Comparación de los tiempos de procesamiento del modelo BDAM en las configuraciones de hardware H-1 y H-2 en su versión secuencial.

### *3.5.1 Experimentación en el procesamiento paralelo/distribuido del modelo BDAM.*

Para realizar los experimentos en el procesamiento paralelo/distribuido del modelo BDAM a través de la biblioteca se utilizó un *cluster Beowulf* conformado por cinco computadoras con las mismas prestaciones. En la Tabla 7 se muestran las características del hardware que poseen cada una de las computadoras del *cluster* y que son de interés para la validación de los resultados obtenidos. El resto de las especificaciones del *cluster* pueden ser consultadas en el Anexo C de la presente investigación.

Tabla 7. Características del hardware de cada nodo del *cluster Beowulf* (los 5 nodos tienen el mismo hardware)

| **Características** | **Hardware** |
|---|---|
| Procesador | Intel(R) Core(TM) i3-4160 |
| RAM | 4096 MB |
| Velocidad | 3.60 GHz |
| Núcleos CPU | (2 físicos y 2 lógicos) |
| Tarjeta de Red | Fast Ethernet (10/100 Mbits) |

En la Tabla 8 se muestra los tiempos de ejecución (en segundos) obtenidos en la ejecución del procesamiento paralelo/distribuido del modelo BDAM en el *cluster Beowulf* descrito anteriormente para la ‘’fuente de datos’’ Monay_Results**.**

Tabla 8. Tiempos de ejecución del procesamiento paralelo/distribuido del modelo BDAM en un cluster *Beowulf*.

| Fuente de datos | Cantidad de Procesadores / Tiempo (s) | | | | |
|---|---|---|---|---|---|
| | 2 | 4 | 6 | 8 | 10 |
| Monay_Results | 561.64 | 292.15 | 197.59 | 147.39 | 125.43 |

En la gráfica de la Figura 20 se muestra a modo de comparación los tiempos de ejecución obtenidos en el procesamiento paralelo/distribuido a través de la biblioteca desarrollada para diferentes cantidades de procesadores.

Los resultados muestran una reducción del tiempo de procesamiento a medida que se incrementa los procesadores. Esto ocurre debido al aumentado del número de nodos para computar el procesamiento paralelo del modelo BDAM. Aclarar que para el cálculo del *speed-up* se tomó como mejor tiempo el procesamiento secuencial del modelo BDAM; el obtenido del **H-1**, ya que los nodos del *cluster Beowulf* poseen estas mismas prestaciones.

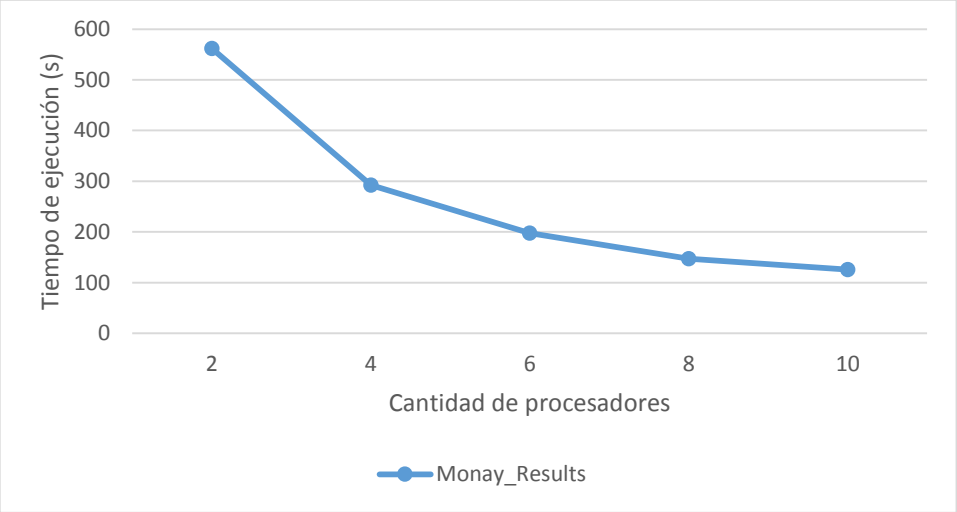


Fig 20. Comparación de los tiempos de ejecución del procesamiento paralelo/distribuido del modelo BDAM en un *cluster Beowulf.*

En la gráfica de la Figura 21 se muestra el comportamiento de ganancia de velocidad, donde se evidencia una reducción del tiempo de ejecución a medida que se incrementa la cantidad de procesadores. El caso más significativo que refleja lo dicho anteriormente es el *speed-up* obtenido para 10 procesadores, donde se llega a reducir hasta 9.25 veces el tiempo de ejecución obtenido en el procesamiento secuencial del modelo BDAM. También se observa que para 10 procesadores el *speed-up* está relativamente cerca del óptimo teórico, lo que indica que para ese caso se alcanza mayor ganancia de velocidad.

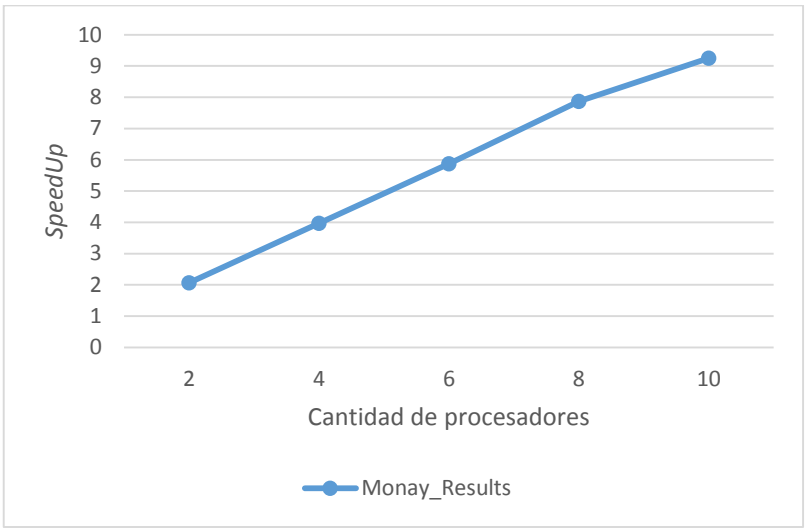


Fig 21. Comportamiento de la ganancia de velocidad del modelo BDAM en el *cluster Beowulf*.

En el procesamiento paralelo del modelo BDAM se hace necesario el empleo de las operaciones de lectura y escritura de datos en varios ficheros para almacenar los resultados. Estas operaciones pueden afectar el tiempo de ejecución total. Un ejemplo que refleja lo dicho anteriormente, sucede cuando se ejecuta el procesamiento con 10 procesadores repartidos en cinco nodos computacionales (2 procesos MPI por nodo), obteniéndose una eficiencia de 92%, siendo este resultado inferior a los obtenidos con menos procesadores repartidos en menos nodos. Esto se debe a que el tiempo de comunicación entre los nodos dedicados a computar tareas y el nodo dedicado a compartir archivos puede aumentar según la cantidad de nodos, afectando así el tiempo total. No obstante una eficiencia de 92% es aceptable y muestra claramente resultados favorables en los experimentos realizados en el procesamiento paralelo del modelo BDAM.

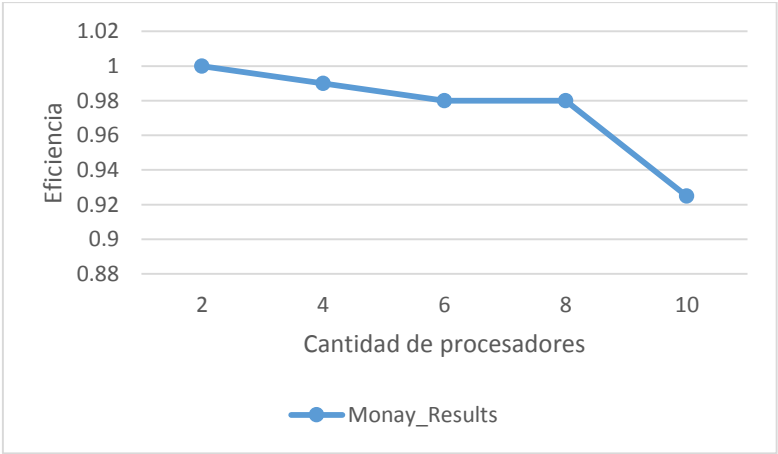


Fig 22. Comportamiento de la eficiencia del procesamiento del modelo BDAM en un *cluster Beowulf*

En la gráfica de la Figura 23 se presenta a modo de comparación el tiempo de procesamiento de los datos del modelo BDAM utilizando un *cluster Beowulf* y el procesamiento secuencial realizado sobre una computadora multi-core. Los resultados obtenidos evidencian la disminución del tiempo de ejecución en el procesamiento de datos haciendo uso de técnicas de programación paralela sobre un *cluster Beowulf*.

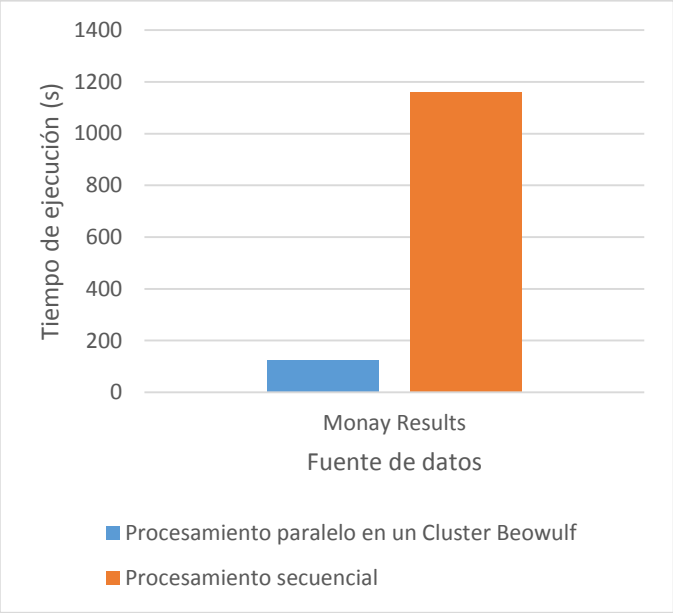


Fig 23: Comparación del tiempo de ejecución obtenido por ambos procesamientos

## Conclusiones del capítulo

En este capítulo se definieron los estándares de codificación para .Net, logrando así una implementación legible y de fácil mantenimiento. Fueron mostrados los principales detalles de implementación de la ejecución de una tarea en un *cluster Beowulf.* El diagrama de componentes sirvió de apoyo para representar cómo la solución está dividida en componentes y cómo se relacionan entre ellos. Se realizaron diversos experimentos donde se comparó el rendimiento en el procesamiento de datos del modelo BDAM haciendo uso de técnicas de programación paralela en un *cluster Beowulf* y el procesamiento secuencial de los datos en una computadora multi-core, comprobando efectivamente que, a medida que se aumenta la cantidad de procesadores se obtienen mejores tiempos de ejecución. Además se realizaron las pruebas unitarias, las cuales sirvieron para la validación de la biblioteca, demostrando la validez de la hipótesis planteada en la introducción del presente trabajo.

## Conclusiones

Se desarrolló una biblioteca que permite ejecutar tareas de procesamiento de datos en un *cluster Beowulf,* la cual fue integrada al Visor de Terrenos donde se evidenciaron los siguientes resultados:

- La implementación de los mecanismos de comunicación de la biblioteca implementada con un *cluster Beowulf* a través del protocolo SSH-2 permitió dotar a la biblioteca de seguridad en el envío y recepción de los datos.
- La estrategia de paralelización por datos presente en el diseño de la biblioteca permite aprovechar las capacidades de escalabilidad de un *cluster Beowulf* para mejorar su rendimiento computacional.
- La distribución equitativa del procesamiento de los datos con igual tamaño permite obtener un rendimiento elevado en un *cluster Beowulf* conformado por nodos con una arquitectura de cómputo homogénea.
- La forma en que se diseñó e implementó la solución permite que pueda ser aplicada a otros tipos de procesamiento de datos.
- La biblioteca desarrollada permite realizar el procesamiento de datos en arquitecturas paralelas de tipo heterogénea y dedicada, donde es posible obtener resultados favorables en cuanto a ganancia de velocidad, demostrando así las capacidades de cómputo que se pueden alcanzar a través de computadoras interconectadas mediante una red local.
- La biblioteca desarrollada redujo el tiempo de ejecución del modelo BDAM en el procesamiento de los datos en el Visor de Terrenos.

## Recomendaciones

- Utilizar técnicas de compresión de datos para el envío y recepción eficiente de los ficheros a través de la biblioteca.
- Agregar a la biblioteca desarrollada la posibilidad de ejecutar tareas de procesamiento de datos a través de otros planificadores de tareas en sistemas distribuidos.

## Referencias Bibliográficas

Almasi, G. S.; Gottlieb, A. Highly Parallel Computing. **1988**.

Balduino, R. Introduction to OpenUP (Open Unified Process). *Eclipse site* **2007**.

Barrett, D. J.; Silverman, R. E. *SSH, the Secure Shell: The Definitive Guide*; O'Reilly Media, Inc., 2001.

Blanco de Frutos, H. Clusterización de Aplicaciones Paralelas Para Su Planificación En Entornos de Cómputo Multi-Cluster. **2012**.

Buyya, R. High Performance Cluster Computing: Architectures and Systems (volume 1). *Prentice Hall, Upper SaddleRiver, NJ, USA* **1999**, *1*, 999.

Carri, M. C. L. Análisis de Performance Y Optimización En Cluster Beowulf, Tese de Mestrado, Universidade de Buenos Aires, 2004.

Cignoni, P.; Ganovelli, F.; Gobbetti, E.; Marton, F.; Ponchio, F.; Scopigno, R. BDAM—Batched Dynamic Adaptive Meshes for High Performance Terrain Visualization. In *Computer Graphics Forum*; Wiley Online Library, 2003; Vol. 22, pp 505–514.

De Floriani, L.; Kobbelt, L.; Puppo, E. A Survey on Data Structures for Level-of-Detail Models. In *Advances in multiresolution for geometric modelling*; Springer, 2005; pp 49–74.

Dongarra, J. J.; Foster, I.; Fox, G.; Gropp, W.; Kennedy, K.; Torczon, L.; White, A. *Sourcebook of Parallel Computing*; Morgan Kaufmann Publishers San Francisco, 2003; Vol. 3003.

Fraire, J. A.; Ferreyra, P.; Marques, C. OpenCL Overview, Implementation, and Performance Comparison. *Latin America Transactions, IEEE (Revista IEEE America Latina)* **2013**, *11* (1), 274–280.

Gove, D. *Multicore Application Programming: For Windows, Linux, and Oracle Solaris*; Addison-Wesley Professional, 2010.

Grama, A.; Gupta, A.; Karypis, G.; Kumar, V. Introduction to Parallel Computing. *Introduction to Parallel Computing, 2nd edn, by A. Grama et al. Pearson Education Limited, Harlow, England (ISBN: 978-0-201-64865-2)* **2003**, *1*.

Hansen, P. B. Model Programs for Computational Science: A Programming Methodology for Multicomputers. *Concurrency: practice and experience* **1993**, *5* (5), 407–423.

Hunt, L. Coding Standards for .NET. **2007**.

Larman, C. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*; Pearson Education India, 2005.

Lindstrom, L.; Jeffries, R. Extreme Programming and Agile Software Development Methodologies. *Information systems management* **2004**, *21* (3), 41–52.

Li, Z.; Zhu, C.; Gold, C. *Digital Terrain Modeling: Principles and Methodology*; CRC press, 2004.
Márquez Pérez, C. D.; Sorribes Gomis, J.; César Galobardes, E. Factores de Rendimiento Asociados a SPMD. **2011**.

Martínez, P. Infraestructura Para Computación de Alta Disponibilidad Y Administración de Recursos Mediante Condor, Facultad de Informática, 2014.

McBryan, O. A. An Overview of Message Passing Environments. *Parallel Computing* **1994**, *20* (4), 417–444.

Montero Montero, R. Desarrollo de Una Interfaz En Silverlight Para La Gestión de Un Sistema de Colas En Clusters. **2009**.

Pajarola, R.; Gobbetti, E. Survey of Semi-Regular Multiresolution Models for Interactive Terrain Rendering. *The Visual Computer* **2007**, *23* (8), 583–605.

Peterson, L. L.; Davie, B. S. Computer Networks: A Systems Approach. **2011**.

Pressman, R. S. *Software Engineering: A Practitioner's Approach*; Palgrave Macmillan, 2014.

Solar, R.; Suppi, R.; Luque, E. High Performance Distributed Cluster-Based Individual-Oriented Fish School Simulation. *Procedia Computer Science* **2011**, *4*, 76–85.

Sommerville, I. *Software Engineering Ninth Edition*; Addison-Wesley, 2011.

SPARSKIT, Y. S. *A Basic Tool Kit for Sparse Matrix Computations-Version 2*; Tech. Rep. Computer Science Department, Univ. of Minnesota, Minneapolis, MN, 1994.

Sterling, T. L. *Beowulf Cluster Computing with Linux*; MIT press, 2002.

Van der Steen, A. J.; Dongarra, J. J. *Overview of Recent Supercomputers*; Citeseer, 1995.

Vlissides, J.; Helm, R.; Johnson, R.; Gamma, E. Design Patterns: Elements of Reusable Object-Oriented Software. *Reading: Addison-Wesley* **1995**, *49* (120), 11.

White, T. *Hadoop: The Definitive Guide*; O'Reilly Media, Inc., 2012.

Yadav, R. *Spark Cookbook*; Packt Publishing Ltd, 2015.

Apache Flink: Scalable Batch and Stream Data Processing https://flink.apache.org/ (accessed May 11, 2016).

Message Passing Interface (MPI) Forum Home Page http://www.mpi-forum.org/ (accessed May 11, 2016).

# Anexos

## Anexo A: Juego de datos para el método *RunJob.*

Tabla 9. Juego de datos para el método RunJob

| Camino | Descripción | Entrada | Resultados esperados |
|---|---|---|---|
| 1-2-3-5 | Recibe una tarea a ejecutar con una determinada configuración. Entre las configuraciones de la tarea se especifica que se quiera compilar el código fuente de la aplicación que procesa un fichero y da como resultado otro. Este código debe estar escrito empleando el estándar C11 de C++. | name = “ProcessorRaster”<br>queueName = “cemcbatch”<br>errorFileName = “ProcessorRaster.err”<br>outputFileName = “ProcessorRaster.out”<br>memory = “1536Mb”<br>nodes = “3”<br>pNodes = “2”<br>execProcessorFile = “@"D:\ProcessorRaster.cpp”<br>inputFiles = “@"D:\input_files” | Una vez ejecutado el comando “qstat -a” en el nodo que puede encolar tareas; se observa toda la información respecto a la misma. |

## Anexo B: Descripciones de las clases del diseño

**ICompiler:** La implementación de esta interfaz permite obtener la forma o comando de un compilador para compilar un código fuente-

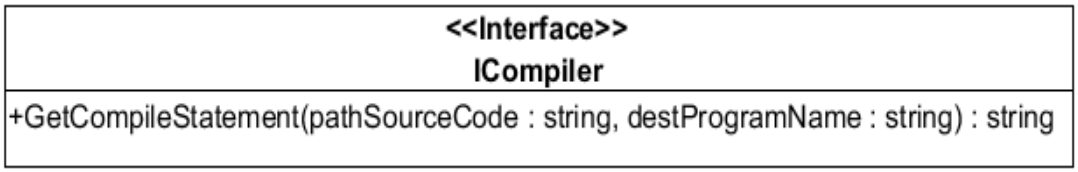


Fig 24. Interfaz ICompiler.

**CPlusPlusCompiler:** Esta clase implementa la interfaz ICompiler, permitiendo obtener el comando para compilar un programa escrito en C/C++.

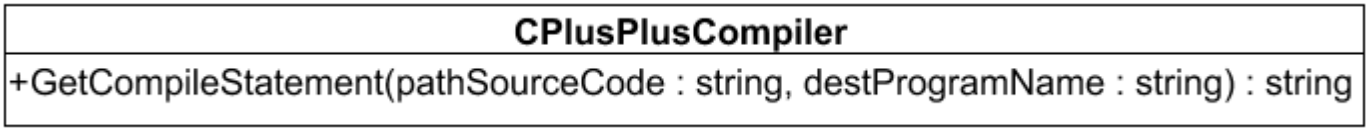


Fig 25. Clase CPlusPlusCompiler

**MpiCCCompiler:** Clase que implementa la interfaz ICompiler, permitiendo obtener el comando para compilar un programa que utilice una biblioteca que implemente algún estándar de MPI.

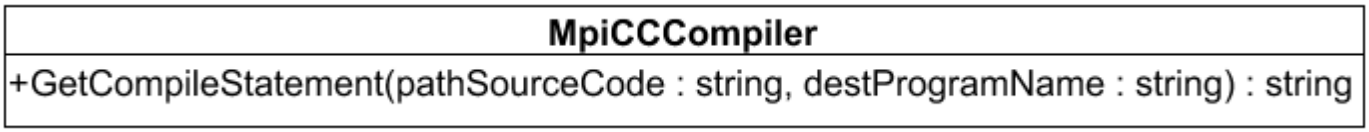


Fig 26. Clase MpiCCCompiler

**JobDirectory:** Clase que permite representar la estructura de directorio creada en un *cluster Beowulf* para una tarea.

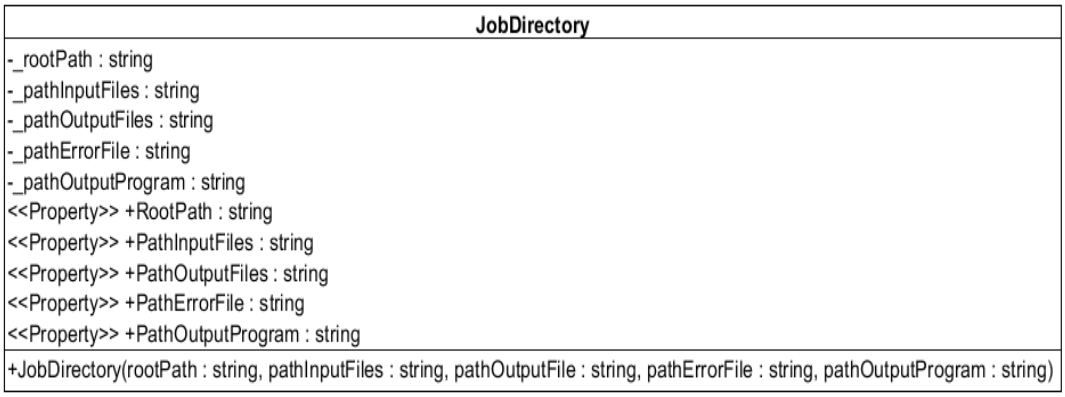


Fig 27. Clase JobDirectory

**JobQueued:** Esta clase guarda el id, estructura de directorio creada y todas las configuraciones de una tarea que haya sido enviada a ejecución.

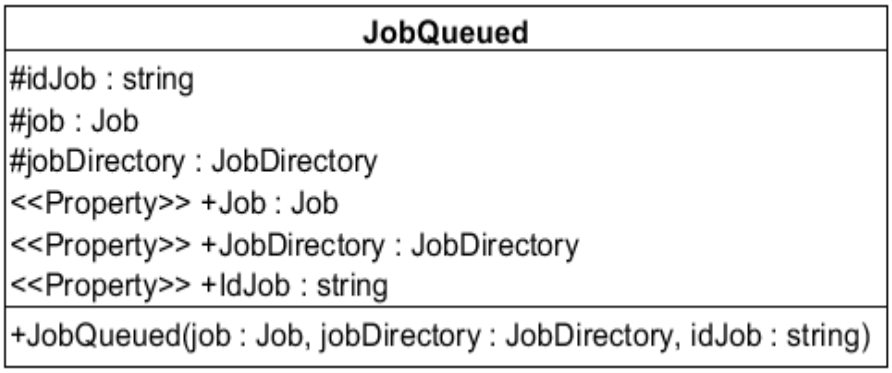


Fig 28. Clase JobQueued

## Anexo C: Descripciones del *Cluster Beowulf.*

Cada computadora tiene como Sistema Operativo Debian-7.5.0-amd64-netinst y son identificadas mediante los siguientes *hostname*: *server*, node1, node2, node3, node4. En todos los nodos (incluyendo el nodo *server*) se encuentran instalados los compiladores g++ y gcc; ambos en su versión 4.9.2, el software MPICH2 en su versión 3.1, el cual está compilado para trabajar de forma nativa con el software Torque. En el *server* está instalado el servidor y el cliente del software Torque 4.2.9: en los nodos solamente el cliente. El *server* es la única computadora habilitada para encolar tareas a la cola con nombre “cemcbatch” previamente creada en el servidor del Torque. Además en todos los nodos se encuentra instalado el software OpenSSH en su versión 6.7p1 el cual se utiliza para establecer conexiones a través del protocolo ssh entre los nodos que forman el *cluster*. En el nodo con *hostname server* además de estar en ejecución un servidor ssh, también se encuentra en ejecución un servidor sftp, el cual permite que los usuarios transfieran sus datos hacia el *cluster* de forma segura.