CECAL

Instituto de Computación, Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

PROYECTO DE GRADO INGENIERÍA EN COMPUTACIÓN

Cloud Computing para develar el desarrollo embrionario

Lucía Marroig y Camila Riverón
lulimarroig@gmail.com, camiriveron@gmail.com

Marzo de 2015

Tutor de Proyecto:
Sergio Nesmachnow, Universidad de la República.

Cloud Computing para develar el desarrollo embrionario
Marroig, Lucía., Riverón, Camila
Proyecto de Grado
CECAL
Instituto de Computación - Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay, marzo de 2015

Índice general

Resumen

Las técnicas de microscopia por fluorescencia y la capacidad de etiquetar proteínas en ambientes celulares han marcado un antes y un después en la forma de estudiar las células. La espectroscopía de correlación de fluorescencia y las técnicas relacionadas son extremadamente útiles para medir cuantitativamente el movimiento de moléculas en células vivas. Este proyecto presenta la aplicación de técnicas y paradigmas de computación científica de alto desempeño para complementar el análisis de las fluctuaciones de fluorescencia por medio de simulaciones estocásticas. En el contexto del proyecto se propone y se desarrolla una aplicación en una infraestructura cloud de computación distribuida para ejecutar simulaciones que representan los complejos procesos biológicos celulares, mediante el diseño de una arquitectura altamente escalable y accesible a los usuarios. Se desarrolla un algoritmo MapReduce en un clúster en el cloud utilizando la tecnología Microsoft Azure. El algoritmo permite la ejecución paralela de diversas simulaciones, independizando la ejecución de los recursos de cómputo locales de los usuarios. La evaluación experimental demuestra la correctitud de la implementación desarrollada y su utilidad como herramienta de computación científica en el cloud.

Este proyecto ha recibido el apoyo del programa “Microsoft Azure for Research Award program” (2014)

Capítulo 1

Introducción

En la actualidad, es cada vez más común la utilización de software y recursos informáticos para resolver problemas de diversa índole y características. Una de las áreas en las que la implantación de sistemas computacionales ha cobrado mayor importancia a lo largo de los últimos años es en el desarrollo de proyectos científicos. La evolución tecnológica de las distintas disciplinas científico-técnicas ha pasado de centrarse en el perfeccionamiento de los instrumentos de detección (incluyendo mejores sistemas de secuenciación en biotecnología, mejores detectores de partículas en física de altas energías, etc.) a preocuparse por la capacidad de analizar grandes volúmenes de datos producidos por instrumentos cada vez más capaces y veloces.

El desarrollo científico que se ha alcanzado en los últimos años ha sido posible gracias al avance tecnológico e investigación para cada una de las diversas disciplinas. Sin embargo, dichos progresos no resultan tan significativos si no se dispone de herramientas que permitan almacenar, procesar, visualizar y comprender la gran cantidad de datos producidos. A la vez, esta evolución ha hecho que las instalaciones científicas hayan aumentado su complejidad intrínseca, trasladando la complejidad de las partes al control del todo. Esta situación hace inevitable la utilización de técnicas altamente avanzadas procedentes de la ingeniería de software que permitan abstraer parcialmente la complejidad de dichos sistemas.

Actualmente existe una creciente tendencia en tecnologías de la información y computación científica, a la cual se denomina Cloud Computing o simplemente cloud [29]. Dicha tendencia se ha convertido en un paradigma que permite ofrecer servicios de computación a través de Internet, donde sistemas de pequeño a gran porte trasladan sus recursos de procesamiento y almacenamiento de datos a grandes centros de datos que proporcionan servicios donde se paga por lo que se consume.

El proyecto que se describe en este informe surge de una iniciativa colaborativa e interdisciplinaria entre el Departamento de Ciencias de la Computación de la Universidad de Buenos Aires (Argentina) y el Centro de Cálculo

de la Universidad de la República (Uruguay). Frente a la creciente necesidad de manejar grandes volúmenes de datos dentro de períodos de tiempo razonables y con altas exigencias de desempeño, se plantea el abordaje de un problema de carácter biológico en el cual se pretende develar el complejo proceso que ocurre en los estadios iniciales del desarrollo embrionario. Para llevar a cabo el proyecto se propone utilizar computación distribuida en el cloud para alojar y procesar el software de simulación, que representa y modela la compleja realidad en cuestión, dando soporte a múltiples usuarios de forma simultánea.

Las principales motivaciones que propulsaron el desarrollo del proyecto incluyen facilitar la usabilidad de las aplicaciones de software de carácter biológico que modela la realidad planteada, independizar las ejecuciones de dichos programas del ambiente local del usuario e innovar en el área científica bajo el paradigma de computación distribuida en el cloud. De la mano de estas motivaciones se encuentran las principales contribuciones desarrolladas en el marco del proyecto:

- La aplicación de técnicas de computación científica de alto desempeño para resolver el problema de simular complejos procesos biológicos.
- Desarrollar un sistema con procesamiento de datos en paralelo, desplegado en una infraestructura de alto desempeño utilizando el paradigma de Cloud Computing.
- Facilitar la configuración y usabilidad de las aplicaciones de software.

En el contexto del proyecto se implementa una arquitectura altamente escalable y accesible haciendo uso de algoritmos empleados en la resolución de problemas de computación de alto desempeño. Los resultados obtenidos muestran que es posible adaptar sistemas científicos, como simulaciones biológicas, en una arquitectura distribuida en el cloud, logrando una paralelización de ejecuciones mediante el modelo de programación MapReduce. La solución implementada muestra resultados promisorios en la construcción de arquitecturas escalables y tolerantes a fallos para la adaptación de sistemas científicos.

El resto del informe se organiza de la siguiente manera. El capítulo 2 introduce los principales conceptos de cloud y analiza los principales proveedores de infraestructuras cloud. En el capítulo 3 se describe el problema biológico que se aborda así como la distribución de tareas y organización del proyecto. Y también se presenta una reseña de trabajos previos vinculados con el desarrollo del proyecto actual. El capítulo 4 presenta los principales aportes del proyecto describiendo de forma exhaustiva la arquitectura del sistema desarrollado. En el capítulo 5 se aborda en detalle la implementación llevada a cabo y las tecnologías empleadas durante el desarrollo. En el capítulo 6 se explican las pruebas realizadas y los resultados obtenidos. El capítulo 7 obra como resumen presentando las conclusiones generales del proyecto y describiendo por último las principales líneas de trabajo futuro.

Capítulo 2

Cloud Computing

2.1. Introducción

Cloud Computing (también llamado computación en el cloud) es una tendencia en tecnologías de la información y computación científica que traslada recursos de procesamiento y almacenamiento de datos de sistemas de pequeño a gran porte a extensos centros de datos que proveen servicios de la categoría "*paga por lo que usas*". Dentro de los recursos que son proporcionados por el cloud se encuentran: redes, servidores, almacenamiento de datos y servicios como software, entre otros. Una de las principales características que el cloud presenta es la rapidez con la que es posible obtener nuevos recursos y liberarlos sin interacción con el proveedor de servicio, así como un mínimo manejo administrativo. Gracias a la facilidad con la que es posible obtener y liberar recursos, el usuario percibe la ilusión de que los mismos son infinitos sin existir una gran inversión en infraestructura. El uso del cloud conlleva un beneficio económico, ya que al pagar sólo por los recursos utilizados se minimiza la cantidad de recursos ociosos. El paradigma de cloud computing promueve la disponibilidad de los recursos de cómputo y servicio, permitiendo el uso por parte de clientes heterogéneos y está compuesto por tres modelos de servicio y cuatro modelos de despliegue [30].

2.2. Conceptos relacionados

En esta sección se presentan conceptos relacionados al cloud que se utilizan a lo largo de los capítulos siguientes.

Clúster. Es un conjunto de recursos tecnológicos (generalmente computadoras) unidos mediante enlaces de alta velocidad en una red de área local. Un clúster provee un ambiente de procesamiento paralelo para aplicaciones que tengan la capacidad de dividir su ejecución y procesamiento entre los diferentes nodos. Entre los objetivos de un clúster se encuentran: mejorar el rendimiento (*performance*), la disponibilidad y la confiabilidad respecto a

una única computadora. Al utilizar un clúster, las tasas de errores se reducen mientras la disponibilidad y confiabilidad aumentan, ya que características inherentes al clúster son la redundancia y recuperación ante fallos [17].

HPC-Clúster. Incorpora una arquitectura de software implementada sobre clústers de forma de proveer alto rendimiento y procesamiento paralelo de datos, para aplicaciones que hacen uso de grandes cantidades de información (Big Data).

Centro de datos. Un centro de datos (*data center*) es un edificio (o una sección del mismo) destinado a albergar una sala informática y sus áreas de soporte. La infraestructura de un centro de datos está conformada por: espacio físico para la instalación de equipos informáticos, red de conexión a Internet y red privada, servicios de operación y supervisión de los componentes que la conforman.

Big Data. Son conjuntos de información de tal magnitud y complejidad que su procesamiento resulta impracticable utilizando manejadores de base de datos tradicionales.

Multi-Tenancy. En este modelo una sola instancia del software se ejecuta en un servidor y sirve a múltiples clientes. El software en el servidor particiona su información y configuración de manera virtual sirviendo de forma específica a cada cliente.

Virtualización. La idea de la virtualización es poder crear servidores, almacenamiento, redes y hasta aplicaciones virtuales sobre máquinas físicas. Esta abstracción es clave en el cloud ya que permite un acceso ubicuo. Para crear sistemas de hardware virtual, suelen utilizarse hipervisores.

Hipervisor. Un *hipervisor* o monitor de máquina virtual (*virtual machine monitor*) es una plataforma que permite aplicar diversas técnicas de control de virtualización para utilizar, al mismo tiempo diferentes sistemas operativos en una misma computadora.

API. La interfaz de programación de aplicaciones, del inglés API (*Application Programming Interface*), es el conjunto de subrutinas, funciones y procedimientos (o métodos) que ofrece cierta biblioteca para ser utilizada por otro software como una capa de abstracción.

Proceso maestro. En un sistema que sigue el modelo maestro–esclavo, el proceso maestro se encarga de lanzar, coordinar y monitorear la ejecución de los procesos esclavos. El proceso maestro envía los datos a procesar a los procesos esclavos.

Proceso esclavo. En un sistema que sigue el modelo maestro–esclavo, el proceso esclavo realiza el procesamiento de los datos enviados por el proceso maestro. Una vez finalizado el procesamiento, el proceso esclavo envía los resultados al proceso maestro y solicita más datos a procesar.

Endpoint. Los puntos de acceso o *endpoints* conforman una interfaz expuesta por una entidad que forma parte de la comunicación entre dos o más entidades o por un canal de comunicación. Los puntos de acceso facilitan una capa de abstracción en la comunicación para que sistemas heterogéneos puedan comunicarse entre sí.

Overhead. El *overhead* es cualquier combinación de exceso de tiempos de cálculo, uso de memoria, ancho de banda o recursos en general que se requiera para conseguir un objetivo particular.

Sistema operativo huésped. El sistema operativo huésped, para un servicio en el cloud, es el sistema operativo instalado en las máquinas virtuales en las que ejecuta el código de aplicación.

Despliegue. Un despliegue de servicios es una instancia de un servicio alojado en el cloud, ya sea en un ambiente de pruebas, en un entorno de producción, o en ambos.

Metadatos. Los metadatos son datos que describen otros datos. Ayudan a buscar recursos de datos y comprenderlos.

IDE. Entorno de desarrollo integrado (*Integrated Development Environment* o IDE) es una aplicación de software que proporciona servicios integrales para facilitar al programador el desarrollo de software.

2.3. Modelos de servicio y despliegue

2.3.1. Modelos de servicio

Cloud computing provee tres modelos de servicio que caracterizan los productos de los diferentes proveedores cloud. Estos modelos de servicio se diferencian en la definición de las capas (desde el hardware hasta el software) para las cuales el usuario posee control, como se puede observar en la Figura 2.1. El tipo de servicio a elegir depende de las necesidades del usuario, el tipo de producto a elaborar, el nivel de conocimiento técnico y el nivel de configuración deseado en el entorno de producción. Los tres modelos de servicio se presentan a continuación:

IaaS - Infrastructure as a Service. Infraestructura como servicio es un modelo de servicio en el cual el hardware está virtualizado en el cloud. La interacción de la aplicación con la IaaS suele darse a través de aplicaciones orientadas a servicios y contratos de servicios, estas aplicaciones utilizan tecnologías como WSDL (*Web Services Description Language*) o servicios RESTful. IaaS brinda una solución que permite satisfacer necesidades computacionales sin límite de escalabilidad en los despliegues, ya que los recursos se obtienen a demanda. Los proveedores poseen las máquinas físicas distribuidas por diferentes centros de datos a lo largo del mundo y mediante la virtualización se ponen a disposición de miles de usuarios al

mismo tiempo. Un ejemplo de una infraestructura como servicio es la brindada por Amazon Web Services, donde se puede acceder a almacenamiento, infraestructura de redes, máquinas virtuales, etc.

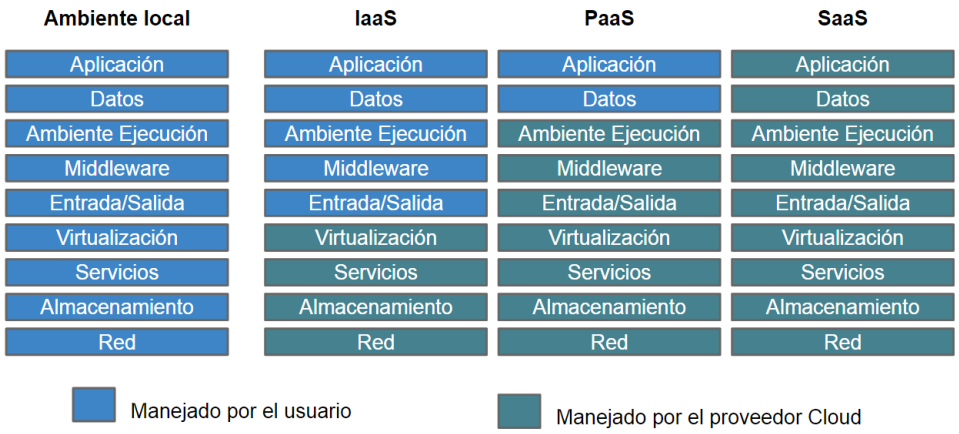

Figura 2.1: Modelos de servicios brindados por el cloud y aspectos controlados en cada uno de ellos.

PaaS - Plataform as a Service. Una plataforma como servicio es un modelo de servicio que se sitúa por encima de IaaS en cuanto al nivel de abstracción de los recursos tecnológicos. Este modelo propone un entorno de *software* en el cual el desarrollador puede crear y personalizar soluciones dentro de un contexto de herramientas de desarrollo que la plataforma proporciona. PaaS reduce la complejidad a la hora de desplegar y mantener aplicaciones, ya que las soluciones PaaS gestionan automáticamente la escalabilidad utilizando más recursos si fuera necesario. La plataforma puede estar basada en uno o varios lenguajes específicos o *frameworks* de desarrollo. Un ejemplo de este modelo es el brindado por Windows Azure, que provee herramientas de desarrollo y despliegue de aplicaciones integrado al IDE Visual Studio.

SaaS - Software as a Service. El modelo software como servicio es el que ha existido desde hace más tiempo, pero ha obtenido su nombre a partir de la definición del paradigma de cloud computing. SaaS básicamente se trata de cualquier servicio basado en la web; claros ejemplos son los sistemas de webmail como Gmail y los CRM (*Customer Relationship Management*) online. En este modelo el usuario accede al software dejando el desarrollo, el mantenimiento, las actualizaciones, el monitoreo y las copias de seguridad al proveedor del servicio. Por más que la mayoría de la gestión de configuración y mantenimiento se realiza por el proveedor, el usuario puede definir algunas configuraciones básicas de la aplicación. Todos los aspectos que no

son propios de la aplicación son transparentes para el usuario, por lo tanto el modelo de servicio SaaS es el más restrictivo en relación a los otros dos modelos de servicio [1].

2.3.2. Contratos de servicio

Service Level Agreement (SLA). SLA o nivel de servicio es un contrato entre el proveedor de servicios cloud y el usuario, que garantiza cierto nivel de rendimiento del sistema. Si existe este contrato y ocurre algún error en el servicio brindado que esté contemplado por el contrato, el proveedor debe corregirlo, liberando al usuario de tomar responsabilidad en el asunto.

Dentro del contrato de servicio se definen aspectos tales como el porcentaje de accesibilidad de recursos. Por ejemplo, Amazon tiene un porcentaje de accesibilidad de 99.5 %, que se corresponde a menos de cuatro horas y media de servicios inaccesibles por año [53]. Este tipo de contrato le da a los usuarios una confianza adicional, garantizando que aunque la demanda que percibe su sistema crezca de manera veloz, será posible obtener los recursos necesarios para satisfacerla. Dentro del contrato de servicio se definen niveles de servicios relacionados a una diversa gama de aspectos, como disponibilidad horaria, tiempo de respuesta, documentación disponible, personal asignado al mantenimiento del servicio, etc.

Un SLA se puede definir en relación a cuál es el mínimo nivel de rendimiento esperado en los diferentes aspectos del servicio brindado por el proveedor cloud. Por lo tanto, es importante leer el acuerdo de nivel de servicio a la hora de contratar el arrendamiento de servicios de un proveedor cloud, ya que en él se especifican claramente qué se puede esperar del servicio. Dado que la implementación del servicio brindado por el proveedor cloud no está disponible para el usuario, el SLA es la única herramienta que tiene el usuario para obtener una especificación detallada del servicio que está adquiriendo [17].

2.3.3. Modelos de despliegue

Es posible categorizar al tipo de infraestructura cloud en tres modelos, donde cada uno de ellos presenta características particulares no sólo en relación a la infraestructura sino en quién es el responsable de controlarla [29]. Los tres modelos de despliegue se presentan a continuación:

Cloud público. La infraestructura en el cloud público se encuentra en servidores externos al usuario. Este tipo de infraestructura se encuentra disponible para el acceso público y el dueño de la misma es el proveedor cloud. Ejemplos: Microsoft Azure, Amazon Web Services, etc [1].

Cloud privado. La infraestructura en un cloud privado es controlada por la organización que la crea y utiliza. Estas infraestructuras son más cercanas al despliegue tradicional de aplicaciones, hacen referencia a redes o centros de

procesamiento de datos propietarios que utilizan tecnologías características de Cloud Computing, como la virtualización [1]. Ejemplo: OpenNebula.

Cloud híbrido. La infraestructura de cloud híbrido es una composición de dos o más clouds (privados o públicos) que permanecen como entidades únicas pero se unen a través de estándares o tecnologías propietarias que permiten la portabilidad de datos y aplicaciones entre los clouds.

2.4. Ventajas y desventajas del cloud

2.4.1. Ventajas

El cloud posee muchas ventajas frente a otros modelos de servicios. Permite a través de la virtualización atender a varios clientes utilizando un modelo multi-tenancy donde los recursos físicos y virtuales se asignan y reasignan de acuerdo a las necesidades de los diferentes usuarios, sin que éstos se den cuenta del uso compartido de recursos. A su vez, los clientes pueden monitorear la utilización de los recursos, lo que brinda transparencia al proceso tanto para el usuario como para el proveedor.

La accesibilidad da la ilusión que los recursos son infinitos, creando un nuevo paradigma donde los sistemas a implementar en el futuro no se ven limitados a la cantidad de recursos disponibles. La posibilidad de escalar tanto horizontalmente como verticalmente bajo demanda, permite crear sistemas elásticos que se adaptan fácilmente a los cambios en el ambiente donde son ejecutados y para el cual fueron creados. El escalado horizontal implica la obtención de más recursos a demanda y la liberación de los mismos según sea necesario, este procedimiento puede involucrar ejecutar una nueva máquina virtual con el servidor escuchando peticiones, etc. El escalado vertical se da cuando un recurso necesita cambiarse por otro con diferentes prestaciones, por ejemplo un mejor poder de procesamiento, menor capacidad de almacenamiento, etc. Por más que existe en cloud computing esta opción, no es el tipo de escalado más común, ya que requiere dejar fuera de servicio el recurso mientras se hace el cambio de prestaciones.

Al utilizar los servicios cloud el usuario deja de responsabilizarse de muchos aspectos que impactan en la economía de la empresa, tal como el personal técnico encargado de mantener el ambiente de infraestructura operativo, las actualizaciones que implican nuevos ciclos de testeo, los gastos en energía y el enfriamiento de la maquinaria (servidores), las medidas de seguridad, el control de acceso para proteger los recursos de la infraestructura, el trabajo administrativo de manejo de licencias, las cuentas de usuario, el soporte, etc. Por lo tanto, cloud computing implica una reducción importante en los costos de la empresa [17].

Además, el cloud es una herramienta fuerte en proyectos colaborativos donde se comparten datos. Estos proyectos pueden pertenecer a diferentes

áreas: científica, económica, financiera o gubernamental; donde se realizan grandes análisis de datos compartidos y distribuidos por el mundo. El hecho de utilizar el cloud involucra aspectos de seguridad y control de acceso a tener cuenta, pero no deja de brindar accesibilidad universal a los datos [29].

2.4.2. Desventajas

Además de brindar muchos beneficios y permitir la optimización de procesos que serían muy costosos o llevarían tiempos inadmisibles en computadoras locales, el cloud presenta algunas desventajas en comparación con el despliegue en un ambiente local al usuario. Dentro de estas desventajas se encuentra la existencia de una brecha entre el cloud y los clientes que utilizan sus servicios, sobre todo a la hora de enviar información y descargar datos. A su vez, el cliente no sabe con exactitud cuál es la distribución de las máquinas virtuales en el cloud (dada la capa de virtualización) y a priori no puede definir el tiempo de transferencia de archivos entre las mismas. Por lo tanto, la latencia en los canales de comunicación es desconocida.

Aparte de los tiempos de transferencia y posibles fallas en los canales de comunicación, existe un punto importante que es la privacidad de los datos. La información utilizada por los sistemas desplegados en el cloud se encuentra en servidores con IP pública, y éste es un aspecto para el cual los usuarios de sistemas relacionados con Big Data tienen que estar preparados y saber manejar. La responsabilidad de proteger los datos sensibles pasa a ser compartida con el proveedor, lo que genera más puntos de vulnerabilidad.

Otro aspecto que se puede ver como ventaja y desventaja de la computación en el cloud es la disminución del control operacional. Por un lado es una ventaja, ya que quita muchas responsabilidades al usuario que solo quiere tener cierto sistema en producción sin preocuparse por lo que sucede detrás, haciendo que el nivel de control de poseer el sistema desplegado en el cloud en comparación con una puesta en producción local sea significativamente menor. Pero, por otro lado, el usuario no tiene la total certeza de cómo el proveedor de servicios cloud opera el mismo y cómo maneja los medios de comunicación con el mundo exterior, incluida la comunicación con el propio usuario, lo que genera riesgos que no existen a nivel local.

Adicionalmente, podría suceder que un usuario quiera cambiar de un proveedor a otro dado aspectos comerciales, nuevos servicios brindados por otro proveedor, etc. Actualmente, la compatibilidad entre diferentes proveedores de servicios cloud no está garantizada, ya que no existen estándares establecidos en el mercado de cloud computing, lo que dificulta esta migración [17].

Dentro de las vulnerabilidades presentes en el cloud se encuentran ataques maliciosos a través de Internet (DoS, etc.), fallos en la infraestructura, como pueden ser fallas eléctricas debido a eventos climáticos y errores en el software del cloud [29].

2.5. El cloud y el mundo científico

El modelo de cloud computing se adapta a las necesidades del mundo científico permitiendo realizar cada día más avances, ya que los recursos informáticos de alto desempeño suelen tener costos altísimos y por lo tanto sólo son accesibles por grandes empresas o centros de investigación. Además, generalmente en las instituciones se cuenta con infraestructura limitada. Esto conduce a tener que compartir el clúster o infraestructura entre diferentes proyectos, tomando turnos o teniendo que reservar con días o semanas de anticipación, limitando el desarrollo y la investigación. El cloud permite acceder a los recursos de forma inmediata a la vez que proporciona mecanismos de administración de recursos, siendo posible contratar solo aquellos que se necesitan de forma personalizada.

La utilización de máquinas virtuales permite a la comunidad científica instalar herramientas y entornos personalizados sobre la arquitectura provista por el cloud, permitiendo a los científicos un ambiente muy flexible sobre el cual es posible adaptar el entorno a medida para el proyecto que se quiera desarrollar.

El manejo y almacenamiento de datos en los proyectos científicos suele ser de vital importancia. Por un lado, debido al gran volumen a manejar y almacenar, y por otro lado, debido a que algunos proyectos cuentan con varias instituciones trabajando de forma conjunta y distribuida geográficamente. Esto podría desencadenar varios problemas a la hora de manejar los datos. Por ejemplo, la obtención de información actualizada y la forma de compartir los datos entre las instituciones puede implicar la descarga o transferencia de volúmenes de datos muy grandes, lo que podría consumir demasiado tiempo.

Uno de los beneficios principales del cloud para las aplicaciones científicas es por lo tanto el mayor poder de cómputo obtenido y la facilidad con la que se pueden escalar los sistemas en caso de necesitar más recursos. Pero aun así, antes de migrar al cloud es importante evaluar si el tipo de algoritmo que forma parte de la aplicación científica (que originalmente puede estar ejecutando en un clúster) es adaptable a alguno de los modelos de computación distribuida en el cloud (MapReduce, etc.), dado que la complejidad del mismo podría no obtener mejoras significativas de rendimiento en el cloud en comparación a ejecutar en un ambiente local [25].

El cloud también brinda la posibilidad de que mayor cantidad de científicos puedan participar y estar involucrados en la puesta en producción de sus proyectos, ya que al existir diferentes niveles de servicios (IaaS, PaaS, SaaS), no necesariamente tienen que ser expertos y poseer un profundo conocimiento en el área informática para poder aprovechar los beneficios del cloud. Dados estos beneficios y los diferentes niveles de servicio que presenta el cloud, existe actualmente una tendencia a migrar los sistemas científicos desde los clústers locales al cloud y existen desarrollos que facilitan el proceso [45].

2.6. Modelo MapReduce

MapReduce es un modelo de programación que permite manejar y procesar grandes volúmenes de información. El modelo fue utilizado originalmente por Google, pero actualmente es utilizado por varias compañías web. MapReduce se basa en el paradigma "*divide y conquistarás*", donde el problema original se divide en subproblemas más pequeños hasta que los mismos se pueden resolver de manera independiente. Los subproblemas luego son asignados a procesos esclavos que los resuelven en paralelo. Al finalizar, todas las soluciones a los subproblemas independientes se combinan para llegar a la solución del problema original. MapReduce busca poder utilizar de manera beneficiosa la estructura de clúster de computadoras, que presenta múltiples recursos de cómputo que pueden utilizarse en paralelo. Para implementar este modelo de "*divide y conquistarás*" se utilizan dos funciones: Map y Reduce.

- **Map.** Es una función que procesa datos en formato *clave-valor* de manera independiente. *Map* resuelve los subproblemas definidos a partir del problema original. Como se observa en la ecuación 2.1, el resultado de ejecutar la función *Map* es un conjunto intermedio de pares *clave-valor*.

$$map(K1, V1) \longrightarrow list(K2, V2) \tag{2.1}$$

- **Reduce.** Es una función que toma los resultados intermedios generados por las funciones *Map* ejecutadas, y los combina para obtener la solución al problema original. La función *Reduce* se aplica sobre un conjunto de pares *clave-valor* que poseen una misma clave. Como se observa en la ecuación 2.2, el resultado de ejecutar la función *Reduce* es una lista de valores.

$$reduce(K2, list(V2)) \longrightarrow list(V3) \tag{2.2}$$

La implementación más difundida de este modelo es Hadoop, realizada por Apache Foundation. Hadoop es un *framework* implementado en Java inspirado en el modelo MapReduce y el sistema de archivos GFS de Google. Este *framework* se utiliza globalmente tanto en áreas comerciales como de investigación y posee una extensa documentación dada la cantidad de usuarios que lo utilizan. Hadoop es una herramienta primaria a la hora de resolver problemas de Big Data [46]. Esta herramienta adapta las tecnologías de Google y presenta un *framework* formado por tres proyectos principales: *HDFS* (Hadoop Distributed File System), *MapReduce* y *Common*. HDFS es un sistema de archivos distribuido que puede almacenar grandes cantidades de datos y que presenta un rendimiento alto y múltiples réplicas en el clúster. MapReduce implementa el modelo de programación para el análisis y procesamiento de datos extremadamente grandes de manera rápida, escalable

y distribuida. Common es un subproyecto que ofrece APIs y componentes integrados en Java para dar soporte al sistema de archivos distribuido y aspectos de entrada/salida en este tipo de sistemas. Common se puede ver como una biblioteca que cuenta con todas las características que HDFS y MapReduce utilizan para manejar la computación distribuida [31].

Siguiendo el modelo propuesto por el clúster de Apache, hay dos tipos de nodos en una infraestructura Hadoop: nodos maestros y nodos esclavos. El nodo maestro toma la entrada y la divide en subproblemas más pequeños, distribuyendolos en los diferentes nodos esclavos y llevando un registro del estado de cada uno de ellos. Los nodos esclavos ejecutan el paso *Map* y cuando finalizan le devuelven el resultado al nodo maestro. Luego que el nodo maestro obtiene todas las soluciones a los subproblemas, las distribuye en los nodos esclavos que las combinan para formar la salida ejecutando el paso *Reduce* [37].

El framework está construido en base a pares *clave-valor*. El formato *clave-valor* es la estructura de información utilizada en la comunicación entre los distintos módulos del *framework*. Estos pares pueden ser tipos primitivos de datos o extensos tipos de datos (*data-types*) [31].

Hadoop cuenta con más etapas además de la ejecución de las funciones *Map* y *Reduce*, que se encargan de administrar y transferir los datos de un nodo al otro [31]. Como se muestra en la Figura 2.2, entre estas etapas se encuentran:

Input Format. Lee los archivos del HDFS, tabla de base de datos o dispositivo que el usuario haya especificado que quiere leer. Este paso toma los datos y los divide en *InputSplits*.

InputSplits. Es un subconjunto de los datos leídos, siendo cada *InputSplit* entregado a un nodo esclavo diferente que ejecutará la función *Map* sobre el mismo.

Map. Cada nodo esclavo ejecuta una tarea *Map* y estos nodos corren paralelamente entre ellos. Un nodo toma los registros *clave-valor* generados de su *InputSplit*, los procesa y genera otros pares *clave-valor*, como se puede observar en la ecuación 2.1.

Combine - Combinar. Es una etapa opcional que ejecuta en cada nodo directamente después de ejecutar la función *Map* y junta los pares con la misma clave que fueron generados por esa tarea.

Shuffle - Ordenar. Ésta es la única etapa donde los nodos se comunican entre si. Los pares *clave-valor* se transfieren entre los nodos para concatenarlos, ordenarlos y particionarlos.

Shuffle - Concatenación. Esta etapa junta todos los datos generados por *Map* y es realizada automáticamente por el framework.

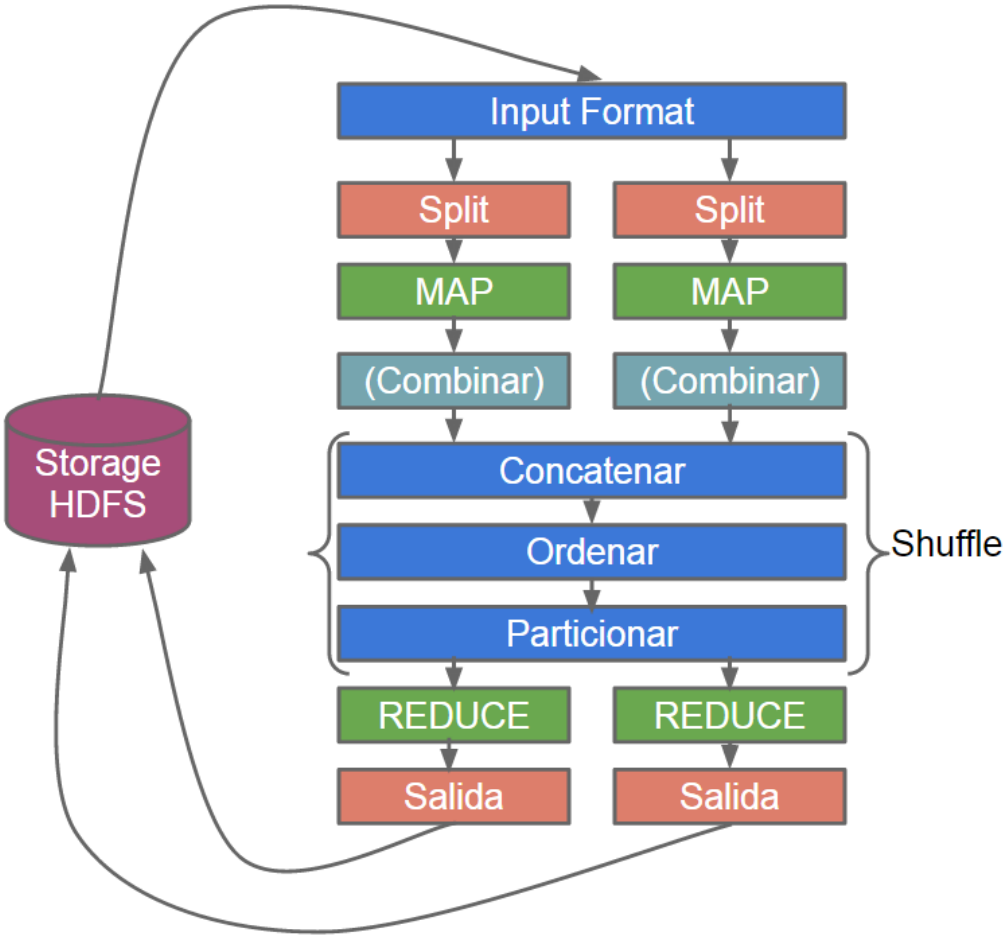

Figura 2.2: Las etapas de MapReduce en Hadoop MapReduce.

Shuffle - Orden. En esta etapa se ordenan los datos. Éstos se pueden ordenar de manera personalizada especificando una implementación; por defecto los mismos son ordenados según sus claves.

Shuffle - Partición. Esta es la última etapa de la fase de ordenamiento de los datos, donde se calcula cómo los datos se deben dividir para la etapa de *Reduce*. La manera en la que se dividen los datos se puede manejar por defecto o definir mediante una implementación brindada por el usuario. Esta etapa debe generar una cantidad de datos igual para cada nodo esclavo que ejecutará la función *Reduce* para garantizar un óptimo rendimiento.

Reduce. *Reduce* se ejecuta tomando todos los pares *clave-valor* con una misma clave y ejecutando algún tipo de función que los combine, como se

puede observar en la ecuación 2.2. Si a una tarea *Reduce* le corresponde una clave, a ninguna otra tarea del mismo tipo le corresponderá esa clave.

Salida. Cada tarea *Reduce* genera una salida al dispositivo de almacenamiento. Por defecto, la salida se genera en archivos parciales, uno por cada tarea *Reduce*, en archivos llamados part-r-00000, part-r-00001, etc. La salida se puede controlar brindando una implementación de una subclase de la clase OutputFormat.

Cuando se inicia un clúster Hadoop MapReduce se requiere un nodo maestro que contenga la información del HDFS y un *JobTracker*. El *JobTracker* o rastreador de trabajos es un planificador y es el punto de entrada a una tarea MapReduce. El *JobTracker* se comunica con los *TaskTrackers* (rastreadores de tareas) que ejecutan en los diferentes nodos esclavos del clúster. Los *TaskTrackers* realizan *pings* periódicos al *JobTracker* para verificar que existan tareas disponibles a ejecutar, y notifican el estado de las tareas que están ejecutando. Por defecto, el *TaskTracker* espera recibir acuse de recibo por parte del *mapper* o *reducer* en un determinado periodo de tiempo, caso contrario el *TaskTracker* asume que la tarea *Map* o *Reduce* falló y aborta el proceso. Si el *JobTracker* tiene una tarea lista para ser procesada se la envía al *TaskTracker* para que la ejecute [31].

2.7. Proveedores de servicios cloud

En esta sección se presentan los proveedores cloud estudiados en el proyecto. Se relevan los servicios brindados y el costo de los mismos, así como las opciones de financiamiento.

2.7.1. Amazon Web Services

Amazon se caracteriza por ser unos de los primeros proveedores de servicios cloud en brindar el modelo de servicio IaaS. Amazon es uno de los proveedores que lleva más tiempo en el mercado al igual que Google App Engine (GAE); la diferencia entre ambos es que GAE brinda servicios del modelo PaaS principalmente.

La base de los servicios brindados por Amazon Web Services (AWS) es la posibilidad de generar máquinas virtuales personalizadas, llamadas *Amazon Machine Image* (AMI), las cuales pueden ser clonadas y reutilizadas. Existen AMIs por defecto con diferentes prestaciones de hardware y sistemas operativos, con diferentes precios disponibles y adaptables a las necesidades de cada usuario. Un aspecto positivo de trabajar con AMIs es que es posible obtener la infraestructura que más se adapte a un determinado objetivo, ya sea la implementación de aplicaciones con un nivel de cómputo muy elevado o que manejen grandes cantidades de datos, etc. El acceso a las instancias de las AMI se realiza de manera remota, al igual que el acceso a un servidor

remoto convencional. AWS brinda una consola de administración donde es posible monitorear el estado de las diferentes instancias de AMI en estado de ejecución en un momento dado. Además, es posible también configurar la ampliación de recursos (disco duro, RAM, etc.) según las condiciones que se hayan establecido y el contrato de servicio existente entre AWS y el usuario. Los precios se definen en relación a una hora de uso de la máquina virtual y los tipos de recursos que la misma posee (capacidad de almacenamiento, RAM, sistema operativo, etc.).

AWS brinda un gran número de servicios, dentro de los cuales se destacan *Elastic Compute Cloud* (EC2) y *Amazon Simple Storage Service* (S3).

Amazon Elastic Compute Cloud (EC2). EC2 es un servicio web que brinda capacidad de cómputo redimensionable, permitiendo ejecutar instancias de aplicaciones bajo distintos sistemas operativos. Una instancia se puede crear a partir de una AMI almacenada en el servicio de almacenamiento S3 o una imagen creada por el usuario. Esta imagen posee el sistema operativo, el ambiente de ejecución de la aplicación, las bibliotecas necesarias y la aplicación que se desea ejecutar. Un usuario puede iniciar y detener una instancia, crear una nueva imagen, agregar etiquetas para identificar instancias y reiniciar una instancia. EC2 permite importar máquinas virtuales desde el ambiente del usuario a una instancia AMI fácilmente, utilizando un importador. También permite el control de tráfico entre varias instancias utilizando un balanceador de carga elástico. Además, EC2 asocia una IP pública a una cuenta, mecanismo que mitiga el fallo de una instancia al redirigir la dirección IP pública a otra instancia de la cuenta, sin necesidad de acceder al equipo de soporte de software [29].

Existen diferentes maneras de contratar las instancias. Una posibilidad es bajo demanda (*on demand*) donde se paga el uso de las instancias por hora. Por otro lado, es posible reservar instancias (*reserved instances*) haciendo un único pago de menor costo, pero siendo también menor la tasa de disponibilidad. Un tercer modelo de contrato de instancias es el llamado *Spot Instances* (puntos de instancia), donde el usuario realiza una oferta para utilizar recursos ociosos reduciendo así considerablemente el costo total gastado, pero sujeto a disponibilidad variable de los recursos, ya que si otro usuario posee instancias reservadas o debe tomarlas bajo demanda, tendrá prioridad sobre su uso.

Las instancias se pueden ejecutar en diferentes regiones geográficas. Cada región tiene varias zonas de disponibilidad, que son diseñadas para minimizar el impacto de posibles fallos en la infraestructura del cloud, aislandolas entre sí. Aunque las zonas son independientes, AWS provee conectividad entre ellas a bajo costo y baja latencia, dentro de una misma región geografica [52].

Amazon Simple Storage Service (S3). S3 es un servicio de almacenamiento distribuido diseñado para almacenar grandes cantidades de datos. El servicio soporta tres operaciones principales: escribir, leer y borrar. Los

datos son almacenados en contenedores que se pueden acceder utilizando las directivas de HTTP, permitiendo que una aplicación pueda manejar una cantidad ilimitada de objetos de entre 1B a 5TB cada uno. Los estándares utilizados por S3 son SOAP y RESTful [29].

Elastic Map Reduce (EMR). AWS provee una implementación del *framework* Hadoop alojada en la infraestructura web de EC2 y S3, que permite al usuario crear flujos de trabajo personalizados. Un flujo de trabajo es una secuencia de pasos MapReduce.

Eucalyptus

Eucalyptus es un sistema de administración de cloud open-source gratuito que utiliza la misma API que AWS, y por lo tanto, permite realizar implementaciones locales para luego migrarlas a AWS. Eucalyptus provee las mismas funcionalidades en términos de despliegue IaaS y puede utilizarse tanto como nube pública, privada o híbrida mientras se disponga del hardware necesario. Las instancias que ejecutan dentro de Eucalyptus son las Eucalyptus Machine Images (EMI, llamadas a partir de las AMIs) que al igual que las AMIs se pueden crear o utilizar versiones ya existentes. La arquitectura de Eucalyptus es distribuida y sus componentes son desarrollados como servicios web. Estos servicios web se ejecutan sobre Ubuntu, Debian y algunas otras distribuciones de Linux o Windows. Actualmente, Eucalyptus no posee un análogo al módulo que implementa Hadoop en AWS (EMR), por lo que se debe utilizar directamente el *framework* de Apache y migrar éste a AWS posteriormente sin poder sacar provecho de EMR [49].

2.7.2. Microsoft Azure

Microsoft Azure provee servicios siguiendo los modelos PaaS y SaaS principalmente, incorporando recientemente el modelo IaaS. La plataforma posee tres componentes core: *Compute* provee un ambiente de ejecución para ejecutar tareas o aplicaciones, *Storage* un almacenamiento de datos escalable y redundante y *Fabric Controller* que permite desplegar, manejar y monitorear las aplicaciones [29].

La API de Windows Azure se basa principalmente en servicios web RESTful, HTTP y XML, presentando una gran facilidad de acceso (ya sea a través de una plataforma web, consola, etc.). Las ejecuciones realizadas por las aplicaciones en la plataforma se implementan como uno o más roles, siendo posible ejecutar varias instancias de cada rol, dependiendo de la demanda de recursos presente. Los posibles roles existentes en la plataforma son: *Web Roles, Worker Roles, Virtual Machine Roles*. Se profundizará sobre estos roles más adelante en esta sección. Microsoft Azure también cuenta con un servicio que presenta una implementación del modelo MapReduce de Apache, llamado HDInsight.

La escalabilidad, el balanceo de carga, el manejo de memoria y la confiabilidad son responsabilidad del módulo *Fabric Controller*, una aplicación distribuida y replicada a través de varios grupos de máquinas, que posee conocimiento y administra todos los recursos de su ambiente: computadoras, *switches*, balance de carga y cada aplicación desplegada en la plataforma [29].

Los servicios brindados por Windows Azure se pueden separar en distintos grupos como se presenta en la figura 2.3.

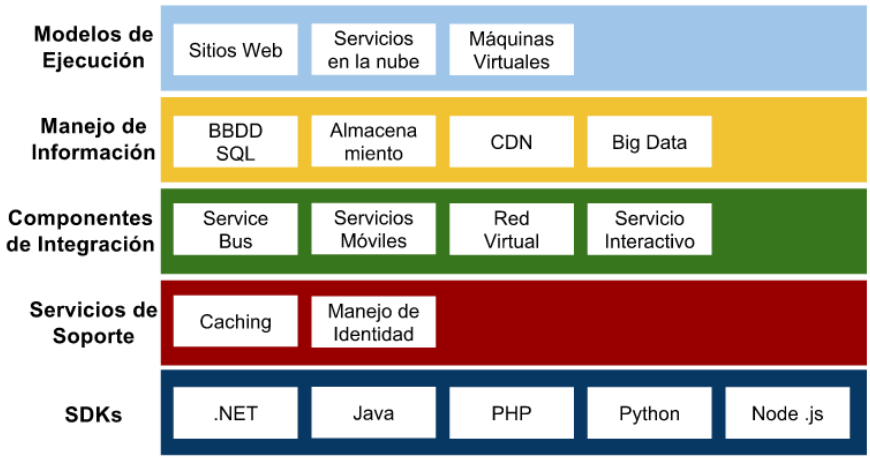

Figura 2.3: Capas de servicios brindadas por Windows Azure

La creación, configuración y monitoreo de estos servicios por parte de los usuarios se puede realizar desde la plataforma web llamada *Microsoft Azure Portal*, donde se presenta una interfaz amigable para el manejo de recursos [12]. Por lo tanto, no es necesario instalar ningún sistema para controlar las aplicaciones en el cloud, aunque para el desarrollo de aplicaciones para Azure, sí es necesario instalar varios sistemas que ayudan a depurar la aplicación. Desde el portal web es posible definir criterios de escalabilidad por porcentaje de uso de CPU, o simplemente programar el escalado en base a conocimiento de períodos de mayor demanda. En varios de sus servicios es posible acceder de manera remota a las máquinas virtuales que los conforman, por ejemplo a un *Worker Role* o a un clúster HDInsight, no así a una instancia de *Web Site*, como se verá más adelante.

Aunque Microsoft Azure es una plataforma propietaria, permite la creación de máquinas virtuales en otros sistemas operativos como Linux, y gracias a su modelo de API basado en servicios REST, permite el desarrollo de sus aplicaciones en diversos lenguajes como: Java, Phyton, PHP, .NET, Node.js, etc.

Modelos de ejecución y Roles

Dentro de los modelos de ejecución provistos por Microsoft Azure se encuentran: *Web Sites*, *Cloud Services* y *Virtual Machines* (Máquinas virtuales). Cada uno de estos modelos posee características específicas que definen el grado de libertad y de responsabilidad por parte del usuario en el desarrollo de los mismos, y dependerá de las necesidades de cada usuario determinar cuál utilizar al momento de realizar una aplicación para Microsoft Azure.

Web Sites. Los *Web Sites* (sitios web) de Azure forman parte del modelo de servicio PaaS, donde el usuario no debe preocuparse de toda la configuración del entorno, como se observa en la Figura 2.4. Son de fácil configuración y puesta en producción, existiendo *templates* (plantillas) en el portal web (*Azure Web Sites gallery*) que permiten crear un sitio con características desarrolladas por terceros. Algunos ejemplos de ello incluyen Drupal o Wordpress, los cuales permiten la creación de nuevos sitios en cuestión de segundos. Además, es posible utilizar *frameworks* de desarrollo como CakePHP. Web Sites de Azure permite tanto la creación de nuevos sitios, como la migración de sitios ya existentes. En ambos casos es posible configurar el escalado del sitio según criterios a definir.

Los *Web Sites* se pueden integrar con otros servicios de Azure, como pueden ser *SQL Database*, *Service Bus* y *Storage*. Se pueden realizar tareas de cómputo mediante la utilización de *WebJobs*. Los mismos permiten cargar y ejecutar archivos con extensión .cmd, .bat, .exe, etc., pero las tareas no pueden ser escaladas de forma independiente al sitio web. Al utilizar *Web Sites*, no es posible acceder de manera remota a las máquinas virtuales que tienen *hosteado* el sitio web, y no es posible configurar controles de seguridad muy elevados.

Cloud Services. Los *Cloud Services* permiten crear aplicaciones web altamente disponibles y escalables en un ambiente PaaS. La diferencia con los *Web Sites* es que no se pueden crear directamente a través del portal de administración, sino que deben ser desarrollados en un ambiente como Visual Studio para luego ser desplegados en el cloud. La principal característica de los Cloud Services es que permiten desarrollar arquitecturas multicapa donde cada capa se puede abstraer tanto física como virtualmente. A su vez, permiten mayor nivel de configuración y personalización, como la ejecución de *scripts* al iniciar una instancia, la realización de configuraciones de red, etc.

Un único Cloud Service está conformado por un componente *front-end* llamado *rol web* (*Web Role*) y uno o más componentes *back-end* llamados *rol de trabajo* (*Worker Role*), siendo cada uno de estos roles escalables independientemente de los demás. Existe un mayor nivel de control sobre la infraestructura en relación a los *Web Sites* (observar la Figura 2.4). Por ejemplo, es posible acceder mediante escritorio remoto a la máquina virtual

en la cual se ejecuta la instancia de un rol específico. Además, los roles que forman parte de los *Cloud Services* pueden acceder a todos los servicios de Azure, incluyendo HDInsight [50].

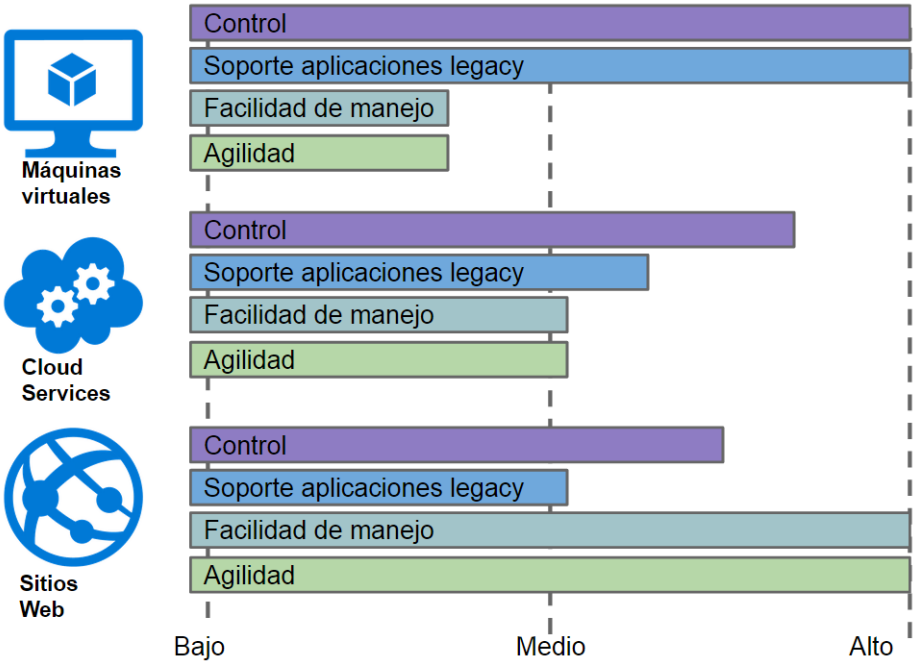

Figura 2.4: Comparación de modelos de ejecución en Azure

Virtual Machines. Las máquinas virtuales de Azure permiten la ejecución de aplicaciones en el cloud. Su modelo de servicio es IaaS y es posible crear máquinas con sistema operativo Windows Server o Linux, o cargar una imagen ya existente de máquina virtual a través del portal de administración. Las máquinas virtuales poseen el mayor grado de control sobre el sistema operativo, permitiendo la configuración del software y de los servicios instalados. Existe una amplia variedad de configuraciones de máquinas para distintas situaciones (ver Cuadro 2.1). Las imágenes de máquinas virtuales para las distintas combinaciones de configuración dentro de la galería de Microsoft Azure se denominan de A0 a A9. Por ejemplo, una imagen A9 proporciona una máquina virtual con 16 *Intel virtual processor cores* y 112GB de memoria RAM [11]. *Virtual Machines* es una buena idea para migrar aplicaciones *on-premises* (en ambientes locales) al cloud, porque dichas aplicaciones se pueden migrar a máquinas virtuales sin la necesidad de realizar grandes cambios en las mismas. También es posible conectar las *Virtual Machines* de Azure a máquinas virtuales locales creando una arquitectura híbrida. Así como con los *Cloud Services*, es posible acceder de manera remota a las máquinas virtuales para ejecutar tareas de administración o control. La principal diferencia entre las *Virtual Machines*, los *Web Sites*, y

los *Cloud Services* es que en las dos primeras, todos los aspectos de infraestructura y arquitectura deben ser controlados por el usuario, brindando más control administrativo pero más dificultad de manejo, y con la necesidad de poseer más conocimiento en tecnologías para su configuración.

Tamaño VM	CPU núcleos	Memoria	Núm. discos
A0	Compartido	768MB	1
A1	1	1.75GB	2
A2	2	3.5GB	4
A3	4	7GB	8
A4	8	14GB	16
A5	2	14GB	4
A6	4	28GB	8
A7	8	56GB	16
A9	16	112GB	16

Cuadro 2.1: Configuraciones de *Virtual Machines* en Azure

Los tres modelos presentados poseen características que según el problema específico de cada usuario brindarán una mejor solución para el uso de la plataforma Azure. Las principales características de cada modelo se pueden observar en la Figura 2.4. Si se desea una aplicación corporativa con poca configuración y acceso a una base de datos, *Web Sites* es la opción que mejor se adapta a estas necesidades. Por otro lado, si se desea crear una arquitectura con más de una capa que ejecute aplicaciones con gran nivel de cómputo y específicas condiciones de escalado, *Cloud Services* es la opción correcta. En cambio, si se desea instalar en el cloud una aplicación que requiere gran configuración del sistema operativo subyacente, la mejor opción en este caso sería utilizar *Virtual Machines*.

Almacenamiento en Azure

El acceso al almacenamiento en el cloud de Azure se realiza mediante servicios web RESTful, lo que lo hace altamente disponible, desde una aplicación web, consola de comandos, etc. Desde el portal web con mucha facilidad se pueden crear las llamadas *Storage Accounts* (cuentas de almacenamiento) en las cuales se permite definir contenedores de datos (*Containers*) y dentro de los cuales se permite crear *Blobs* (Figura 2.5), uno de los tres servicios de almacenamiento que posee Azure. Existen a su vez otros servicios de almacenamiento entre los que se encuentran: *Tables y Queues*.

Según el tipo de cuenta de Microsoft Azure que el usuario posea se limita la cantidad de *Storage Accounts* permitidas. Dentro de una *Storage Account* se puede tener un número ilimitado de *Containers*, y dentro de los mismos se puede contener una cantidad ilimitada de blobs, mientras respeten los tamaños máximos de almacenamiento permitidos. Una *Storage Account* puede contener hasta 200TB entre *Blobs*, *Tables* y *Queues*.

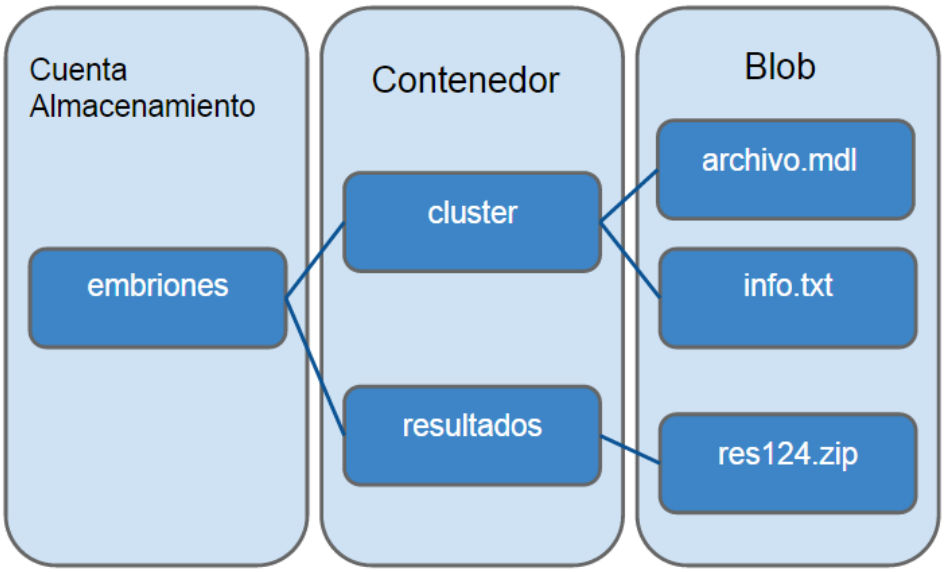

Figura 2.5: Componentes Azure Storage - Blobs

Blobs. Un *blob* es un archivo de cualquier tipo. Los *blobs* se pueden clasificar dentro de dos categorías: *Block* y *Page Blob*, siendo la mayoría de los archivos *Block Blobs*. Un único *Block Blob* puede almacenar hasta 200GB, mientras que los *Page Blobs* pueden almacenar hasta 1TB y son más eficientes cuando un rango de bytes del archivo es modificado con regularidad. El acceso a un *Blob* se realiza utilizando una URL con el siguiente formato:

```
http://<storageAccount>.blob.core.windows.net/<container>/<blob>
```

Los *blobs* se pueden crear desde sistemas como *Azure Storage Explorer* o utilizando la API directamente en el desarrollo de aplicaciones para Microsoft Azure.

Tables. El servicio de *Azure Tables* (tablas) permite almacenar grandes cantidades de datos estructurados. Una tabla puede ser definida dentro de una *Storage Account* (pero no dentro de un *Container*, como es el caso de los *blobs*) y está conformada por un conjunto de entidades. La diferencia entre las tablas de base de datos y las tablas de Azure es que las presentes en Azure no poseen un esquema de entidades, por lo cual las entidades dentro

de una misma tabla pueden presentar distintos conjuntos de propiedades. El límite en la cantidad de tablas que una *Storage Account* puede contener está delimitado por la capacidad límite de almacenamiento de la cuenta. Una entidad puede contener hasta 1MB de información, y las propiedades que ésta presenta (que pueden ser hasta 252) se representan en el formato *nombre-valor*. Cada entrada de la tabla tiene a su vez tres propiedades del sistema que especifican una clave de partición, una clave de entidad y un *timestamp* (marca de tiempo). Las entidades con la misma clave de partición pueden ser recuperadas de manera más rápida, mientras que la clave de entidad identifica unívocamente a una entidad dentro de una partición.

Queues. El servico de almacenamiento *Azure Queue* permite almacenar una gran cantidad de mensajes que pueden ser accedidos desde cualquier lugar del mundo utilizando llamadas HTTP o HTTPS de manera autenticada. El tamaño máximo de estos mensajes es de 64KB y una cola de mensajes (*Queue*) puede almacenar millones de éstos, hasta llegar a la capacidad límite de almacenamiento de la cuenta asociada.

HDInsight

HDInsight es un servicio de Microsoft Azure que aprovisiona y despliega clústers de Apache Hadoop en el cloud, brindando un *framework* para manejar, analizar y reportar grandes volúmenes de datos. Es decir, HDInsight brinda una implementación del modelo MapReduce de Apache Hadoop para el análisis de Big Data.

Los clústers Hadoop en HDInsight utilizan una versión de la distribución de *Hortonworks Data Platform* (HDP) y un conjunto de componentes de Hadoop de dicha distribución. En la versión actual de HDInsight, la 3.1, se utiliza la versión 2.1.7 de HDP y 2.4.0 de Apache Hadoop.

En HDInsight el sistema de archivos de Hadoop (HDSF) se almacena como *Blobs* en un *Container* asociado al clúster en la *Storage Account*.

2.8. Arquitecturas en el cloud

Las arquitecturas en el cloud son inherentemente distribuidas y siguen siempre alguna de las variantes de *cliente-servidor*, donde un componente cliente solicita y consume servicios de otro llamado servidor. Una característica particular de esta relación cliente-servidor en el cloud y que es de suma importancia para obtener características deseables como la tolerancia a fallos, es que los servidores no mantienen estado de las conexiones con sus clientes, sino que cada una de ellas se maneja de manera independiente. Este tipo de arquitectura no solo responde de mejor manera a las fallas sino que también es escalable. Las características que se desea potenciar en las arquitecturas en el cloud son:

- **Escalabilidad** - Se desea aprovechar la ilusión de poseer *recursos infinitos* brindada por el cloud para crear arquitecturas altamente escalables, donde se permita agregar o eliminar de manera sencilla componentes según la demanda del sistema.
- **Confiabilidad** - Es importante asegurar el correcto funcionamiento de los componentes de *hardware* y *software* que conforman el sistema, ya que el mismo no se encuentra en un ambiente local al usuario. Para potenciar este punto, se debe estudiar el contrato de servicio (SLA) del proveedor cloud elegido y aprender bajo qué condiciones los proveedores garantizan que los sistemas no tendrán fallas o caídas. Al momento de definir la arquitectura, puede ser beneficioso definir componentes replicados y distribuidos geográficamente (manejados con un balanceador de carga, por ejemplo) asegurando que la probabilidad de falla del sistema global disminuya considerablemente.
- **Accesibilidad** - Esta propiedad hace referencia al tiempo en el cual el sistema se encuentra disponible y funcionando correctamente en relación al tiempo que efectivamente se espera que funcione. Para potenciar esta característica es importante estudiar el contrato de servicio del proveedor cloud, así como sacar provecho de la redundancia de componentes al momento de definir la arquitectura del sistema.
- **Tolerancia a fallos** - La tolerancia a fallos es la capacidad del sistema de recuperarse y volver a un estado estable frente a una falla física o del sistema en sí. Se debe considerar la tolerancia a fallos al momento de definir los distintos componentes y los roles de la arquitectura del sistema, así como las tecnologías a utilizar para la implementación del mismo.

Para que el sistema implementado cuente con estas cualidades hay que tener en cuenta otros aspectos y características que pueden jugar un papel negativo o perjudican de alguna manera al usuario, por el hecho de desarrollar un sistema en el cloud. Algunos de estos aspectos se mencionan a continuación:

- **Costo** - Diseñar una arquitectura que cumpla con características que fomenten las cualidades deseadas puede implicar un costo extra en el contrato con el proveedor de servicios cloud. Un punto a tener en cuenta es si se quiere migrar un sistema al cloud buscando optimizar costos en relación a mantener una arquitectura local, ya que existen costos adicionales que no se encuentran en los ambientes locales, como la transferencia de datos entre dos componentes de la arquitectura, por ejemplo.
- **Complejidad** - Se debe considerar que arquitecturas complejas conllevan mayores dificultades de configuración, mantenimiento y a su vez pueden generar deterioros en el rendimiento del sistema. Se puede generar un deterioro en el sistema si los componentes que conforman el

mismo se encuentran distribuidos a grandes distancias y el sistema no es diseñado para soportar grandes latencias en términos de comunicación, por ejemplo.

- **Velocidad** - Se puede esperar de un sistema desplegado en el cloud mejoras en los tiempos de respuesta así como enlentecimientos, dependiendo de la arquitectura diseñada y las herramientas utilizadas para mitigar latencias. Algunas herramientas para mitigar latencias pueden ser utilizar una red de distribución de contenido (*Content Distribution Network - CDN*) o un servicio de caché; cabe destacar que estas herramientas pueden generar gastos extras.
- **Portabilidad** - No existen actualmente estándares en relación a los servicios cloud, por lo que resulta muy difícil migrar de un proveedor de servicios cloud a otro.
- **Seguridad** - Es importante tener en cuenta que toda comunicación entre componentes de la arquitectura con el mundo exterior se realizan a través de Internet y es, por tanto, de suma importancia utilizar protocolos seguros para mantener la confidencialidad de los datos. Utilizar encriptación o generar una red privada virtual (VPN) entre el usuario y el cloud son buenas prácticas para mitigar este punto.

A continuación se presentan algunas *arquitecturas tipo* en el cloud y las características que ellas presentan:

1. **Arquitectura autoescalable:**

   La Figura 2.6 muestra una arquitectura autoescalable donde se saca provecho de unas de las características principales del cloud, el escalado horizontal, en el cual el sistema obtiene y libera recursos a demanda. Este autoescalado se realiza en la capa de aplicación, donde se encuentra la lógica del sistema. Por otro lado, existe un balanceador de carga que distribuye los pedidos entre las distintas instancias de servidores ejecutando en un momento dado. La arquitectura autoescalable también cuenta con una réplica a nivel de base de datos, mitigando posibles fallas en la base de datos principal. Esta réplica se encuentra siempre actualizada y disponible para tomar el rol de base de datos principal en caso de generarse un error o falla en la infraestructura.

2. **Arquitectura en múltiples centros de datos:**

   La Figura 2.7 muestra una arquitectura en la cual existe una réplica tanto a nivel de base de datos como de aplicación. Estas réplicas se encuentran ubicadas en distintas localidades geográficas, potenciando la tolerancia a fallos del sistema. Mediante este mecanismo, se mitiga una falla en la infraestructura de alguno de los centros de datos. La base de datos principal es replicada en un segundo centro de datos, y en caso de existir una falla en el centro de datos principal, la base de datos

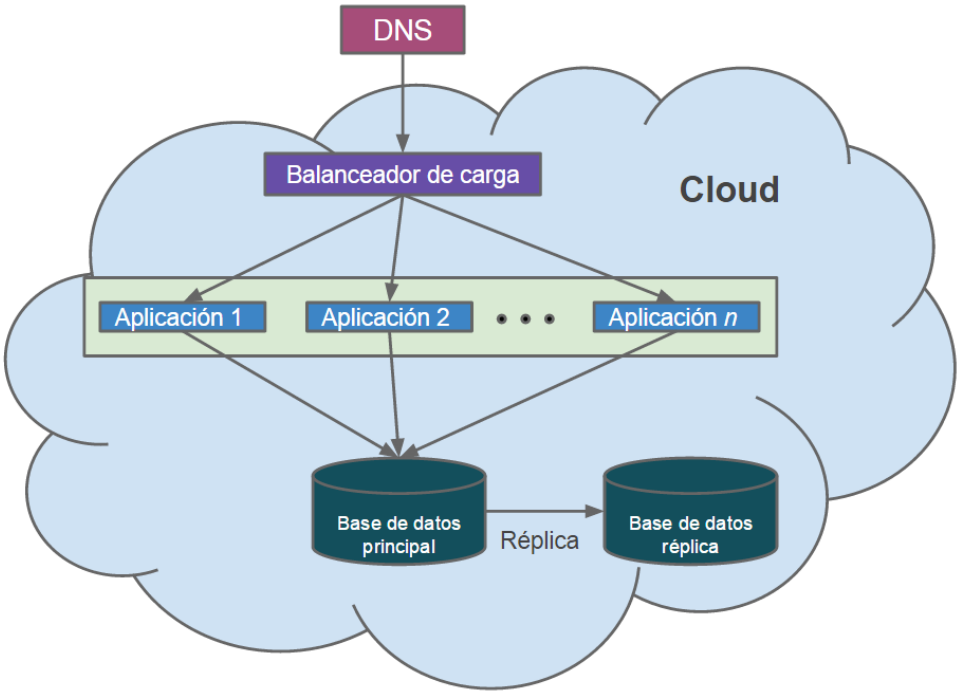

Figura 2.6: Arquitectura autoescalable.

réplica toma el rol de base de datos *principal*. Este tipo de arquitectura puede generar gastos extras en lo que refiere a la transferencia de datos entre un centro de datos y otro.

2.9. Costos de servicios en el cloud

Para acceder a los servicios brindados por el proveedor cloud es necesario abonar ciertos costos por la utilización de recursos, tráfico en la red entre regiones geográficas, etc. Se realizó un relevamiento de dichos costos, donde tanto AWS como Microsoft Azure, el cuál se detalla a continuación.

2.9.1. AWS

En Amazon, los precios se basan en la ubicación del servicio Amazon S3. Se provee de una capa de uso gratuito que permite comenzar a utilizar algunos servicios limitados de Amazon S3. Al registrarse, los clientes reciben 5GB de almacenamiento estándar, 20.000 solicitudes de tipo «GET», 2.000 solicitudes de tipo «PUT» y 15GB de transferencia de datos saliente al mes durante un año. Además, este proveedor brinda a sus usuarios una aplicación web denominada 'Calculadora mensual sencilla' que les permite realizar

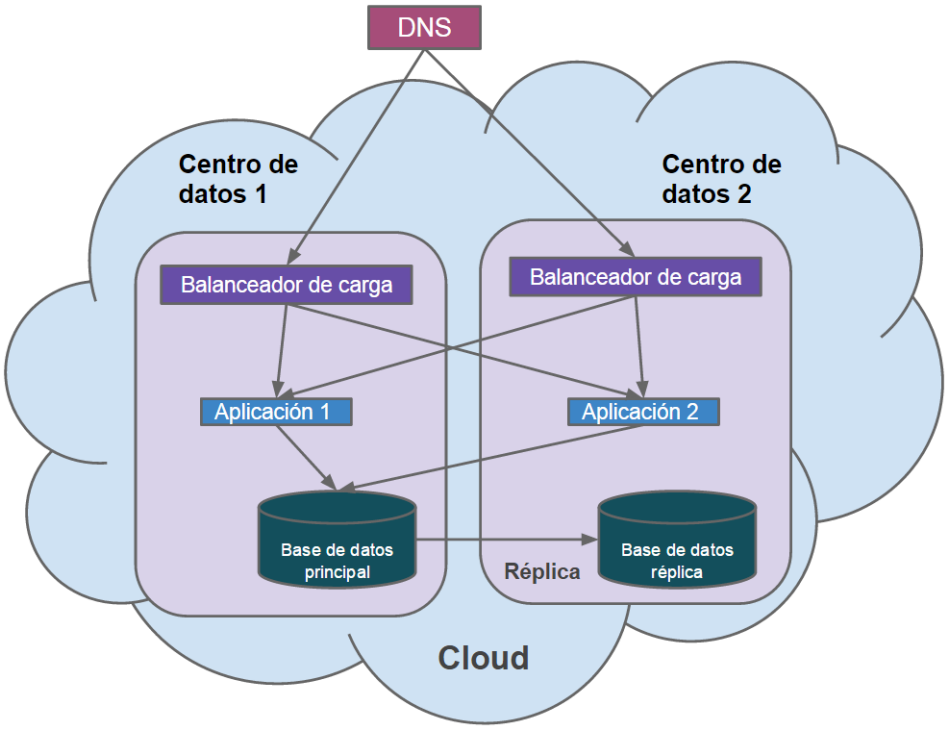

Figura 2.7: Arquitectura en múltiples centros de datos.

una estimación de su factura mensual. Como referencia se utiliza la región estándar de Estados Unidos y se muestran precios en dólares americanos en las tablas 2.2 a 2.4 [44].

2.9.2. Microsoft Azure

En este caso, los costos se evalúan de manera diferente, dependendiendo del tipo de servicio que se utiliza. De todas formas, al igual que en el caso de AWS, Azure proporciona un nivel mínimo de acceso a servicios de manera gratuita durante un período de prueba de treinta días, donde brinda a sus usuarios un bono de doscientos dólares. Adicionalmente, cada mes es posible utilizar servicios básicos de forma gratuita como la ejecución de hasta diez sitios web por región o la utilización en una cuenta de almacenamiento de hasta 999MB. Como referencia se utiliza la región estándar de Estados Unidos y se muestran precios en dólares americanos en las tablas 2.5 a 2.9. Además, en el sitio web de Microsoft Azure, se pueden observar los costos para otros servicios como el servicio de caché administrado, servicios móviles, *Service Bus* o bases de datos SQL [8].

Cuadro 2.2: AWS Precios según tipo de almacenamiento

	Estándar	**Redundancia Reducida**	**Poca frecuencia acceso a datos**
Primer TB/mes	$0.0300 por GB	$0.0240 por GB	$0.0100 por GB
Siguientes 49TB/mes	$0.0295 por GB	$0.0236 por GB	$0.0100 por GB
Siguientes 450TB/mes	$0.0290 por GB	$0.0232 por GB	$0.0100 por GB
Siguientes 500TB/mes	$0.0285 por GB	$0.0228 por GB	$0.0100 por GB
Siguientes 4000TB/mes	$0.0280 por GB	$0.0224 por GB	$0.0100 por GB
Más de 5.000TB/mes	$0.0275 por GB	$0.0220 por GB	$0.0100 por GB

Cuadro 2.3: AWS Precios según solicitud

	Precios
Solicitudes PUT, COPY, POST o LIST	$0.005 por cada 1000 solicitudes
Solicitudes de restauración y archivado en plan «Poca frecuencia acceso»	$0.05 por cada 1000 solicitudes
Solicitudes de eliminación	Gratis
GET y todas las demás solicitudes	$0.004 por cada 10000 solicitudes
Restauraciones de datos en Glacier	Gratis

2.10. Financiación en clouds públicos

AWS permite obtener una suscripción general donde se permite acceder a cualquiera de los servicios que provee AWS, abonando el monto correspondiente de forma mensual. Además de las suscripciones generales, proporciona servicios para agentes gubernamentales como FedRAMP. Estos servicios ofrecen una propuesta estandarizada para la evaluación de la seguridad, la autorización y el control continuo de productos y servicios en el cloud hasta un nivel moderado; particularmente, todas las agencias del gobierno de Estados Unidos pueden aprovechar los paquetes AWS Agency ATO, almacenados en el repositorio de FedRAMP, para evaluar AWS respecto a sus aplicaciones y sus cargas de trabajo, ofrecer autorizaciones de uso de AWS y trasladar cargas de trabajo al entorno de AWS. AWS también cuenta con planes de otorgamiento de cuentas para proyectos científicos y de investigación [43].

Cuadro 2.4: AWS Precios por transferencia de datos

	Precios
Transferencia ENTRANTE de datos a Amazon S3	
Todas las transferencias entrantes de datos	$0.000 por GB
Transferencia SALIENTE de datos de Amazon S3 a	
Amazon EC2 en la región Norte de Virginia	$0.000 por GB
Otra región de AWS	$0.020 por GB
Amazon CloudFront	$0.000 por GB
Transferencia SALIENTE de datos de Amazon S3 a Internet	
Primer GB/mes	$0.000 por GB
Hasta 10TB/mes	$0.090 por GB
Siguientes 40TB/mes	$0.085 por GB
Siguientes 100TB/mes	$0.070 por GB
Siguientes 350TB/mes	$0.050 por GB

Cuadro 2.5: Azure precios de instancias de tipo A de Web y Worker roles

	Precios por instancia
1 núcleo, 768MB de RAM	$0,02/hr
1 núcleo, 1,75 MB de RAM	$0,08/hr
2 núcleos, 3,5 GB de RAM	$0,16/hr
4 núcleos, 7 GB de RAM	$0,32/hr
8 núcleos, 14 GB de RAM	$0,64/hr

Cuadro 2.6: Azure precios de máquinas virtuales

	Precios por máquina
Máquinas Windows	
1 núcleo, 768 MB de RAM	$0,018/hr
1 núcleo, 1,75 MB de RAM	$0,074/hr
2 núcleos, 3,5 GB de RAM	$0,148/hr
4 núcleos, 7 GB de RAM	$0,296/hr
8 núcleos, 14 GB de RAM	$0,592/hr
Máquinas Linux	
1 núcleo, 768 MB de RAM	$0,018/hr
1 núcleo, 1,75 MB de RAM	$0,044/hr
2 núcleos, 3,5 GB de RAM	$0,088/hr
4 núcleos, 7 GB de RAM	$0,176/hr
8 núcleos, 14 GB de RAM	$0,352/hr

Cuadro 2.7: Azure precios de almacenamiento

	Con redundancia local	**Con redundancia de zona**	**Con georedundancia**
Blobs en bloques hasta 6000GB	$142,01/mes	$177,52/mes	$286,66/mes
Tablas y colas hasta 6000GB	$395,12/mes	-	$497,36/mes
Archivos hasta 6000GB	$240,00/mes	-	$300,00/mes

Cuadro 2.8: Azure precios ancho de banda

	Precio
Salida de Estados Unidos y Europa hasta 990GB	$85,70/mes
Salida en Asia Pacífico y Japón hasta 990GB	$135,93/mes
Salida en Brasil hasta 990GB	$178,29/mes

Cuadro 2.9: Azure precios clúster HDInsight

	4 nodos	**8 nodos**	**16 nodos**
1 núcleo, 768MB de RAM	$476,00/mes	$952,00/mes	$1.904,00/mes
1 núcleo, 1,75 MB de RAM	$952,00/mes	$1.080,00/mes	$3.808,00/mes
2 núcleos, 3,5 GB de RAM	$1.904,00/mes	$1.904/mes	$4.160,00/mes
4 núcleos, 7 GB de RAM	$2.040,00/mes	$3.808,00/mes	$7.616,00/mes
8 núcleos, 14 GB de RAM	$2.112,00/mes	$4.224,00/mes	$8.448,00/mes

Los tipos de suscripción de Azure se pueden agrupar en tres grandes categorías: promociones con recursos gratuitos, ofertas de bolsas de horas para desarrolladores y entorno de producción. En el primer tipo de suscripción existen dos grupos, uno para usuarios MSDN y otro general. Ambas suscripciones disponen de un conjunto de servicios gratuitos. En general, este tipo de suscripción se utiliza para análisis experimentales, aprendizaje o proyectos piloto. Las bolsas de horas para desarrolladores permiten

contratar algunos servicios adicionales a menor costo, «núcleo de desarrollo acelerado» («development accelerator core») y «desarrollo acelerado extendido» («developer accelerator extended»); la diferencia entre ambas radica en el número y tipo de recursos. Por último, el entorno de producción es una suscripción general donde se paga por lo que utiliza, pudiendo acceder a cualquier servicio proporcionado por Azure abonando una factura mensual. Al igual que AWS, Azure proporciona financiación y cuentas gratuitas para proyectos científicos y de investigación [27].

Capítulo 3

Descripción del problema a resolver

Este capítulo presenta una introducción al problema científico que propone abordar el proyecto. El análisis científico principal consiste en el abordaje de una realidad biológica, donde se pretende analizar ciertas transformaciones celulares mediante algoritmos matemáticos y simulaciones. En la sección 3.2 se especifica y detalla el enfoque del proyecto y en la sección 3.3 se describe la planificación general del mismo. Por su parte, el objetivo central del proyecto de grado está enfocado en el estudio del paradigma de computación distribuida en el cloud, por lo cual se presenta un análisis de trabajos relacionados con computación científica en el cloud en la sección 3.4.

3.1. Análisis de fluorescencia para develar el desarrollo embrionario

Esta sección describe el problema abordado en el proyecto. Se explican los mecanismos empleados para analizar células y moléculas, así como los componentes de software empleados para simular dichos mecanismos.

3.1.1. Fluorescencia

La fluorescencia es la capacidad de algunas sustancias de absorber luz a una determinada longitud de onda. En general esta absorción se lleva a cabo en el rango ultravioleta, y luego se produce la emisión de luz en una longitud más larga. Dicho de otra manera, las sustancias absorben fotones con una determinada energía y liberan fotones con menor energía. Este proceso se produce de forma casi inmediata; la luz es recibida y vuelta a emitir en millonésimas de segundo. A raíz de la inmediatez con la que se produce el proceso, se puede decir que la fluorescencia dura tanto como el estímulo, ya que cuando éste cesa, también cesa el fenómeno de fluorescencia. Ésta es

la principal diferencia con el fenómeno de fosforescencia, en el cual la luz absorbida se vuelve a emitir luego de transcurrido un cierto lapso de tiempo. También se le llama fluorescencia al fenómeno mediante el cual algunas sustancias son capaces de absorber otras formas de energía, como rayos X o rayos catódicos, y esta energía es liberada en forma de luz, de manera casi inmediata. El fenómeno de fluorescencia tiene múltiples aplicaciones, desde los tubos de luz fluorescentes que podemos ver habitualmente en hogares y oficinas, hasta técnicas de laboratorio para detectar antígenos y anticuerpos [22].

El fenómeno de fluorescencia puede definirse de la siguiente manera. Los electrones de un átomo (o varios átomos que conforman una molécula) se hallan orbitando a distintos niveles, donde cada nivel tiene determinada energía. Cuando la luz o rayos X llegan a los electrones que se encuentran en niveles de baja energía, éstos pasan a un estado de excitación, lo que produce que salten a una órbita de mayor energía. Este cambio en el nivel de energía ocasiona que el átomo pase a un estado inestable, por lo que luego el electrón deberá regresar a la órbita que le corresponde. Cuando esto sucede, emite la energía sobrante, la cual se traduce parcialmente, en las sustancias fluorescentes, en luz emitida. El resto de la energía se traduce en vibraciones de la molécula, es decir, en calor [15]. Las sustancias en las cuales se observa el fenómeno de fluorescencia se llaman fluoritas. Dentro de ellas se manifiestan diferentes capacidades fluorescentes, las cuales se pueden medir con la siguiente fórmula:

$$\Phi = \frac{CantidadFotonesEmitidos}{CantidadFotonesAbsorbidos}$$

Al resultado de esta relación se le llama rendimiento cuántico. El máximo rendimiento sería del cien por ciento, cuando todos los fotones absorbidos son reemitidos. Sustancias con un cociente de 0,1 (es decir que de 10 fotones que absorben sólo uno es reemitido) aún se consideran fluorescentes [18].

La generación de imágenes por fluorescencia permite visualizar moléculas más allá del límite de resolución de un microscopio, tal y como se puede ver en la Figura 3.1. La microscopía fluorescente es una técnica clave en el diagnóstico clínico (por ejemplo, en inmunología, patología, microbiología, citogenética) y en ambientes de investigación. La microscopía de fluorescencia confocal, en particular, se ha convertido en una herramienta esencial muy pertinente para el estudio de la dinámica estructural y molecular en células vivas. La fluorescencia habilita las técnicas de generación de imágenes como TIRF, FRET, FLIM, FLIP y FRAP. Esta herramienta es importante para identificar los minerales fluorescentes, contaminantes e impurezas en la ciencia de materiales, en la geología, en la inspección de semiconductores y en la protección ambiental.

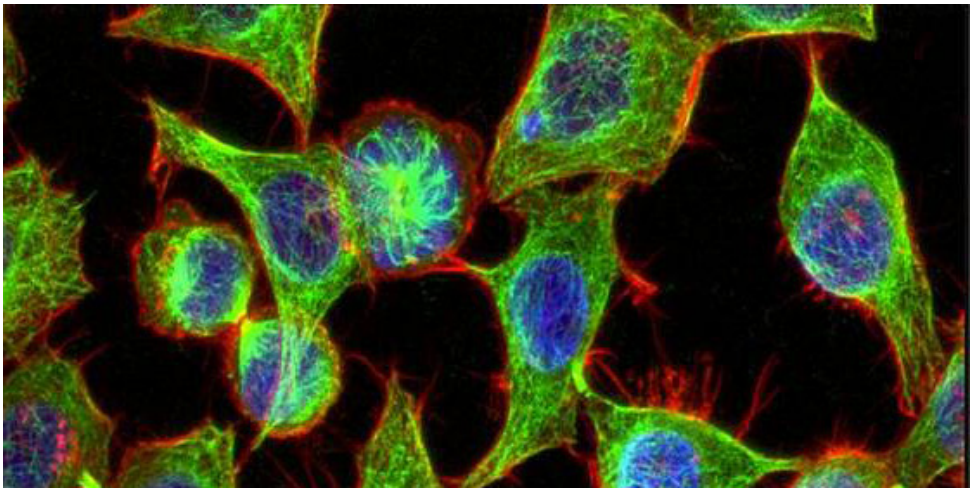

Figura 3.1: Fluorescencia en microscopía

3.1.2. Caso de Estudio

El análisis biológico tiene como objetivo descubrir la transformación de las diferentes células que componen un ser vivo, en cada una de las partes de su cuerpo, mediante la observación del resultado de fluorescencia. Es necesario tener siempre en cuenta el entorno y el tiempo, ya que ambos pueden afectar la velocidad y forma de desarrollo del organismo.

El análisis inicia con el estudio de un caso de baja complejidad para luego lograr generalizar los resultados y aplicarlos sobre diferentes seres vivos. Comienza observándose la evolución de las células de un insecto. Se pretende dilucidar cómo y de qué manera las células componen las distintas partes del cuerpo del insecto. Para ello, se realiza un seguimiento del movimiento de las moléculas dentro de la célula y se encuentra la dinámica asociada a este proceso. Al realizar el seguimiento de las dinámicas se puede averiguar qué está sucediendo y cómo se lleva a cabo la agrupación de células para cada parte del cuerpo.

Los recientes avances en técnicas de microscopía por fluorescencia y la capacidad existente para etiquetar proteínas con etiquetas fluorescentes en un ambiente celular, como por ejemplo las proteínas fluorescentes genéticamente codificadas, han constituido un cambio significativo en el modo de estudiar la biología de las células. Mientras los experimentos tradicionales de la bioquímica para analizar biomoléculas requieren aislamiento y la ge-

neración de un entorno artificial, estas nuevas tecnologías permiten a los científicos explorar los procesos biológicos de relevancia "in situ".

Desde que comenzó a utilizarse la microscopía para observar las células, se volvió evidente que la distribución de los componentes celulares, desde pequeñas proteínas hasta organelos, no es homogénea ni en espacio ni en el tiempo. Los intentos por comprender las reglas que gobiernan la organización intercelular y la respuesta de las células a estímulos específicos se han vuelto una prioridad. La espectroscopia por correlación de fluorescencia (*FCS* por sus siglas en inglés) [16] es extremadamente utilizada para obtener medidas cuantitativas del movimiento de las moléculas en células vivas [23]. En la Figura 3.2 se presenta un esquema de las medidas en la espectroscopia por correlación de fluorescencia.

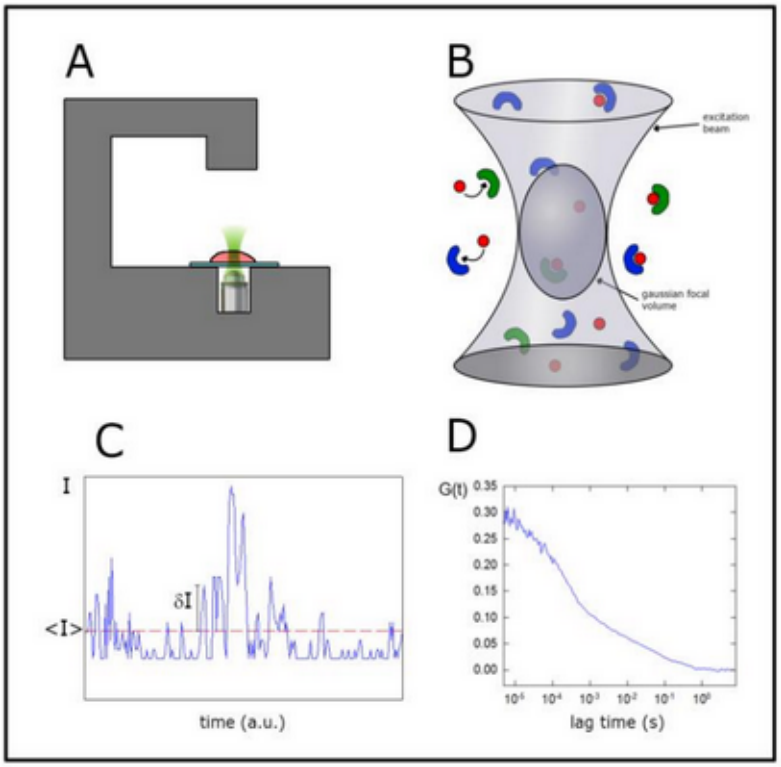

Figura 3.2: Correlación de fluorescencia

En la Figura 3.2, el cuadro A muestra una representación de la configuración experimental requerida para realizar este tipo de análisis por correlación de fluorescencia. La muestra (por ejemplo células) se coloca sobre la platina del microscopio, el cual puede ser confocal o un microscopio por "estímulo de dos fotones". El láser que realiza el estímulo es enfocado en un punto

limitado de difracción sobre la muestra y los fotones fluorescentes producidos en el pequeño volumen de observación (como se muestra en el cuadro B) son recolectados en función del tiempo. En el cuadro C se representa la intensidad de la señal de fluorescencia obtenida en el experimento FCS. Esta señal muestra las fluctuaciones que se producen con el movimiento de las moléculas fluorescentes hacia dentro y fuera del volumen de observación. Se puede demostrar que la amplitud de la fluctuaciones es inversamente proporcional al número de moléculas en el volumen de observación mientras que la duración está dada por la dinámica de dichas moléculas. Esta información se obtiene calculando la función de autocorrelación (ACF por sus siglas en inglés), como se muestra en la siguiente ecuación.

$$G(\tau) = \frac{< \delta I(t) \cdot \delta I(t+\tau) >}{< I(t) >^2}$$

Donde

- $It = I(t) - < I(t) >$ representa las fluctuaciones en la intensidad.
- $<>$ indica el promedio de tiempo.
- τ es un tiempo de retraso.

En el cuadro D se muestra una gráfica típica producto de aplicar la función ACF.

Ya que la técnica captura las fluctuaciones de fluorescencia producidas por el movimiento de cada una de las moléculas, es necesario reunir una cantidad considerable de puntos de datos para obtener información estadísticamente relevante sobre el proceso dinámico subyacente. Para modelos simples como el de difusión Browniana, el análisis ACF produce funciones analíticas a las que fácilmente se les pueden otorgar los datos experimentales para obtener parámetros relacionados con la movilidad de las moléculas. Desafortunadamente, muchos procesos dinámicos acerca de células no pueden ser interpretados con modelos tan simples y en muchos casos ni siquiera es posible obtener funciones analíticas a través de análisis teóricos para modelos más complejos [34]. Por este motivo, se utilizan técnicas de simulación para modelar escenarios más complejos.

3.1.3. Componentes de software

A escala celular, un número finito de moléculas interactúan dentro de complejos escenarios definidos por la célula y las membranas del organelo. Para poder simular eventos estocásticos celulares (como movimientos, interacciones, reacciones) con espacios realistas y utilizando tiempo y recursos de cómputo razonables, se deben utilizar técnicas de optimización numérica específicas [47].

En el caso de los sistemas biológicos típicos que utilizan este tipo de técnicas de optimización en conjunto con probabilidades de reacción de Monte

Carlo, es posible estudiar sistemas biológicos considerando su evolución durante amplios períodos de tiempo, desde milisegundos hasta minutos [26].

Mcell. La aproximación estándar para sistemas de reacción-difusión ignora la naturaleza discreta de los reactivos, así como el carácter estocástico de sus interacciones. Las técnicas basadas en la ecuación química maestra, tales como el algoritmo de Gillespie [21], asumen que en cada instante de tiempo las partículas se encuentran distribuidas de manera uniforme en el espacio. De forma de tener en cuenta tanto la distribución espacial completa de los componentes como el carácter estocástico de sus interacciones, se suele utilizar en los análisis una técnica basada en las dinámicas Brownianas.

El componente de software Mcell [26] se basa en un algoritmo orientado a eventos, denominado *función de las dinámicas de la reacción de Green* (GFRD), el cual utiliza funciones de Green combinando en un único paso la propagación de las partículas en el espacio con las reacciones entre ellas [51]. En el algoritmo GFRD, las partículas se mueven de forma difusa; se asume que si existe una reacción entonces sigue un proceso de Poisson y ocurre de manera inmediata. Esto significa que el evento de la reacción puede desacoplarse del movimiento difuso de la partícula. El paso de tiempo del algoritmo está determinado de forma tal que solamente se tienen en cuenta partículas individuales o pares de partículas, evitando reglas de reacciones más complejas.

Dada la naturaleza del proceso subyacente (rastreando moléculas individuales), es necesario simular un gran número de puntos de datos para obtener una simulación estadísticamente relevante. Dichos experimentos numéricos requieren tiempos de cómputo extremadamente elevados; cada molécula (punto de dato) debe ser rastreada en su movimiento, y cada interacción con otras moléculas (reacciones) debe ser modelada y resuelta para cada paso del tiempo simulado. El algoritmo se ejecuta de forma secuencial; obtiene como dato de entrada un archivo que representa el estado inicial de las moléculas, los tipos de las mismas y posibles interacciones. Luego comienza el procesamiento donde se realizan análisis complejos sobre el escenario inicial y se llevan a cabo las diversas interacciones para cada paso del tiempo. La salida de MCell será el estado final de las moléculas para cada instante de tiempo.

Las técnicas de computación de alto desempeño (HPC) cumplen con un rol fundamental a la hora de obtener la eficiencia computacional requerida para procesar volúmenes de puntos de datos considerablemente grandes y durante una cantidad de pasos de tiempo suficiente, de forma de captar el proceso biológico. El diseño e implementación de una aplicación específica que utilice HPC para analizar las fluctuaciones en la fluorescencia representa un desarrollo original e innovador [34].

FERnet El software FERnet simula el comportamiento de un microscopio al leer y escanear la información de entrada. Dicha entrada es la salida procedente de una ejecución de MCell. Al escanear esta información se realiza

un proceso de etiquetado de las moléculas que componen cada una de las células. Luego se obtiene la traza de fotones de fluorescencia, contando los fotones que fueron emitidos durante un pequeño lapso de tiempo. Al combinar estas dos herramientas se obtiene el resultado de fluorescencia buscado, intentando explicar qué sucede con las células al comparar los resultados con los modelos previamente existentes: difusión pura (*pure diffusion*), reacción dominante (*reaction dominant*), modelo híbrido (*hybrid model*), difusión eficaz (*effective diffusion*) y modelo completo (*full model*), cuya explicación va más allá del alcance de este proyecto.

3.2. Especificación del proyecto

El enfoque principal de desarrollo del proyecto consiste en el estudio, comprensión y aplicación del paradigma de computación distribuida en el cloud y su aplicación para la resolución de un problema de computación científica relacionado con el análisis del desarrollo embrionario. Se realiza un análisis de la realidad biológica sobre la que se basará la instrumentación de la aplicación en el cloud. A su vez, se realiza una investigación en el área de computación distribuida haciendo especial énfasis en computación en el cloud. Tras el estudio y entendimiento del paradigma, se estudian las distintas plataformas y proveedores de servicios sobre los cuales pueden implementarse aplicaciones distribuidas. Mediante un análisis exhaustivo se hace posible seleccionar una de estas plataformas para orientar el desarrollo.

Como ya se explicó en la sección anterior, existen dos componentes de software (MCell y FERnet) que permiten simular el proceso biológico de desarrollo embrionario. La realidad en cuestión tiene un alto grado de complejidad, como también lo tiene el cómputo que se realiza durante la simulación. Un usuario puede ejecutar una instancia de Mcell y la misma puede demorar desde unos pocos segundos hasta varias horas dependiendo de los parámetros de configuración de entrada. Por su parte, el software FERnet debe esperar hasta obtener la salida de una simulación de MCell que utilizará como entrada para su ejecución. Para el usuario, esto representa un problema ya que el resultado de las ejecuciones es muy importante y necesario para sus trabajos de investigación, pero se pierden valiosas horas de trabajo con los computadores prácticamente inutilizados mientras ejecutan dichas simulaciones. Por otro lado, si el usuario desea probar distintos conjuntos de parámetros debe ingresarlos manualmente y volver a lanzar la simulación una y otra vez.

A raíz del gran insumo de tiempo de configuración y cómputo que requieren las ejecuciones de MCell y FERnet, surge la necesidad de poder independizar el cómputo de estas aplicaciones de la computadora del usuario, a la vez que revalorizar las ejecuciones, brindando más poder y facilidad al usuario. Para esto, resulta ideal el paradigma de computación en el cloud

donde pueden alojarse desde pequeños componentes de software hasta grandes sistemas, en servidores distribuidos a lo largo del mundo, y hacer uso de recursos computacionales fundamentales, como el procesador y el espacio de almacenamiento, en los servidores remotos.

La idea principal del sistema consiste en permitir al usuario la ejecución remota del software MCell mediante la utilización de un portal web, optimizando los tiempos de espera para el usuario al usar servidores específicamente diseñados para aplicaciones de alto desempeño. El sistema permite el lanzamiento automático de la ejecución de FERnet una vez que se obtuvo la salida de MCell requerida y también permite al usuario la ejecución en paralelo de diversas configuraciones de MCell, mediante el ingreso simple y la configuración de un conjunto de parámetros de entrada. El sistema se complementa con una página de consulta de resultados, donde el usuario puede monitorear el estado de sus simulaciones y acceder a la salida de sus ejecuciones de forma permanente.

Uno de los principales objetivos del proyecto se centra en lograr una arquitectura que permita soportar los requerimientos funcionales descritos anteriormente y cumplir con requerimientos no funcionales deseables en todo sistema distribuido de alta calidad. Un aspecto fundamental de la arquitectura es que sea altamente escalable y permita un correcto funcionamiento con la ejecución de simulaciones concurrentes, utilizando los recursos disponibles bajo demanda provistos por el proveedor de cloud elegido. Se busca aprovechar las características del cloud para ejecutar un gran número de simulaciones en paralelo, permitiendo la recuperación ante fallos. Cabe destacar que la paralelización es un aspecto fundamental del proyecto. Se debe investigar el paradigma MapReduce, analizar su aplicación e integrarlo a la arquitectura en la plataforma seleccionada, para lograr realizar diversas simulaciones de forma simultánea y así obtener ventaja en relación a la ejecución de las aplicaciones en un ambiente local al usuario.

3.3. Planificación inicial del proyecto

Al inicio del proyecto se identificaron las fases más relevantes requeridas para llevar a cabo las tareas previamente descritas, de manera tal de lograr cumplir con los objetivos planteados. Para ello se creó un cronograma constituido por las fases que se detallan a continuación.

3.3.1. Cronograma

Primera fase La etapa inicial del proyecto consiste en el relevamiento, búsqueda de información, estudio del paradigma de Cloud Computing y análisis del problema de desarrollo embrionario. Este último punto comprende el estudio y la comprensión del problema biológico en cuestión, la familiarización con los productos de software previamente existentes (Mcell

y FERnet), así como la adaptación de dichas aplicaciones al entorno de trabajo elegido. A su vez, al finalizar esta etapa se seleccionan la plataforma y las tecnologías que serán utilizadas durante el resto del proyecto.

Segunda fase El objetivo de esta etapa es diseñar, definir y especificar la arquitectura de la aplicación. Para esto se tiene en cuenta la plataforma elegida en la etapa anterior y se evalúa qué características son viables para ser incluidas en el proyecto. También se realizan pruebas de rendimiento y escalado.

Tercera fase Esta fase consiste en la implementación de la aplicación sobre la plataforma cloud, realizando pequeñas modificaciones en la arquitectura en caso de ser necesario. También se deben realizar pruebas de correctitud y validación y se lleva a cabo un análisis experimental de desempeño.

Cuarta fase En esta última etapa se realiza un análisis de eficiencia computacional a la vez que se finalizan el producto de software y el informe final.

3.4. Trabajo relacionado

Esta sección provee una revisión de los trabajos recientes que han abordado la computación distribuida aplicada a problemas científicos en el cloud.

3.4.1. Computación distribuida en el cloud

Jakovits y Srirama (2013) [25] estudian la utilización del *framework* MapReduce para adaptar aplicaciones científicas al cloud sin perder eficiencia y escalabilidad. Los autores realizan una clasificación de algoritmos en cuatro categorías según su adaptabilidad al modelo y estudian el desempeño de MapReduce para la resolución de los problemas en cada una de ellas. Como principales resultados se obtiene que MapReduce se adapta bien a problemas simples llamados comúnmente *trivialmente paralelos*, donde se necesita una única ejecución de MapReduce para obtener la solución buscada. También se concluye que el modelo MapReduce no funciona correctamente para algoritmos complejos que involucran un alto número de iteraciones (sobre tareas del tipo MapReduce). Se estudia como alternativa para este tipo de algoritmos el *framework* Twister, que es especialmente diseñado para algoritmos iterativos. El estudio busca brindar herramientas y resultados a científicos que busquen adaptar sus problemas al cloud y puedan utilizar la clasificación propuesta para decidir qué *framework* se ajusta mejor a sus necesidades sin perder eficiencia y escalabilidad, además de sacar provecho a las características inherentes de los *frameworks* diseñados para el cloud, como la tolerancia ante fallos.

Richman, Zirnhelt y Fix (2014) [40] realizan un estudio sobre cómo cloud computing puede beneficiar largas ejecuciones de simulaciones. La clave consiste en la ejecución en paralelo, permitiendo realizar un análisis de paráme-

tros con la realización de barridos paramétricos. De esta forma, se obtiene una mejora en los tiempos de ejecución de extensas simulaciones de mayor porte permitiendo resolver problemas más complejos en menor tiempo. El beneficio del uso de cloud computing se deriva de la implementación de un caso de estudio donde se evalúa el ciclo de vida de gasto energético (*LEC*) de 1.080.000 escenarios de diseños de casas individuales en Toronto. Los autores implementan una solución en AWS S3 utilizando 24 máquinas virtuales donde se consigue disminuir los tiempos de ejecución para los 1.080.000 escenarios de 563 días en una única máquina con un procesador de 2.5GHz a 28 días, realizando una partición equitativa de datos donde cada bloque consume aproximadamente la misma cantidad de tiempo en ser procesado. Los resultados muestran que es posible implementar una arquitectura en el cloud que provea una reducción de tiempos en simulaciones, permitiendo el análisis de parámetros a gran escala y la optimización de proyectos, a la vez que se reduce la retroalimentación durante etapas iniciales de diseño en sistemas de simulación.

Pallickara, Pierce, Dong y Kong (2009) [35] plantean una plataforma de servicios web para la planificación de tareas tanto en clusters *grid* como en el cloud, llamada Swarm. La plataforma Swarm integra infraestructuras *grid* con clusters en el cloud para resolver problemas científicos específicos, permitiendo lanzar y monitorear tareas que ejecutan en los distintos ambientes. La plataforma está compuesta por varios componentes de software entre los que se encuentran Swarm-Grid y Swarm-Hadoop; Swarm-Grid lanza ejecuciones de tareas en un cluster del tipo *TeraGrid* y Swarm-Hadoop lanza tareas MapReduce en un cluster Hadoop conformado por varias instancias de AMI ejecutando en la infraestructura cloud de Amazon Web Services. Las tareas llegan a un componente llamado *Large Task Load Optimizer* que las clasifica y decide a qué componente de software asignarlas para su ejecución (grid o cloud). Se realizan experimentos ejecutando cantidades de tareas que varían entre 200 y 2000 tanto en grid como en cloud. Los resultados demuestran que el *overhead* generado por la generación de tareas en el cloud es mucho menor que en el clúster *grid*, ya que los recursos se asignan rápidamente. Cabe destacar que los datos en el ambiente cloud Hadoop con las tecnologías utilizadas son copiados desde el nodo maestro a cada nodo esclavo para cada ejecución en la arquitectura desarrollada. El trabajo concluye que la ejecución de una gran cantidad de tareas pequeñas presenta mejores tiempos de ejecución en el cloud, mientras que la ejecución de una reducida cantidad de tareas pero que requieren un mayor poder de cómputo y lectura de datos se adapta mejor al ambiente grid.

3.4.2. Computación distribuida en infraestructura clúster

Garcia, Iturriaga y Nesmachnow [20] presentan un trabajo de aplicación de computación paralela para resolver dos problemas científicos en la

plataforma de computación *grid* GISELA distribuida entre Europa y América Latina. Las aplicaciones científicas trabajan sobre información climática siendo una de ellas el procesamiento semi–automático de imágenes históricas climáticas (*DigiClima*) y la otra un paquete de software de simulación de dinámica de fluidos (*caffa3d.MB*), respectivamente. El paralelismo en *DigiClima* se consigue realizando una partición de datos siguiendo un modelo *maestro–esclavo*. Por otro lado, en *caffa3d.MB* se realiza una exploración de parámetros para la simulación utilizando un modelo *maestro–esclavo*. Los resultados muestran un *speed up* casi lineal para *DigiClima* concluyendo que con la obtención de más recursos de cómputo es posible obtener mayores beneficios del uso del *grid*; mientras que para *caffa3d.MB* se obtuvieron grandes mejoras en rendimiento, sobre todo en los casos de ejecución de simulación de mayor porte alcanzando un *speed-up* de hasta 15. Asimismo, se verificó que la ejecución distribuida en una arquitectura de *grid* optimiza los tiempos de ejecución y exploración de parámetros en grandes simulaciones.

Ortega y Nesmachnow [33] plantean un algoritmo paralelo para la ejecución de los sistemas Mcell y FERnet, encargados de resolver un problema de reacción-difusión en moléculas. La ejecución toma lugar en un ambiente *grid* con memoria distribuida utilizando técnicas de computación de alto desempeño. Para el sistema se utiliza una arquitectura *maestro-esclavo* implementada con la tecnología MPI sobre una infraestructura *grid*. El nodo maestro ejecuta la simulación Mcell y brinda los resultados de cada iteración a otro componente del nodo maestro a través de un *pipeline*, que a su vez lanza la ejecución de lectura de fotones en los nodos esclavos. Se realizan pruebas variando el número de microscopios y como resultado se obtiene una mejora importante en *speed-up* para las simulaciones con mayor cantidad de microscopios, mientras que en el caso de menor cantidad de microscopios la tasa de *speed-up* se ve limitada por la velocidad de generación de datos de Mcell a procesar por los nodos esclavos. Los resultados muestran una obtención de mejores tiempos de ejecución en simulaciones de gran porte (dada la cantidad de datos a procesar) en arquitecturas distribuidas.

En la misma línea de trabajo, Da Silva, Nesmachnow, Geier y Mocskos [14] presentan un análisis experimental aplicando un enfoque grid/cloud para la resolución de diferentes escenarios de análisis de fluorescencia con microscopia mediante la ejecución de los componentes de software biológicos MCell y FERnet. Se plantea como plataforma una infraestructura grid voluntaria utilizando el *middleware* OurGrid, donde los usuarios prestan sus recursos ociosos para realizar ejecuciones que requieren alto poder de cómputo. OurGrid también puede ser utilizado como middleware en infraestructuras cloud. En el trabajo se utiliza la técnica de distribución de dominio, particionando el conjunto de modelos en un flujo de trabajo y distribuyendolo en los recursos computacionales disponibles. Se realizan pruebas en una infraestructura heterogénea de recursos informáticos (residentes en Argentina, Brasil, México y Uruguay) y se ejecutan seis modelos de simulación

con distintos niveles de complejidad, modificando el paso de tiempo de la simulación para cada uno de ellos, obteniendo 24 posibles ejecuciones o instancias. Se ejecutan los modelos en una infraestructura local (UdelaR, en Uruguay) para comparar tiempos de ejecución. Los resultados muestran que la ejecución de 24 instancias de manera serial toma aproximadamente 10,81 horas mientras que la ejecución en la infraestructura grid toma aproximadamente 2,78 horas, obteniendo un speed up que ronda el valor 4. El trabajo propuesto concluye que el uso de una infraestructura grid voluntaria es un método efectivo para la ejecución de las simulaciones biológicas complejas.

3.4.3. Resumen

Los trabajos relacionados descritos en esta sección muestran resultados alentadores en la ejecución de sistemas científicos en arquitecturas distribuidas de tipo grid/cloud. También promueven el uso de modelos de programación para la ejecución en paralelo de tareas como MapReduce para obtener mejoras en los tiempos de ejecución en sistemas de modelado de simulaciones estocásticas, así como sistemas científicos con gran poder de cómputo en general. El proyecto de grado actual busca seguir estas líneas de trabajo y aportar una arquitectura distribuida altamente escalable y tolerante a fallos en una infraestructura cloud pública, que permita el correcto funcionamiento de sistemas de simulación de gran porte.

Capítulo 4

Arquitectura del Sistema

El objetivo de este capítulo consiste en brindar una descripción detallada de la arquitectura del sistema. La arquitectura diseñada se construyó de forma tal que cumple con todos los requisitos no funcionales que se plantearon al inicio del proyecto, a la vez que se enriquece y adapta respecto a las posibilidades tecnológicas que provee Microsoft Azure, la plataforma utilizada. Se presentan cada uno de los módulos que componen la arquitectura analizando ventajas y desventajas que presenta cada uno, y se describe en detalle cómo se unen y comunican construyendo la infraestructura general de la aplicación.

4.1. Elección de plataforma de desarrollo

Se decidió indagar con mayor nivel de detalle en dos de las tres plataformas propietarias de cloud más reconocidas: Microsoft Azure y AWS, dejando por fuera el estudio de Google App Engine.

Al comenzar con el estado del arte sobre cloud computing y las diferentes plataformas cloud, se encontró una extensa y exhaustiva documentación sobre AWS: tutoriales, foros, etc., dejando clara evidencia de lo implantada que está la plataforma en el ambiente cloud. No fue así, por lo menos al comienzo, para Microsoft Azure y es por esta razón que se dedicó mucho tiempo a estudiar los distintos servicios y capacidades provistos por AWS. Cuando se realizó un análisis de ambiente de desarrollo, surgió el inconveniente de no poseer una cuenta con la cual desplegar los componentes en el cloud de Amazon, pero sí existía la posibilidad de desarrollar las aplicaciones utilizando Eucalyptus y posteriormente migrar al cloud de AWS. Eucalyptus parecía ser la opción a utilizar, pero surgió un inconveniente al no existir soporte, a la fecha de la investigación, para el módulo de Apache Hadoop en AWS, Elastic Map Reduce. Este punto fue decisivo a la hora de elegir la plataforma, ya que se debía utilizar directamente el *framework* de Apache Hadoop y luego migrarlo a AWS sin poder sacar provecho de EMR

(como la integración a otros servicios de AWS, etc.). Además, para el uso directo de la plataforma AWS se debía solicitar una beca para la obtención de una cuenta destinada a investigación, cuyo plazo de inscripción era muy avanzado el proyecto, por lo que no se tendría una respuesta inmediata.

Por otro lado, Microsoft Azure, más recientemente difundido en el mundo cloud, disponía de una cuenta gratuita por dos meses con permisos limitados pero los suficientes para comenzar a armar un prototipo. Surgió la posibilidad de asistir a un curso de capacitación de Microsoft Azure en Buenos Aires y posteriormente en Montevideo (*Microsoft Azure for Research Training, Universidad de la República, 2014*) y así aprender sobre los componentes y servicios principales existentes en la plataforma. Dicho curso se complementaba con la obtención de una cuenta gratuita por estudiante con un vencimiento de 6 meses, teniendo esta cuenta por estudiante mayores privilegios que la cuenta brindada por defecto en la plataforma. Se comenzó entonces una investigación sobre las capacidades de Azure y fue posible encontrar material en abundancia incluyendo tutoriales, ejemplos de aplicaciones, etc. Era posible por tanto trabajar de manera gratuita con Microsoft Azure hasta que se obtuviera una cuenta definitiva destinada a investigación y con mayor poder de recursos en el cloud. Se planteó el proyecto a estudiar y se pidió una cuenta con el porte necesario a *Microsoft Azure for Research* destinada a proyectos de investigación; la misma fue brindada en un rango de 3 meses aproximadamente, siendo el tiempo justo al momento de probar estructuras de mayor porte, mayor poder de cómputo, etc.

Al indagar más profundamente en la plataforma de Azure, se decidió comenzar por el aspecto que hizo tomar la decisión de dejar AWS de lado en primera instancia: la implementación de Apache Hadoop dentro de Microsoft Azure. HDInsight implementa Apache Hadoop y es un servicio maduro y completamente integrado a la plataforma de Azure, así como fácil de utilizar y acceder, tanto por consola (Microsoft Azure Powershell) como desde el ambiente de desarrollo. Se realizó también un estudio del resto de los servicios disponibles para la implementación para los restantes componentes de la arquitectura, y se encontró una amplia gama de opciones con distintos niveles de libertad en la configuración (tanto IaaS como PaaS). Fueron de mayor interés los Cloud Services, un servicio PaaS fácilmente configurable y de fácil utilización y aplicación al problema planteado en el proyecto actual. Fue entonces cuando se decidió elegir a Microsoft Azure como plataforma de desarrollo del proyecto de integración de aplicaciones biológicas a la computación distribuida en el cloud.

4.2. Descripción general de la arquitectura

La mayoría de las aplicaciones científicas se pueden describir como procesamientos por lotes (*batch*) que se ejecutan utilizando un alto consumo de recursos de CPU y memoria RAM, sin la intervención de un usuario. Los componentes de software que se buscan adaptar a un entorno en el cloud, MCell y FERnet, son del tipo de aplicaciones de procesamiento por lotes. Los sistemas de procesamiento por lotes presentan ciertas características como lectura de un archivo inicial, procesamiento del archivo para finalmente realizar un post-procesamiento con la obtención de archivos de salida almacenando el resultado final. Al cumplirse las características recién descritas, los sistemas MCell y FERnet se pueden ejecutar de manera paralela de forma totalmente independiente, siendo por lo tanto el problema planteado de la categoría *trivialmente paralelo*, donde se requiere mínima o ninguna comunicación entre las tareas paralelas para obtener el resultado final.

Buscando contemplar las características inherentes del tipo de sistemas con el que se iba a trabajar, se buscó definir una arquitectura donde se le permitiera al usuario independizarse de la ejecución de las simulaciones y la posibilidad de obtener los resultados en cualquier momento futuro. Como se describió en el capítulo 3, se busca utilizar por tanto las características intrínsecas del cloud para obtener una arquitectura donde sea posible ejecutar una amplia cantidad de instancias de parejas Mcell-FERnet de manera paralela y así obtener ventaja en relación a la ejecución de estos sistemas en un ambiente de ejecución local al usuario.

El flujo buscado en la aplicación se describe a continuación:

1. El usuario ingresa al sistema mediante un portal web, donde es posible especificar los parámetros con los que se desea ejecutar MCell y posteriormente FERnet, dando la posibilidad de ingresar un rango de valores para ciertos parámetros que generarán múltiples ejecuciones de la simulación. El sistema retorna al usuario un identificador con el cual se le permite monitorear y obtener los resultados una vez finalizada la ejecución de las simulaciones.
2. Un rol de trabajo se encarga de obtener los parámetros y crear los archivos de entrada para las diferentes simulaciones, almacenándolos para el acceso de otros roles.
3. Un segundo rol de trabajo toma los archivos de entrada para configurar y ejecutar la simulación en un clúster HDInsight, en el cual, mediante la utilización del modelo MapReduce se ejecutan las diferentes simulaciones en paralelo. Este rol monitorea y almacena metadatos del estado de las tareas en ejecución para poder proporcionar información al usuario durante el proceso.
4. El clúster HDInsight se encarga de ejecutar las distintas simulaciones y crea archivos comprimidos con las salidas de FERnet que son

almacenadas para acceso y retribución al usuario.

5. El usuario puede acceder en cualquier momento al sitio web y, utilizando su identificador, corroborar el estado de las tareas que lanzó y finalmente obtener los archivos resultantes de las distintas simulaciones.

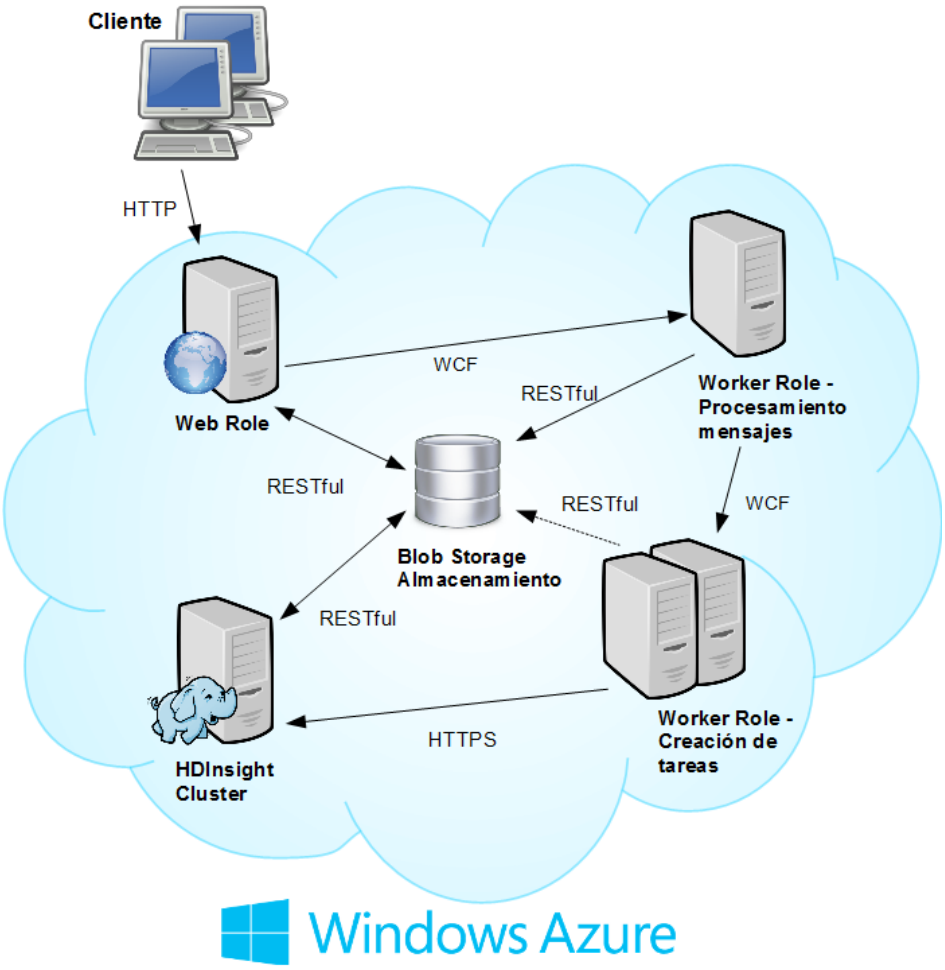

Figura 4.1: Componentes principales de la arquitectura propuesta.

Buscando cumplir con este flujo se especifica en la Figura 4.1 los componentes principales de la arquitectura propuesta y la interacción del usuario con el entorno de ejecución. El usuario accede mediante un sitio web a una interfaz amigable que le permite indicar algunos de los parámetros utilizados para la ejecución de Mcell y FERnet; esta información es enviada luego desde el sitio web a un rol de trabajo (*Message Processing*) realizando un llamado a un servicio de envío de mensajes asíncronos, *Windows Communication*

Foundation (WCF), expuesto por este rol. Cuando el rol de trabajo recibe el mensaje, crea los archivos de entrada para las simulaciones en paralelo y los almacena en la cuenta de almacenamiento creando *blobs* para cada uno de ellos. Por cada archivo de entrada para MCell generado, el rol realiza una llamada a un servicio WCF expuesto por otro rol de trabajo (*Job Creation*) encargado de crear las tareas que son ejecutadas en el clúster HDInsight.

Después de crear la tarea y ponerla en ejecución en el clúster HDInsight, el rol *Job Creation* monitorea la ejecución de cada tarea almacenando el estado de cada una de ellas para futuras consultas del usuario en la cuenta de almacenamiento. Por otro lado, las tareas se ejecutan en el clúster HDInsight y los resultados y archivos de control son almacenados bajo la estructura de *blobs*.

El usuario puede, en cualquier momento, solicitar información sobre sus ejecuciones utilizando un identificador brindado al comenzar las simulaciones. De esta manera, el sitio web controla los archivos actualizados por el rol de trabajo Job Creation, presentando esta información al usuario en una interfaz amigable.

4.2.1. Almacenamiento

El almacenamiento es el componente central de la arquitectura, ya que todos los demás componentes se comunican con él en algún momento del flujo. Por lo tanto el almacenamiento se cataloga como el cuello de botella de la arquitectura propuesta.

Se utiliza una cuenta de almacenamiento Azure (*Azure Storage Account*) y dentro de los tres servicios brindados por la cuenta, se hace especial uso de los *blobs* como formato de almacenamiento de información. Todos los accesos a la cuenta de almacenamiento, tanto de consulta como de creación y eliminación de datos, se realizan mediante servicios RESTful, liberando al componente de almacenamiento de la responsabilidad de mantener estados, etc.

Para mitigar el posible único punto de fallo del sistema, Azure presenta un almacenamiento georedundante manteniendo copias de la cuenta de almacenamiento en seis lugares diferentes, tres de ellos geográficamente distantes con una distancia de al menos 600km, por consiguiente, la probabilidad de fallo de todas las instancias de almacenamiento simultáneamente es casi nula.

4.2.2. Comunicación entre componentes

Pruebas iniciales con MASB

Inicialmente se propuso utilizar el servicio *Microsoft Azure Service Bus* (MASB) brindado por la plataforma Azure para la comunicación entre componentes, basado en cola de mensajes. Este servicio proporciona una in-

fraestructura organizada, segura y disponible para la comunicación general entre componentes ejecutándose en la plataforma Azure, la distribución de eventos a gran escala, así como la publicación de servicios, etc.

MASB es una herramienta de mensajería asincrónica y desacoplada de los componentes que desean comunicarse, asegurando la llegada de los mensajes, brindando así un servicio de mensajería confiable. Esta tecnología presenta características deseables para el proyecto por lo que se decidió utilizar MASB para la implementación de la comunicación entre los roles, principalmente su servicio denominado *Service Bus Queue*, utilizando colas de mensajes. Las primeras pruebas de la tecnología aplicada a la arquitectura diseñada no dieron buenos resultados, ya que el *overhead* del envío y recepción de mensajes resultó muy grande. Ocurrían con regularidad problemas intermitentes a la hora de establecer la conexión. Si bien siempre se lograba establecer la conexión, en ocasiones ocurría un error de forma repetida (en la Figura 4.2 se muestra el registro del error) donde el mismo *framework* debía reintentar la conexión, en algunos casos esto provocó que el establecimiento de la conexión tomara más de un minuto.

Se realizó una pequeña investigación en un intento por resolver este problema y se encontró que era un factor común que ocurría entre los desarrolladores de Azure. También se halló que Windows Azure estaba trabajando para el lanzamiento de una nueva versión de MASB donde se mejorarían el manejo de errores y los mecanismos de reintentos, pero la liberación de esta nueva versión aún no se había realizado lo que se convirtió en un impedimento [9]. En cuanto al envío y recepción de mensajes los tiempos de respuesta encontrados varían entre 5 y 20 segundos mientras que en WCF fueron ampliamente mejores recibiendo la respuesta de forma inmediata, es por ello que finalmente se decidió utilizar la tecnología Windows Communication Foundation para estas comunicaciones específicas.

```
Microsoft.ServiceBus.Messaging.MessagingCommunicationException: The socket connection was aborted.
  at System.ServiceModel.Channels.SocketConnection.HandleSendAsyncCompleted()
  at System.ServiceModel.Channels.SocketConnection.OnSendAsync(Object sender, SocketAsyncEventArgs
  --- End of inner exception stack trace ---
  at System.Runtime.AsyncResult.End[TAsyncResult](IAsyncResult result)
  at System.Net.Security._SslStream.WriteCallback(IAsyncResult transportResult)
  --- End of inner exception stack trace ---
  at System.Net.Security._SslStream.EndWrite(IAsyncResult asyncResult)
  at System.ServiceModel.Channels.StreamConnection.EndWrite()
  --- End of inner exception stack trace ---
```

Figura 4.2: Error establecimiento conexión MASB

Tecnologías utilizadas: WCF y REST

La comunicación entre componentes de la arquitectura se da en todos los casos utilizando tecnologías web 2.0; la comunicación mediante roles se

realiza a través de *Windows Communication Foundation* (WCF), mientras que la comunicación entre los distintos componentes de la arquitectura y la cuenta de almacenamiento se realiza utilizando servicios RESTful.

WCF. WCF es un *framework* que permite la creación de aplicaciones orientadas a servicios que permite enviar mensajes asincrónicos de un punto de acceso a otro, mientras se cumplan ciertos criterios definidos en el contrato de servicio. Un punto de acceso puede ser un cliente o servidor que solicita o envía información a otro. Es posible enviar mensajes simples, como caracteres o XML, así como más complejos, por ejemplo datos binarios. Entre las características principales de WCF se destacan [42]:

- WCF provee múltiples patrones de mensajes. El más común de ellos es el pedido/respuesta donde un punto solicita información a otro. Luego existen otros patrones como mensajes en un solo sentido, donde se envía información pero no se solicita ninguna respuesta. También existe un patrón más complejo donde los dos puntos establecen una conexión por la cual se envían mensajes de ida y vuelta.
- WCF se basa en contratos de datos (*Data Contracts*) donde se especifica el formato que tendrá la información enviada, se define una clase con los atributos deseados y correctamente marcados (con los llamados *tags*). Una vez creada la clase que representa los datos a enviar y/o recibir, el servicio automáticamente genera los metadatos que permiten corroborar que el cliente envía los tipos de dato deseados.
- WCF admite intercambio de mensajes confiable utilizando sesiones implementadas sobre mensajería *WS-Reliable* y *MSMQ* (*Microsoft Messaging Queue*, un servicio de .NET para el manejo de mensajes recuperables). WCF garantiza que un mensaje es recibido por más que ocurran errores a nivel de protocolo de transporte y permite independizar al origen del destino, garantizando la comunicación entre ambos agentes aunque alguno de ellos no se encuentre disponible. Para implementar este servicio WCF se basa en el uso de colas donde se almacenan los mensajes en estados intermedios del proceso de comunicación.

RESTful. *Representational State Transfer* (RESTful) es una arquitectura de servicios web basada en recursos. Los recursos representan datos y funcionalidades y son accedidos utilizando una URI (*Uniform Resource Identifier*), que típicamente es un enlace (*link*) en la web. Esta arquitectura de servicios conduce a propiedades deseables como buen rendimiento, escalabilidad y modificabilidad, características deseables en una arquitectura distribuida. En un servicio REST los recursos son manipulados mediante un conjunto simple y bien definido de operaciones. A su vez, este tipo de arquitecturas fomentan una arquitectura global de la aplicación de *cliente/servidor* y está especialmente diseñada para utilizar protocolos de comunicación sin estado, normalmente HTTP [32].

Azure provee una interfaz de programación de aplicaciones (API) de acceso al servicio de almacenamiento con *blobs* utilizando llamados a servicios REST mediante HTTP, definiendo una lista de funciones sobre los *blobs* entre las que se encuentran: listar contenedores, obtener *blob*, eliminar *blob*, asignar propiedades a un *blob*, etc. A continuación se presenta un ejemplo de URI donde es posible obtener un *blob* realizando un pedido GET mediante HTTP [2]:

```
https://myaccount.blob.core.windows.net/mycontainer/myblob
```

La URI especificada obtiene el archivo almacenado en el blob denominado *myblob* que se encuentra dentro un contenedor llamado *mycontainer* correspondiente a una cuenta de almacenamiento Azure de nombre *myaccount*.

Balance de carga. Para la comunicación entre nodos que utilizan WCF, se implementó un algoritmo de balanceo de carga descrito en detalle en el capítulo 5, el cual redistribuye los pedidos entre las distintas instancias de los roles tomando en cuenta métricas de rendimiento como porcentaje de uso de CPU y memoria disponible en un periodo de tiempo especificado.

4.3. Cloud Service

Como se explicó en el capítulo 2, los *Cloud Services* de Azure son un servicio que entra en la categoría *PaaS*, donde el usuario tiene un mayor nivel de control y configuración sobre los componentes pero deja como responsable de los aspectos de fiabilidad y administración a la plataforma de Microsoft Azure.

La principal diferencia entre los roles web y de trabajo en un *Cloud Service* es que el rol web posee por defecto una instancia de IIS (*Internet Information Services*) en ejecución que permite desplegar sitios web dejándolos accesibles a los usuarios. El rol de trabajo también puede poseer una instancia de IIS en ejecución, pero no viene configurado por defecto. Ambos roles utilizan máquinas virtuales con Windows Server para instanciarse. Dentro de un *Cloud Service* puede existir una gran cantidad de instancias de cada rol que son administradas por la plataforma de Azure. Por ejemplo, si una instancia falla y queda deshabilitada, la plataforma la reinicia automáticamente evitando la intervención de un administrador, a no ser que el error generado no se pueda solucionar de esta manera en cuyo caso el administrador debe intervenir.

Por otro lado, se aprovecha la alta escalabilidad de *Cloud Services*, ya que las instancias que lo componen pueden ser escaladas horizontalmente de manera configurable, monitoreando métricas como porcentaje de CPU y uso de memoria, como muestra la Figura 4.3. En la Figura 4.3 se puede observar

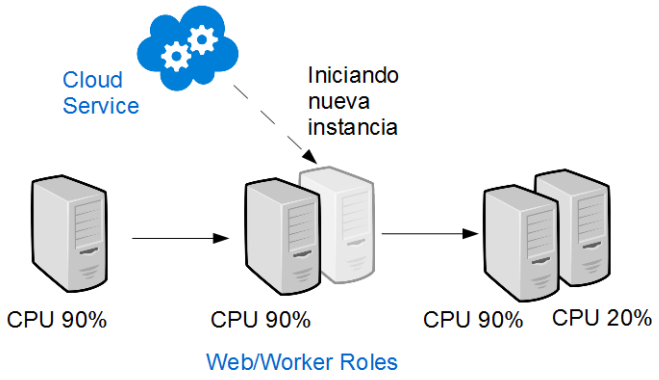

Figura 4.3: Escalado horizontal *Cloud Service* por métrica de uso de CPU.

inicialmente un rol con un porcentaje de uso CPU del 90 %. Suponiendo que en la configuración del Cloud Service especifica un porcentaje óptimo de CPU entre 60 % y 80 %, la plataforma de *Cloud Service* detecta un nivel de uso de CPU superior al deseado, y por tanto, crea y pone en ejecución una nueva instancia del rol. Se obtienen finalmente dos instancias en ejecución donde una de ellas posee un porcentaje de uso de CPU de 90 % y la otra 20 %.

Cuando sucede que todas las instancias de un rol superan el umbral definido en la configuración, se crean por parte del *framework* nuevas instancias a demanda y se apagan instancias si los recursos se encuentran ociosos, evitando así generar costos extras por recursos inutilizados. Para realizar estas acciones Azure implementa las máquinas virtuales que instancian a los roles sin estado. Este mecanismo permite poner en ejecución a las máquinas virtuales de manera rápida para crear nuevas instancias de roles o transportarlos de una ubicación a otra. Es importante por tanto, al momento de implementar estos roles, mantener los estados guardados en el almacenamiento para que no se pierdan. La escalabilidad es un punto crítico en el uso eficiente del cloud, permitiendo aprovechar todas las características que no son posibles adquirir en una arquitectura local [19].

Se decidió utilizar el servicio de *Cloud Services* como herramienta de construcción de los componentes de arquitectura que interactúan con el almacenamiento y el clúster HDInsight, dada su alta escalabilidad, fiabilidad y facilidad de configuración personalizada, con bajos costos de administración. Este servicio soporta grandes cantidades de usuarios sin requerir demasiada administración y brindando una tasa de fallo de la plataforma inferior a una arquitectura en un ambiente local dada la gran escalabilidad que presenta.

Se definen entonces tres tipos de roles dentro del Cloud Service como se puede observar en la Figura 4.4: un rol web encargado de la interacción con el usuario y dos roles de trabajo, uno encargado de la creación de archivos

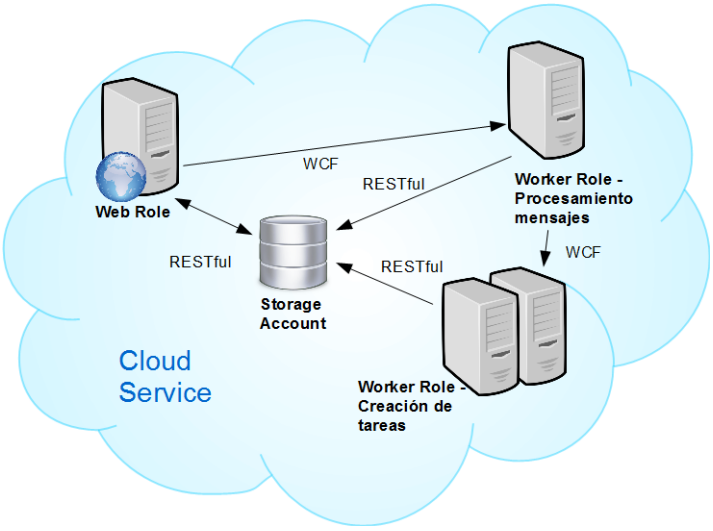

Figura 4.4: Componentes que conforman el *Cloud Service* definido.

de entrada para la simulación y otro encargado de ejecutar y monitorear las distintas simulaciones biológicas, así como de actualizar los metadatos en el almacenamiento para consulta del usuario a través del rol web. Los componentes que conforman al *Cloud Service* definido se describen a continuación.

4.3.1. Rol web

El rol web cumple un papel fundamental al ser el punto de acceso a la aplicación por parte del usuario. Se utiliza un rol web para este componente ya que posee una instancia de IIS en ejecución, permitiendo albergar el sitio web del proyecto. Este sitio web recibe los datos de entrada del usuario para ejecutar la simulación de MCell y FERnet y posee un *back-end* dentro del *Web Rol* que se encarga del acceso al *Blob Storage* así como de la comunicación con el resto de los roles mediante WCF. Para el acceso al almacenamiento se utilizan como credenciales certificados X.509 como se explicará con más detalle en el capítulo 5. El acceso al almacenamiento por parte del rol web se debe a que éste rol se encarga de asignar un identificador al usuario que puede ser utilizado posteriormente para monitorear el estado de sus tareas y obtener los resultados de las diferentes simulaciones. Este tipo de roles soporta una amplia cantidad de clientes debido al escalado horizontal brindado por Azure. El contrato de servicio de Azure especifica que para todas las máquinas virtuales orientadas a Internet que tengan dos o más instancias implementadas en el mismo conjunto de disponibilidad, se garantiza conectividad externa al menos durante el 99,95 % del tiempo [3].

4.3.2. Rol de trabajo Message Processing

El rol de procesamiento de mensajes (*Message Processing*) se implementa a través un rol de trabajo. Su objetivo principal es interpretar la entrada del usuario y realizar un barrido paramétrico sobre los distintos parámetros ingresados para crear los diferentes archivos de entrada para las ejecuciones de MCell y FERnet. Estos archivos son almacenados en el *Blob Storage* para acceso futuro. El rol de trabajo exhibe un servicio mediante un punto de acceso que recibe llamados WCF con los mensajes enviados desde el rol web; este aspecto será especificado con más nivel de detalle en el capítulo 5. Cuando el rol recibe un llamado del rol web, crea el archivo de entrada para MCell y lo almacena en el *Blob Storage* para después realizar mediante WCF un llamado a un punto de acceso expuesto por el rol de trabajo *Job Creation*. WCF garantiza que el mensaje llega al destinatario, por lo que luego de enviado el mensaje, el rol queda en estado ocioso esperando recibir más llamados del rol web.

4.3.3. Rol de trabajo *Job Creation*

El rol de creación de tareas (*Job Creation*) se implementa al igual que el *Message Processing* mediante un rol de trabajo, siendo su principal objetivo tomar uno de los archivos MDL (formato de archivos de entrada de MCell) generados y configurar una tarea MapReduce para generar su ejecución en el clúster HDInsight.

La razón por la cual el rol *Job Creation* se encuentra desacoplado del *Message Processing* subyace en que posteriormente a poner en funcionamiento la tarea en el clúster, el rol *Job Creation* monitorea la ejecución de la tarea en el clúster HDInsight, actualizando metadatos que sirven para informar del estado de la ejecución al usuario. Si los dos roles *Message Processing* y *Job Creation* se encontraran unificados y este hipotético rol ejecutara una tarea en el clúster HDinsight, el mismo consumiría recursos obteniendo y actualizando metadatos sobre la tarea en ejecución, generando un posible cuello de botella si se diera una repentina ráfaga de usuarios queriendo utilizar la aplicación. Este posible cuello de botella se mitiga separando los dos roles, de manera que el rol *Message Processing* se encuentre disponible la mayoría del tiempo para recibir nuevos mensajes del usuario y la disponibilidad del rol *Job Creation* se apoye fundamentalmente en el autoescalado horizontal de *Cloud Services*. De esta manera, se genera un *overhead* menor de tiempo de creación de nuevas instancias y el rol *Message Processing* responde de manera automática al pedido del usuario mejorando la experiencia del mismo.

4.4. Clúster HDInsight

Como se explicó en el capítulo 2, HDInsight es la implementación de Azure para Apache Hadoop y YARN; el sistema de archivos de Hadoop se llama Hadoop Distributed File System (HDFS) y en Microsoft Azure se mapea a un contenedor de un *Blob Storage* determinado. Se puede apreciar la arquitectura general de HDInsight en la Figura 4.5, donde el clúster se conforma por un nodo maestro y varios nodos esclavos que ejecutan tareas y almacenan metadatos y resultados en el Blob Storage.

HDInsight ejecuta en un conjunto de máquinas virtuales de Azure que son provisionadas cuando se crea el clúster y utiliza una base de datos Azure SQL Database para almacenar metadatos del clúster. El clúster se encuentra aislado del resto de los componentes de Azure y su acceso se da mediante una puerta de acceso (*gateway*) segura que expone un *endpoint* único y se encarga de la autenticación de los componentes para acceder al clúster.

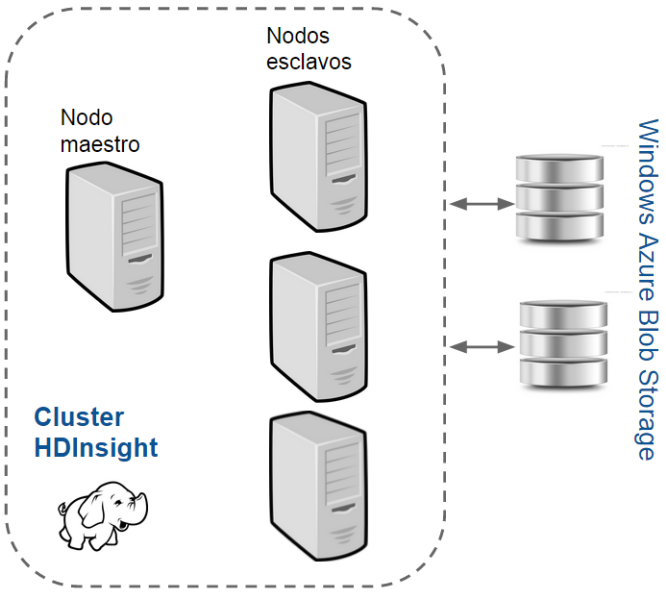

Figura 4.5: Arquitectura general de HDInsight.

4.4.1. Arquitectura general y almacenamiento

Al momento de desplegar la aplicación en el cloud, es importante tener en cuenta que el clúster se encuentre físicamente en la misma ubicación que la cuenta de almacenamiento dentro del centro de datos de Azure, para no generar un *overhead* por la transmisión de datos entre el clúster y el acceso a datos. Este punto es configurable a la hora de crear el clúster a través del portal web de Azure.

El clúster creado para este proyecto cuenta con dos nodos maestros (*Head Nodes*) encargados de la administración de los datos y ejecuciones dentro del clúster, así como ocho nodos esclavos encargados de ejecutar las tareas MapReduce. Las características de estos componentes se pueden observar en la Tabla 4.1. La cantidad de nodos esclavos (*Data Nodes*) existentes en el clúster, limita la cantidad de tareas MapReduce que se pueden ejecutar de manera paralela. Un aspecto importante de HDInsight y su integración con el resto de la plataforma Azure es el manejo del sistema de archivos de Hadoop (HDFS), el cual se implementa como un contenedor de una cuenta de almacenamiento utilizando *blobs*. Este punto facilita la cohesión entre los distintos componentes de la arquitectura ya que poseen el acceso común a la *Storage Account*.

Cuadro 4.1: Componentes del clúster HDInsight utilizado en el proyecto

Componente	**Cantidad**	**Tamaño**
Nodo maestro	2	A3 (4 cores, 7 GB RAM)
Nodo esclavo	8	A3 (4 cores, 7 GB RAM)
Contenedor (Blob Storage)	1	máx. 500 TB [41]

Nodo maestro

El nodo maestro se encarga de la administración del clúster. Decide cuándo y en qué nodo se ejecutarán las distintas tareas MapReduce, y expone servicios entre los que se encuentran:

- **HiveServer.** Es un software de almacén de datos construido sobre Hadoop que permite realizar consultas y manejar grandes conjuntos de datos en almacenamiento distribuido usando un lenguaje similar a SQL llamado HiveQL.
- **Pig.** Es una plataforma de alto nivel que permite ejecutar tareas MapReduce complejas en un conjunto grande de datos utilizando un lenguaje de *scripting* llamado Pig Latin.
- **Sqoop.** Es una herramienta que transfiere datos en masa desde Hadoop a una base de datos relacional como SQL.
- **Oozie.** Es un servicio de administración de flujos de trabajo.
- **Templeton.** Es un servicio encargado del aprovisionamiento, administración y monitoreo del clúster.
- **Ambari.** Es un servicio encargado del aprovisionamiento, administración y monitoreo del clúster.

Donde el único servicio altamente utilizado en el contexto de este proyecto es Templeton. Este servicio es utilizado por la API de HDInsight para obtener el estado de una tarea lanzada en el clúster, permitiendo saber si la misma se encuentra en ejecución, si finalizó exitosamente o si falló.

Nodo esclavo

Los nodos esclavos ejecutan servicios que dan soporte a la coordinación y ejecución de tareas así como acceso a datos:

- TaskTracker
- DataNode
- Pig
- Hive Client

Todos los nodos que conforman el clúster HDInsight, poseen un componente interfaz entre lo que el nodo interpreta como HDFS y el *Blob Storage* llamado HDSF API, como podemos observar en la Figura 4.6. Este hecho hace transparente el acceso al *Blob Storage* por parte de cada nodo del clúster. A su vez, cada nodo posee un almacenamiento local donde cada tarea *Map* o *Reduce* puede copiar archivos hacia el *Blob Storage*; es importante destacar que estos datos almacenados localmente son eliminados cuando se elimina el clúster, no siendo así con los datos almacenados en el *Blob Storage*. Es posible también aprovisionar a los nodos con archivos en caché local para que su acceso sea directo, este punto es fundamental a la hora de brindar bibliotecas para la ejecución de la rutina *Map*, por ejemplo.

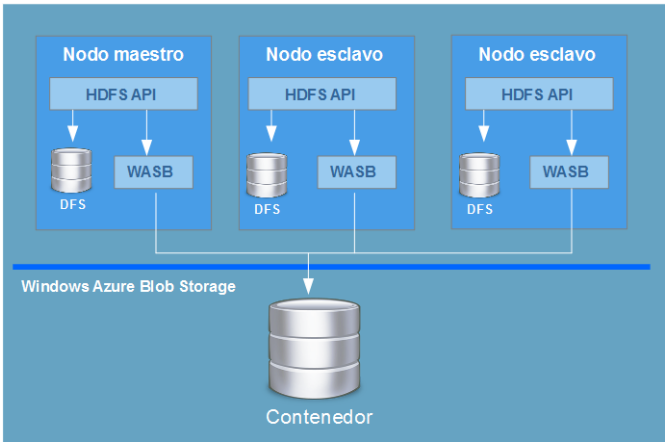

Figura 4.6: Arquitectura de almacenamiento de HDInsight.

Cabe destacar que Azure permite el acceso por escritorio remoto a cualquier de los nodos del clúster, herramienta muy útil a la hora de depurar la implementación de tareas MapReduce.

4.5. Almacenamiento

La cuenta de almacenamiento o *Storage Account* es uno de los componentes centrales de la arquitectura y juega varios roles dentro de la aplicación.

Por un lado almacena los archivos que se generan a lo largo del flujo de la aplicación, desde las distintas entradas para MCell, la salida de FERnet y el archivo comprimido con todos los resultados de la ejecución de FERnet que se entrega al usuario; a su vez, almacena metadatos que se utilizan para monitorear las distintas tareas en ejecución así como archivos de control, detallados en el capítulo 5. Todos estos datos son almacenados en forma de *blobs* dentro de la cuenta, siendo el servicio de almacenamiento brindado por Azure el servicio más utilizado en el proyecto. El único caso excepcional se da para la implementación del balanceo de carga entre los distintos roles del *Cloud Service*, donde se requieren métricas de rendimiento que son almacenadas en tablas y no en *blobs*. En el algoritmo de balanceo de carga se utiliza el servicio de *Diagnostics* de Microsoft Azure para monitorear tanto máquinas virtuales como roles web y de trabajo y obtener métricas de rendimiento, como se explicará con más detalle en el capítulo 5.

Como se mencionó en la sección 4.2.2, la comunicación con la cuenta de almacenamiento se realiza invocando servicios RESTful lo que la hace altamente disponible y escalable. En caso de fallar la cuenta de almacenamiento, se puede acceder a alguna de sus réplicas sin inconvenientes ya que para acceder se utilizan protocolos sin estado, como HTTP.

4.6. Escalabilidad

La escalabilidad afecta directamente a la disponibilidad, en el caso en que la aplicación falle debido a una sobrecarga de pedidos, ejecuciones, etc. Las aplicaciones escalables pueden adaptarse a los cambios en la demanda sin dejar a todo el sistema inutilizable. Como se explicó en el capítulo 2, existen dos tipos de escalados: horizontal y vertical. Un escalado vertical en la plataforma de Azure implica volver a desplegar los recursos, lo que generalmente no es una opción factible; es por ello que el escalado por excelencia en el cloud es el escalado horizontal y es el que ayuda a implementar sistemas flexibles.

4.6.1. Cloud Service

El escalado en los roles web y de trabajo de un *Cloud Service* se pueden configurar fácilmente a través del portal de administración de Azure, el cual presenta una interfaz amigable como se puede apreciar en la Figura 4.7. Es posible programar los escalados sacando provecho del conocimiento de la demanda del sistema en fechas u horarios especiales. Además, se puede configurar el autoescalado horizontal especificando algunos parámetros entre los que se encuentran:

- Métricas a utilizar como disparadores del escalado, en este proyecto se utilizó el porcentaje de uso de CPU.

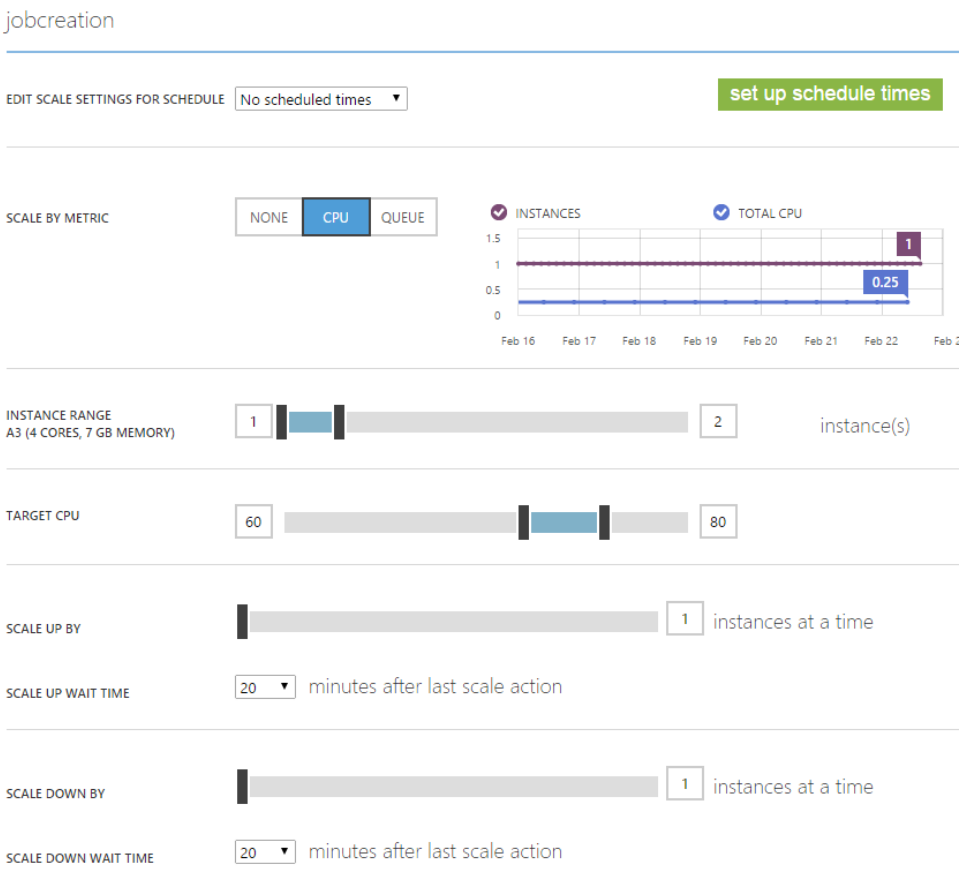

Figura 4.7: Autoescalado de rol de trabajo *Job Creation* en el portal Azure.

- Rango permitido de instancias, definiendo un mínimo y máximo. Este punto sirve para adaptarse al tipo de cuenta de Microsoft Azure que se esté utilizando y cumplir los criterios del contrato de servicio (SLA).
- Porcentaje de CPU deseado, Azure agrega o elimina instancias para mantener el porcentaje de CPU en el rango especificado.
- Número de instancias a agregar por vez al momento de escalar hacia arriba.
- Tiempo de espera desde el último escalado hasta agregar una nueva instancia.
- Número de instancias a eliminar por vez al realizar un escalado hacia abajo.
- Tiempo de espera desde el último escalado hasta eliminar una instancia.

Una vez que las métricas indican que se debe escalar y particularmente en el caso de escalar hacia arriba, existe un tiempo de espera en el cual se pone en marcha la creación de una nueva instancia para ese rol, donde la plataforma de Azure crea la máquina virtual y configura el sistema operativo, etc. Esto quiere decir que durante ese pequeño período de tiempo seguramente la utilización del sistema no sea óptimo y se noten ciertas demoras en respuesta.

4.6.2. HDInsight

El clúster HDInsight permite escalado tanto vertical como horizontal, pero en ambos casos el clúster debe ser reiniciado, o más bien recreado. Al momento de crear un clúster se especifican la cantidad de nodos maestros y esclavos, y el respectivo tamaño de los mismos. Si por alguna razón se debiera escalar, se debe dar de baja el clúster y volverlo a crear con los parámetros deseados. En principio, esto implica un tiempo en el cual el clúster, y por tanto la aplicación en general, no se encuentra disponible. El tiempo aproximado de creación del clúster se encuentra entre 20 y 30 minutos, por lo que es el tiempo aproximado de baja del sistema.

Un aspecto no menor es que al recrear el clúster, no necesariamente se pierden los datos utilizados por el mismo, ya que si se vuelve a crear en el mismo contenedor que existía previamente, puede seguir accediendo a los mismos datos. Los archivos de configuración del clúster pasan a ser nuevos y diferentes, pero todos los archivos de variables de control del sistema y datos se mantienen intactos.

4.6.3. Storage Account

El componente que más interacción posee con el *Blob Storage* y que mayor nivel de datos genera es el clúster HDInsight. Las cuentas de almacenamiento de Azure tienen límites en la velocidad de salida y entrada de datos; por ejemplo, en una cuenta de almacenamiento con georedundancia nos encontramos con los siguientes límites dentro de las regiones de EEUU [41]:

- Máxima velocidad de ingreso de datos: 10 Gbps.
- Máxima velocidad de salida de datos: 20 Gbps.

Lo que sucede cuando estos límites se sobrepasan es que Azure realiza lo que se llama ahogamiento (*throttling*) de pedidos, lo que se proyecta en el clúster como una falla de tareas donde se ven errores de entrada/salida, o errores en las conexiones al *Blob Storage* con respuesta HTTP 500 o 503, por ejemplo [48].

Para mitigar este ahogamiento y aunque no se utilizó en este proyecto, es posible escalar también el almacenamiento de la arquitectura utilizando más de una cuenta de almacenamiento. Este punto es posible de dos maneras,

una de ellas es al momento de aprovisionar el clúster HDInsight definiendo cuentas de almacenamiento adicionales, donde el clúster puede escribir y leer información. Sin embargo, si el clúster ya fue creado y se desea escalar leyendo de otras cuentas de almacenamiento, se puede realizar en la definición de las distintas tareas, pero conlleva una falla de seguridad ya que HDInsight almacena las claves de acceso a las cuentas de almacenamiento encriptadas en archivos de configuración, aspecto que no se realiza si estas claves se asignan por código.

4.7. Tolerancia a fallos

En todo sistema distribuido debe tenerse diversas consideraciones para poder brindarle al usuario la mayor disponibilidad y confiabilidad posible. Uno de los grandes aspectos es la manipulación de errores para lograr una pronta recuperación. A continuación se analizan los puntos de la aplicación donde pueden surgir fallos y las respectivas acciones llevadas a cabo.

4.7.1. Almacenamiento

En la arquitectura diseñada para el proyecto actual se cuenta con un espacio de almacenamiento central utilizado por todos los nodos de la aplicación para obtención, generación y mantenimiento de datos y metadatos, siendo incluso utilizado por el clúster HDInsight para almacenar información relativa a las tareas que ejecuta. Sin embargo, esto no representa un problema ya que la cuenta de almacenamiento utilizada es de tipo geo-redundante, lo que significa que dentro de la región de almacenamiento que está siendo utilizada en el momento, la información se encuentra replicada seis veces. Tres réplicas se encuentran dentro de la región primaria y otras tres en la región secundaria a cientos de kilómetros de distancia de la primera, lo que asegura un alto nivel de accesibilidad. En caso de un fallo en la región primara, el sistema de recuperación de fallos procede a acceder a la región secundaria. Gracias a esta georedundancia es prácticamente imposible que se genere un cuello de botella o dificultad a la hora de consultar los datos, por lo tanto no representa un problema que todos los nodos de la aplicación hagan uso intensivo de la información guardada en dicha cuenta de almacenamiento. Por otro lado, si ocurriera un fallo a nivel de *hardware* dentro de la región geográfica primaria, el mismo sistema de recuperación de fallos de Azure se encarga de que la información continúe siendo accesible en todo momento; para esto es clave que el tipo de cuenta sea georedundante.

4.7.2. Comunicación

Otro punto propenso a fallos es el sistema de comunicación WCF. El mismo se basa en el envío de pedidos asíncronos desde un punto de acceso

a otro, ubicados en diferentes roles del *Cloud Service*. Los fallos pueden ocurrir al no poder enviar el pedido de forma exitosa de un extremo a otro, ya sea por falta de respuesta o por problemas en la red. Para solucionar este inconveniente el marco de trabajo WCF provee un servicio de mensajería confiable como se explicó en la sección 4.2.2, donde se utilizan colas de mensajes y se provee un mecanismo de reintentos. Si luego de un periodo de tiempo no se logra establecer la conexión, significa que hay un problema mayor de infraestructura donde alguno de los nodos del *Cloud Service* no se encuentra disponible, lo cual es muy poco probable como se explicará a continuación. Adicionalmente en la comunicación entre roles, no se tiene en cuenta para el balanceo de carga las instancias de roles de trabajo que no se encuentren *saludables*, este aspecto es manejado por el marco de trabajo del *Cloud Service* de Azure.

4.7.3. Cloud Service

Ya que el *Cloud Service* es un servicio del tipo PaaS, la infraestructura que se encuentra por debajo de la implementación de cada rol, es completamente manejada por el proveedor de servicio, en este caso Microsoft Azure. En el contrato de servicio (SLA) se establece que para cada rol que cuente con un mínimo de dos instancias la disponibilidad del mismo será mayor al 99 %. Por ello, cada rol del *Cloud Service* fue configurado para contar inicialmente con dos instancias (eventualmente podrían ser más gracias a la herramienta de escalado automático) lo que asegura una tasa de disponibilidad muy alta. Azure también se encarga de monitorear y gestionar el estado de las instancias de cada rol, si alguna de ellas no se encuentra *saludable*, el administrador de recursos se encargará de reiniciarla.

4.7.4. Clúster HDInsight

Otro elemento esencial de la aplicación es el clúster HDInsight, donde se realiza el principal cómputo del proyecto: las simulaciones. Existen varios puntos donde se pueden producir fallos dentro del clúster. Como un servicio propio de HDInsight el clúster cuenta con dos *Head-Nodes* (nodos maestros) lo que asegura que no existe un único punto de fallo. Otro punto propenso a fallos del clúster son las subtareas del algoritmo MapReduce, es decir los *mappers* y *reducers*. Para prevenir la posible falla y también como parte del servicio HDInsight, las tareas *Map* y *Reduce* que fallan se re-ejecutan hasta una determinada cantidad de veces configurable. De esta manera se logran mitigar los fallos esporádicos en la ejecución de la aplicación.

4.8. Seguridad

La plataforma de Windows Azure cuenta con diversos servicios que son utilizados para crear arquitecturas distribuidas en el cloud. Para poder utilizar estos servicios se aplican distintos métodos de autenticación, como una cuenta de Microsoft si se accede mediante el portal web o la consola PowerShell y certificados en caso de acceder a los servicios mediante código. Azure provee algunos puntos con respecto a la seguridad entre los que se encuentran:

- **SSL.** Se utiliza *Secure Sockets Layer* entre todas las comunicaciones internas entre componentes de Azure.
- **Manejo de certificados y clave privada.** Azure provee una arquitectura donde los certificados no se encuentran almacenados en el mismo lugar que los componentes que los utilizan. Los certificados y las claves privadas se agregan a la plataforma Azure utilizando el portal web y las claves se almacenan de manera encriptada.
- **Cuentas de almacenamiento.** Cada cuenta de almacenamiento de Azure posee su propia clave de acceso secreta donde es posible asignar una cuenta de almacenamiento a cada aplicación y así mantener los datos protegidos. En el caso de los *Cloud Services*, la clave de acceso se almacena encriptada en el archivo de configuración.

4.8.1. HDInsight

El primer punto sobre seguridad en HDInsight a tener en cuenta es que el sistema de archivos HDFS se implementa sobre un contenedor en el *Blob Storage* y por tanto es crítico generar un acceso seguro al contenedor para que los datos generados por el clúster no sean accedidos de manera maliciosa. Para ello es necesario guardar las claves de acceso a la cuenta de almacenamiento de manera encriptada como se mencionó en el punto anterior.

Los contenedores dentro de una cuenta de almacenamiento poseen distintos niveles de acceso. Es importante que toda la información que se quiera mantener confidencial se encuentre dentro de la categoría *apagado* (*off*), que implica que sólo el administrador puede acceder a la información del contenedor utilizando las credenciales correspondientes.

Por otro lado, el clúster permite acceder a las máquinas virtuales utilizando escritorio remoto. Por lo que es importante disponer de una contraseña fuerte y segura que no sea fácilmente descifrable [6].

Capítulo 5

Implementación

En este capítulo se presentan aspectos generales de la implementación de la aplicación, realizando especial énfasis en componentes esenciales y describiendo los obstáculos que debieron superarse durante el desarrollo. Asimismo, se detallan las tecnologías empleadas, las decisiones tomadas y el proceso de desarrollo del sistema.

5.1. Instalación de entorno y tecnologías

Durante el desarrollo del proyecto se utilizaron diversas herramientas de software con diferentes propósitos. Desde el comienzo, y debido a que la plataforma de cloud seleccionada es Microsoft Azure, se trabajó sobre el sistema operativo Windows (tanto Windows 7 como Windows 8).

Para interactuar con los servicios brindados por Azure es un requisito contar con una cuenta de Microsoft, con la cual se ingresa al portal web de manejo y configuración [12]. Dependiendo del nivel de privilegio, costo y tipo de cuenta, es posible acceder a diferentes características y servicios. A lo largo del proyecto se utilizaron tres tipos de cuentas. Inicialmente, se optó por una cuenta de prueba gratuita durante un período de sesenta días. La cuenta gratuita permite un nivel de servicios muy limitado, especialmente en el área de clúster y computación de alto desempeño. De todas maneras, permitió obtener un entendimiento general del portal y un acercamiento primario a los servicios brindados por la plataforma Azure. Luego, como se mencionó en el capítulo 4, a través de los cursos de capacitación realizados fue posible acceder a una cuenta de capacitación con mayores niveles de privilegio. La cuenta de capacitación fue utilizada durante aproximadamente la mitad del proyecto y permitió un mayor grado de acción y desarrollo en comparación con la cuenta gratuita.

Durante el transcurso del proyecto se aplicó para una cuenta por concepto de investigación que brinda mayores capacidades que la cuenta de capacitación. Finalmente se logró acceder a la cuenta de investigación que

se continuó utilizando hasta la finalización del proyecto. Para comenzar a utilizar la cuenta de investigación fue necesario realizar un proceso de migración que involucró al portal web así como al entorno de desarrollo que se describe más adelante.

A través del portal web de Azure es posible crear y gestionar cuentas de almacenamiento, *Cloud Services*, clústers HDInsight, etc. Una vez configurados estos servicios es posible desarrollar funcionalidades o ejecutar tareas a través de otras herramientas. Se procedió a la instalación de *Microsoft Web Platform Installer*, una herramienta que permite descargar e instalar programas vinculados al desarrollo y utilización de Windows Azure. A continuación se describen los principales componentes de software utilizados.

Azure Powershell. Esta herramienta consiste de una interfaz de consola (CLI) con posibilidad de escritura y unión de comandos por medio de instrucciones. Azure Powershell permite establecer una conexión con la cuenta de Azure, así como consultar y hacer uso de los servicios en general. En el proyecto, la herramienta fue el mecanismo primario para configurar y lanzar tareas en el clúster HDInsight mientras aún no se había desarrollado el portal web. Posteriormente, Azure Powershell se utilizó para obtener el identificador de suscripción que luego fue almacenado en un archivo de configuración del proyecto web y para el manejo y descarga de certificados de seguridad necesarios para desplegar el portal web en el cloud. Sus ventajas incluyen la facilidad de instalación y de uso. Su desventaja, es que Microsoft realiza cambios en los comandos de Azure Powershell los cuales de forma frecuente dejan de ser soportados en la versión inmediatamente anterior de la herramienta.

NetBeans. Se instaló y utilizó el entorno integrado de desarrollo NetBeans junto con la edición estándar de Java 7 (SE JDK). Ya que Hadoop está desarrollado originalmente en Java, se optó por desarrollar en este mismo lenguaje el algoritmo MapReduce para la aplicación. Utilizando NetBeans se generó un proyecto conteniendo todas las clases y bibliotecas necesarias para la ejecución de una tarea MapReduce. La implementación de dicho proyecto se detalla más adelante en este mismo capítulo.

Visual Studio. Como entorno principal de desarrollo se utilizó Microsoft Visual Studio versión 2012. A través de esta plataforma fue posible el desarrollo completo del portal web principal del proyecto. Se utilizó el lenguaje C# nativo de .Net, con páginas web desarrolladas en asp.net y HTML embebido. Fue necesario instalar el SDK para Visual Studio provisto por Microsoft Azure, esto significa que se agregaron herramientas al entorno de desarrollo que permitieron una integración completa con los servicios en el cloud brindados por Azure. Esto resultó en el desarrollo de un sitio web con una completa conectividad e interacción con las cuentas de almacenamiento, servicios de recolección de métricas, clúster HDInsight y otras funcionalidades del cloud.

SVN. Tanto para la implementación del algoritmo MapReduce como para la implementación del portal web se utilizó un sistema de versionado. Para mantener las versiones de los proyectos utilizados se hizo uso de la herramienta de Apache Tortoise SVN. Dicha herramienta requiere de la configuración de un servidor para luego acceder a los datos del mismo a través de un cliente instalado en el ambiente local al usuario. Dentro de la suscripción de Azure se creó un servidor que utiliza Windows Server como sistema operativo. En el mismo, se aloja el repositorio de datos de Tortoise SVN que contiene tanto el código del proyecto del portal web y como del proyecto que contiene la implementación del algoritmo MapReduce. Para la recuperación de dichos proyectos se instaló el cliente Tortoise. Tortoise SVN permite operaciones concurrentes sobre el mismo archivo en diferentes clientes, control de cambios e historial de versiones.

Azure Storage Explorer. Este componente de software permite la conectividad y acceso a todos los *blobs*, tablas y colas pertenecientes a una cuenta de almacenamiento Azure. La conexión a esta cuenta se realiza a través de una clave de acceso que se puede obtener desde el portal web de Azure. *Azure Storage Explorer* se utilizó durante el proyecto principalmente para monitorear y acceder a los resultados de las tareas que ejecutan en el clúster HDInsight.

Versiones y Miscelánea. En la tabla 5.1 se pueden observar las versiones de cada componente de software utilizadas en la aplicación. Adicionalmente, se utilizaron otras herramientas para el mantenimiento de archivos versionados, en este caso Dropbox para compartir archivos y proyectos y Google Drive para recolección de información, intercambio de dudas y archivos en general. Se utilizó Latex para la elaboración del informe.

5.2. MCell y FERnet

Para la implementación de la aplicación distribuida en el cloud, se utilizaron dos componentes de software, MCell y FERnet, cuyo propósito y utilidad general fueron detallados en el capítulo 3. Por un lado, ambos programas fueron desarrollados originalmente en el lenguaje C y diseñados para ejecutar sobre el sistema operativo Linux. Por otro lado, la aplicación que se desarrolló durante el proyecto de grado debe ejecutar en su totalidad sobre la plataforma Windows. Esto se debe a que la tecnología empleada es Microsoft Azure y tanto las máquinas virtuales de los *Cloud Services* como los servidores maestro y esclavo del clúster HDInsight utilizan Windows Server. Debido a estos dos factores fue necesario realizar cambios en la implementación en varios componentes de software para poder compilarlos y ejecutarlos sobre el sistema operativo Windows. Se empleó el ambiente integrado de desarrollo CodeBlocks, en el que se creó un proyecto por programa y se realizó la adaptación a Windows. En la sección de Anexos A se encuentra

Cuadro 5.1: Versiones de Software

Componente	**Versión**
Microsoft Visual Studio	Ultimate 2012
Microsoft .Net Framework	4.5
Microsoft .Net Framework SDK	4.5
NetBeans IDE	7.4
Java SE Development kit 7	1.7.0 (v51)
Microsoft Web Platform Installer	5.0
Microsoft Azure Tools for Visual Studio	2.4
Microsoft HDInsight Emulator for Windows Azure	1.0.0
Microsoft Azure Storage Tools	2.5.1
Windows Azure Storage Emulator	3.3
Azure Storage Explorer	6.0
Windows Azure Libraries for .Net	2.3
Microsoft Azure Powershell	0.8.7
Microsoft ASP.Net Web Pages	1.0.2
Service Bus (Solo en pruebas iniciales)	1.0
TortoiseSVN	1.8
IIS Express	8.0
CodeBlocks	13.12

un tutorial detallado que contiene las acciones necesarias para llevar a cabo la adaptación de MCell y FERnet de Linux a Windows.

Luego de obtener ambas aplicaciones compiladas y en funcionamiento sobre Windows, se generaron los archivos necesarios para poder ejecutarlas de forma independiente al entorno de desarrollo. En este caso el funcionamiento de ambas aplicaciones se fusiona y se utiliza en conjunto de forma serial, donde la salida de una ejecución de MCell se convierte en la entrada de FERnet en el clúster HDInsight.

5.2.1. MCell

Para el proyecto se utiliza la versión MCell3 del software, con ciertas modificaciones realizadas por el equipo de trabajo de la UBA para que sea posible comunicar la salida de MCell con FERnet utilizando un *pipeline*. Esta funcionalidad no es utilizada en este proyecto al ser una arquitectura distribuida. MCell realiza simulaciones del movimiento de ciertas moléculas dentro de un escenario definido y teniendo en cuenta las reacciones que ocurrirán entre dichas moléculas en caso de colisionar. Para cada paso de tiempo, se genera una lista que contiene las posiciones de las moléculas. Dichas posiciones serán luego analizadas por FERnet para determinar cuántos fotones se emitieron. Cada ejecución de MCell tarda un tiempo considerable

en comparación al proceso en general, aproximadamente un 60 % del tiempo.

Para la ejecución de simulaciones, MCell utiliza una especificación definida en formato MDL (*model description language*). Este tipo de archivos en general tienen extensión .mdl, pero esto no es un requisito obligatorio. Un archivo MDL es un archivo de texto que contiene comandos separados por espacios en blanco. Para la implementación de MCell, los comandos del archivo MDL de entrada se agrupan como se explica a continuación.

- **Inicialización.** Estos comandos permiten la configuración de parámetros globales como el paso de tiempo, el particionamiento espacial y la duración de la simulación.
- **Definición de moléculas.** Especifican los nombres y constantes de difusión de las moléculas para la simulación.
- **Definición de reacciones.** Permiten especificar las reacciones que ocurren entre las diferentes moléculas y la tasa a la que ocurren dichas reacciones.
- **Geometría.** Estos comandos describen las membranas y fronteras dentro de las cuales se llevará a cabo la simulación, además de la configuración inicial de las moléculas en dicho espacio.
- **Salida.** Se especifica que tipo de información se mostrará como salida de la ejecución. Es posible configurar la salida para que incluya el progreso de la simulación en consola y las listas de reacciones entre moléculas para cada paso de tiempo.

5.2.2. FERnet

FERnet se encarga de analizar la salida generada por MCell. Cuenta con varias modalidades de análisis simulando el comportamiento real de un microscopio. En el proyecto actual se trabajan con las siguientes dos configuraciones posibles:

- **Punto (*Point*)** Se simula la posición del microscopio en un punto dado de la muestra y se toman los valores de fotones desde ese punto fijo. Este modo genera un sólo archivo de salida.
- **Multipunto (*Multi*)** Con este modo se genera una grilla donde se definen centros que se separan una distancia dx y dy definidas en el archivo de configuración de FERnet, y donde cada uno de esos puntos simula un microscopio. Se genera un archivo de salida por cada punto donde se simula la existencia de un microscopio que capta los fotones emitidos por las especies.

En todos los casos, FERnet se encarga de generar un conteo de fotones por cada paso de tiempo, que luego es utilizado para realizar los mismos

análisis que se realizarían utilizando datos experimentales (por ejemplo, correlación). Actualmente, este análisis posterior debe ser llevado a cabo por el propio usuario.

5.3. Clúster HDInsight

El clúster HDInsight representa el componente fundamental del proyecto. Se creó y utilizó para albergar un software que implementa el modelo MapReduce, permitiendo realizar ejecuciones de simulaciones en forma paralela buscando optimizar los tiempos de cómputo. Estas simulaciones en paralelo se aplican tanto para un usuario que ejecuta un número determinado de tareas, como para numerosos usuarios que se conectan al sistema concurrentemente.

La configuración y creación del clúster se realiza a través del portal web de configuración de Azure. En el portal se seleccionan el tipo de clúster, la ubicación geográfica, la cantidad de nodos, la cuenta de almacenamiento asociada, los nombres de usuario y las contraseñas correspondientes. El clúster puede ser de varios tipos: *Hadoop* para cargas de trabajo de consulta y análisis, *HBase* para cargas de trabajo no SQL y *Storm* para cargas de trabajo de procesamiento en tiempo real. Para la implementación del proyecto se utilizó un clúster HDInsight de tipo Hadoop. Una vez configurado el clúster se puede monitorear el estado y métricas de utilización generales a través del portal web de Azure. Además, es posible acceder de manera remota a cualquiera de los servidores que conforman el clúster, incluyendo al nodo maestro.

El sistema de almacenamiento del clúster utilizado en el proyecto es una cuenta de Azure granulada en *blobs*. Es por ello que para proporcionar archivos de entrada y obtener resultados intermedios y salidas fue de vital importancia la utilización del software *Azure Storage Explorer*. Sin embargo, también se realizaron accesos a través del portal web y a nivel del código en la solución web desarrollada.

5.3.1. MapReduce

Para la implementación del algoritmo MapReduce se generó un proyecto Java. Este proyecto se denomina *CienciaCelularMR*. La figura 5.1 muestra un diagrama de la estructura del algoritmo.

Main Es la clase principal del proyecto, donde se generan y lanzan las tareas *mapper* y *reducer* requeridas. Como se puede observar en la Figura 5.2 se utilizó una herramienta para la composición de *mappers* provista por Apache Hadoop denominada ChainMapper. ChainMapper permite la ejecución de varias tareas de tipo *Map* en cadena, es decir al finalizar la primer tarea *mapper* se lanza la ejecución de la segunda, utilizando como entrada para

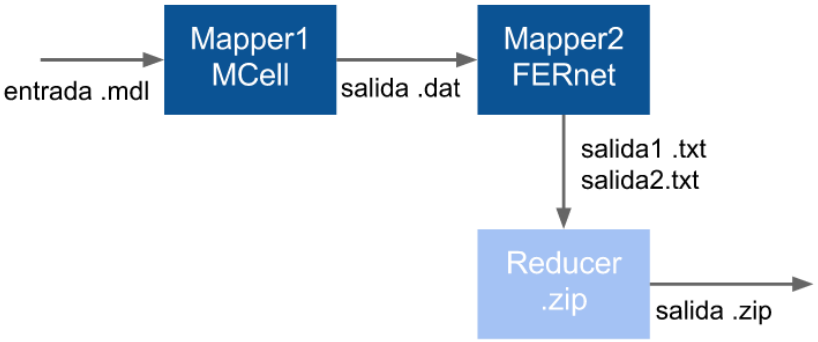

Figura 5.1: Estructura algoritmo utilizando MapReduce

ésta la salida de la anterior, y así sucesivamente. Luego de que todas las tareas *mapper* finalizan se procede a la ejecución de la tarea *reducer*. En este caso se utilizaron dos *mappers* concatenados correspondientes a la ejecución de MCell y FERnet respectivamente, y además se implementó una tarea *reducer*.

En la clase Main también se realizan configuraciones del ambiente YARN como se explica en la sección 5.3.2 y se añaden archivos requeridos para la ejecución de las tareas *mapper* y *reducer* al *cache* distribuido de Hadoop. Estos archivos necesariamente deben encontrarse almacenados en la *Storage Account* y deben ser accesibles para cualquier nodo del clúster a través de su URL. Los archivos se agregan al *cache* especificando su URL a través de la clase para configuración de tareas (*JobConf*). Una vez realizada esta acción, los archivos pueden ser accedidos desde el resto de las clases del proyecto, ya que la plataforma se encarga de copiar aquellos que sean necesarios a cada nodo esclavo antes de ejecutar la tarea. La eficiencia de este mecanismo reside en el hecho de que cada archivo es copiado una única vez por tarea, y que además son agregados al *cache* de cada nodo sin la necesidad de copiarlos al disco duro. En este caso los archivos agregados a *cache* comprenden los ejecutables de MCell y FERnet, sus respectivas bibliotecas utilizadas, y el archivo de configuración de FERnet [55].

File Input Format y Record Reader Como se explicó en el capítulo 2, una de las primeras tareas que se realiza al ejecutar una tarea MapReduce es tomar los datos de entrada y particionarlos en *splits*, donde cada *split* será entregado a una tarea *map*, la clase encargada de realizar este trabajo es InputFormat dentro del *framework* Hadoop. Por defecto se realiza una partición en splits según la configuración de la tarea (como tamaño mínimo y máximo de split, o cantidad de nodos disponibles), pero también es posible realizar una implementación propia de la interfaz InputFormat para

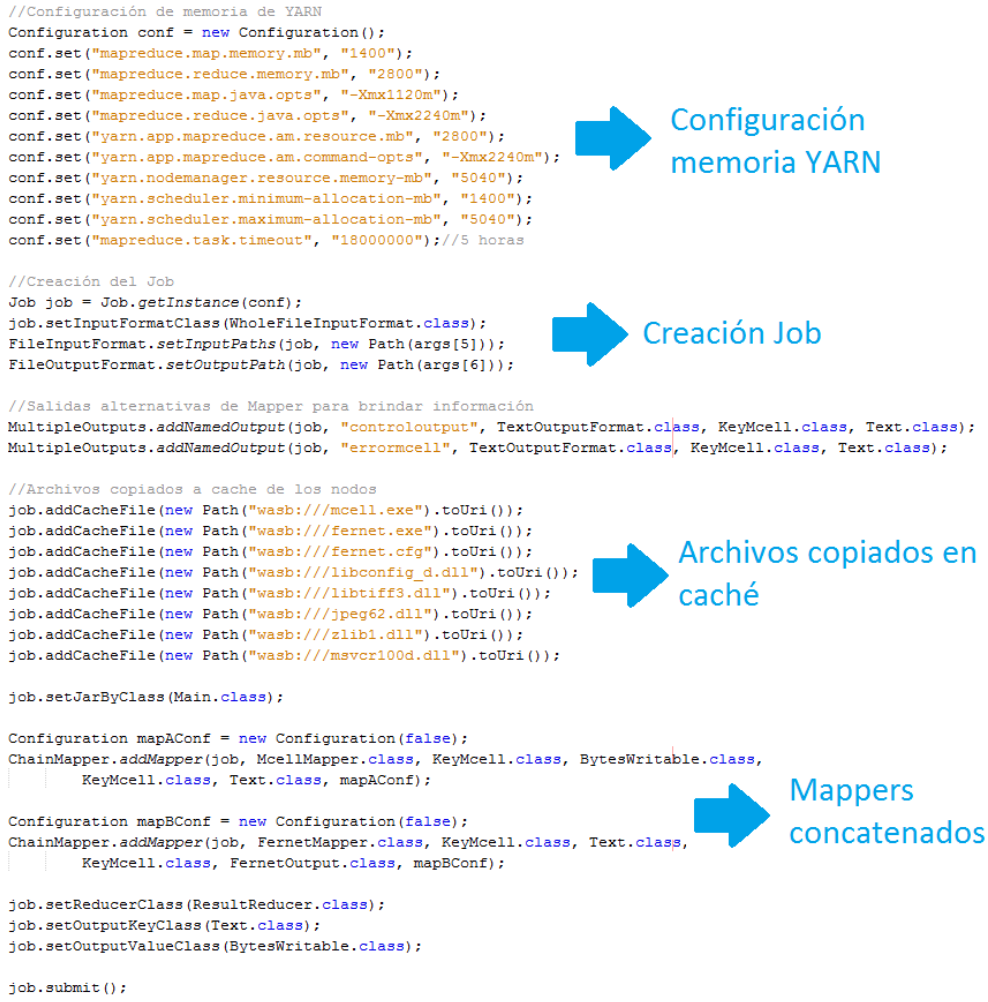

Figura 5.2: Implementación clase *Main* tarea MapReduce.

personalizar la división en splits. En el contexto de este proyecto no se desea realizar una partición de los datos de entrada como se suele realizar en Big Data, sino que se busca utilizar el clúster HDInsight (y por tanto Hadoop) como herramienta de paralelización de tareas. Por este motivo existe un único archivo de entrada, y no se permite su partición. Para hacer uso del archivo de entrada sin particionar, se implementa la interfaz InputFormat de forma que tome todo el archivo y genere un solo *split* con él. Para ello se implementa una clase llamada WholeFileInputFormat.java extendiendo la ya existente clase de Hadoop FileInputFormat (que gestiona los *splits* como

archivos) y donde se sobreescriben dos métodos de suma importancia para lograr el cometido:

- **isSplitable.** Especifica si el archivo se puede separar en *splits*. En la implementación realizada se retorna que no es posible separar en *splits*.
- **createRecordReader.** Devuelve una instancia de un RecordReader, encargado de armar los pares clave-valor a partir de un *split* que se enviarán al *mapper* para su procesamiento.

Es por tanto necesario definir una implementación propia de la interfaz RecordReader donde además de tomar todo el archivo como único valor a enviar al *mapper*, se define la clave que se utilizará a lo largo de todo el flujo MapReduce. Se crea la clase *WholeFileRecordReader.java* que procesa el *split* generando un par clave-valor sobreescribiendo el método *nextKeyValue()*, donde se asigna como clave una estructura (definida en la clase *KeyMcell.java*) donde se almacena: el identificador del usuario, el identificador de la tarea y el modo de ejecución de FERnet, y como valor los bytes del archivo MDL. Los atributos de la clave se obtienen parseando el nombre del archivo MDL que conforma el *split* y es importante por tanto no cambiar la nomenclatura de este archivo para el correcto funcionamiento de la aplicación.

Mappers. En la implementación actual de la tarea MapReduce se incluyeron dos *mappers*.

- **MCell *mapper*** Esta tarea se encarga de la ejecución de MCell para cada una de las simulaciones de cada usuario. Se implementa una clase *McellMapper.java* que extiende la clase *mapper* de Hadoop sobreescribiendo el método *map*. Recibe como entrada una clave que contiene el identificador de simulación, el identificador de tarea y además el modo de ejecución de FERnet, así como los bytes del archivo de extensión MDL que requiere MCell para ser ejecutado. Los bytes que componen el archivo MDL son almacenados en un archivo con el nombre *entradaMCell.mdl*, de forma local al nodo esclavo que se encuentra ejecutando la tarea *map*. Con el archivo MDL (*entradaMCell.mdl*) y el archivo ejecutable de MCell que se encuentra cargado en memoria es posible lanzar la ejecución de la simulación.

  Mientras se realiza la ejecución se almacena la salida que se produce en consola, ya que es de interés para el usuario y luego estará disponible para ser descargada desde la página de obtención de resultados del portal web. Esta salida se almacena en un archivo de tipo *out* en un *blob* de la cuenta de almacenamiento cuyo nombre sigue el formato «*scicluster/test/output-Idsimulación.IdJob/controloutput-m-00000*», y contiene identificadores tanto del usuario como de la tarea. Esta porción de código retorna la clave y el nombre del archivo resultado

de la ejecución de MCell con el formato «*user/admin/salidaMCell-Idsimulación.IdJob.dat*» para ser utilizados por el siguiente *mapper*.

- **FERnet *mapper*** En este caso, la función del *mapper* es ejecutar el programa FERnet. Se implementa una clase *FernetMapper.java* que extiende la clase *mapper* de Hadoop sobreescribiendo el método *map*. Como parámetros de entrada se reciben una clave con el identificador de la simulación, el identificador de tarea y el modo de ejecución de FERnet, así como una entrada con el nombre del archivo que contiene la salida de la ejecución de MCell (con formato «*user/admin/salidaMCell-Idsimulación.IdJob.dat*»). Este archivo se encuentra almacenado como *blob* en la cuenta principal de almacenamiento y se accede al mismo de forma directa. Luego se procede a la ejecución de FERnet. Además del archivo ejecutable y la salida de MCell, se utiliza el archivo de configuración de FERnet que se encuentra almacenado en el *cache* distribuido de Hadoop. La salida en este caso es una colección de pares con la misma clave y como valor una estructura definida en la clase *FernetOutput.java* donde se especifican: el nombre del archivo generado por FERnet (por ejemplo «*out_point.txt*»), el identificador de la tarea para el usuario dado y los bytes del archivo de salida de FERnet. Si el modo de ejecución de FERnet es *point*, la salida se compone de un único archivo. Sin embargo, si se emplea el modo de ejecución *multi* la salida de FERnet se compone de dos o más archivos. Por este motivo, se puede obtener como salida una colección de pares clave-valor y no sólo uno.

Adicionalmente, para ambos *mappers* se realiza una actualización de progreso constante durante la ejecución. Esto se debe a que el marco de trabajo cuenta con un mecanismo de *timeout*, es decir que si durante un determinado período de tiempo la tarea en cuestión no reporta progreso, el rastreador de tareas (*JobTracker*) matará dicha tarea asumiendo que no se puede completar su ejecución como se explicó en el capítulo de cloud. De esta forma, reportando el progreso se asegura que la tarea continúe ejecutando aún cuando su tiempo de cómputo sea elevado.

Reducer. La función *Reduce* se encarga de agrupar todos los valores con la misma clave para luego realizar algún tipo de acción sobre ellos. Se implementa una clase *ResultReducer.java* que extiende la clase *reducer* de Hadoop sobreescribiendo el método *reduce*. *ResultReducer* recibe el identificador de la simulación y todos los pares (clave, valor) asociados a ese identificador. En la implementación de la función *reduce* del proyecto, la clave se compone del identificador de simulación junto con el identificador de tarea y el modo de ejecución de FERnet. Es decir, que para un usuario, se ejecutarán la misma cantidad de *reducers* que de simulaciones. Se utiliza el mismo formato de clave que reciben las funciones MCell *mapper* y FERnet *mapper*. Se recibe como valor una colección de estructuras *FernetOutput.java* donde se especi-

fica la información de cada archivo de salida del FERnet para una simulación dada. Con la ayuda de la clase auxiliar ZipFileWriter.java, se crea un ZIP con todos los archivos correspondientes y se almacena el archivo resultante en un *blob* con nombre de la forma «*data/Idsimulación.IdJob-resultados.zip*», dentro del contenedor de la cuenta de almacenamiento asociada al clúster.

Flujo En la Figura 5.3 se puede observar un ejemplo de flujo de ejecución de la tarea MapReduce donde participan todas las clases descritas, visualizando las claves y valores utilizados para cada paso de la tarea así como los momentos donde se guardan archivos en el *Blob Storage* (sin tener en cuenta la escritura y lectura de archivos que realiza Hadoop internamente para ejecutar la tarea).

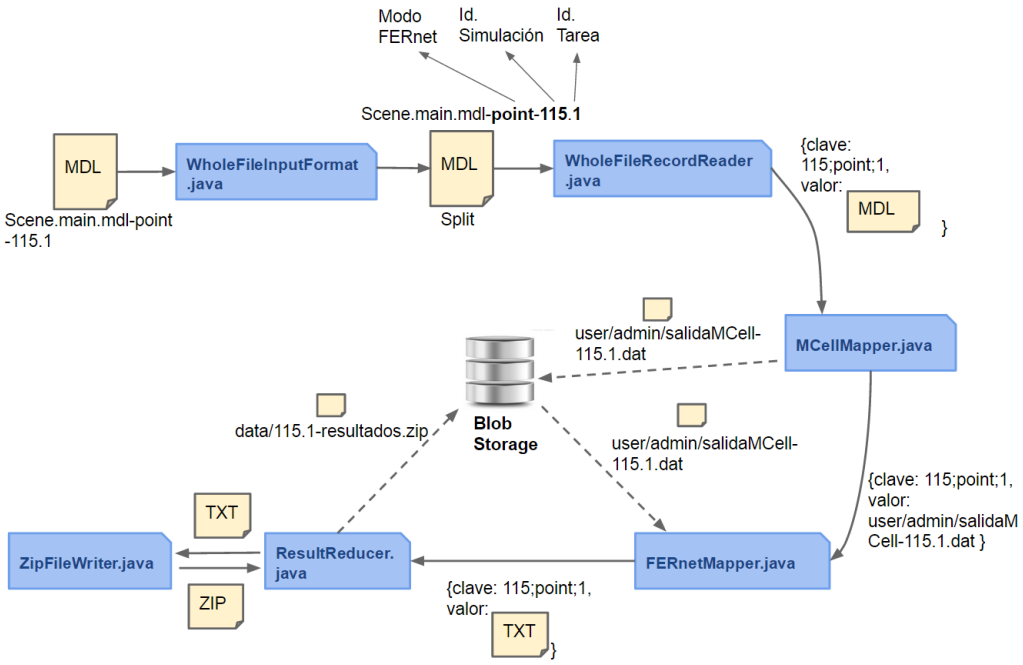

Figura 5.3: Ejemplo de flujo de implementación MapReduce realizada.

5.3.2. Configuración del clúster

Apache Hadoop YARN (*Yet Another Resource Negotiator*) es una tecnología utilizada para administrar clústers. YARN es una de las características clave a partir de la versión 2 de Hadoop. Se creó para dividir las dos funciones principales del JobTracker (NameNode), separando e independizando la gestión de recursos por un lado y la planificación y el monitoreo de las tareas por el otro. De esta separación surgen dos elementos:

- ResourceManager (RM): Este elemento es global y se encarga de toda la gestión de los recursos.
- ApplicationMaster (AM): Este elemento existe por aplicación y se encarga de la planificación y el monitoreo de las tareas.

YARN le da un contexto a las tareas MapReduce indicando qué recursos pueden utilizar. Cada tarea *map* y *reduce* se ejecuta en lo que se llama contenedor (*container*), que básicamente es un entorno con recursos asignados. El *ResourceManager* administra estos contenedores y el contenedor brinda permisos a las tareas sobre qué cantidad específica de memoria y procesador pueden utilizar. El *ApplicationMaster* se encarga de tomar ese contenedor y presentarlo al *NodeManager* que administra un nodo, para que use el contenedor como contexto de sus ejecuciones.

Debido a que las simulaciones pueden consumir muchos recursos dependiendo de las características de sus parámetros de entrada, se buscó una configuración de memoria para los contenedores que permita el uso eficiente del clúster. Para ello se siguió un tutorial brindado por Hortonworks (quien brinda la implementación de Hadoop y YARN para Azure) de cómo configurar YARN para sacar mayor provecho de los recursos disponibles al ejecutar tareas MapReduce [38].

Inicialmente, se deben conocer los recursos disponibles en cada nodo del clúster. En el caso del clúster creado para el proyecto, los detalles se pueden observar en el Cuadro 5.2.

Cuadro 5.2: Recursos disponibles en clúster HDInsight

Nodo	**RAM**	**Cores**	**Discos**
Nodo maestro	7 GB	4	2
Nodo esclavo	7 GB	4	2

Luego, se debe definir la memoria disponible, para la cual se debe tener en cuenta la memoria reservada que necesita el sistema para ejecutar procesos del sistema y los procesos Hadoop. Utilizando una tabla presente en el tutorial, se definió que la memoria disponible es de 5GB. Posteriormente, se pasa a calcular la máxima cantidad de contenedores permitidos en cada nodo utilizando la siguiente fórmula:

$$numCont = min(2 \times numCores, 1.8 \times numDiscos, \frac{RAMdisponible}{minTamanioCont})$$

Donde,

- **numCont** es la cantidad de contenedores a calcular.
- **numCores** es la cantidad de núcleos en cada nodo.
- **numDiscos** es la cantidad de discos duros en cada nodo.

- **RAM disponible** es la memoria disponible en el nodo, dejando por fuera la memoria reservada del sistema.
- **minTamanioCont** es el tamaño mínimo de contenedor que viene dado por una tabla del tutorial. Se calcula según la cantidad de memoria RAM disponible, siendo el caso del clúster con el cual se trabaja de 512MB.

El cálculo final para determinar la memoria a asignar a cada contenedor (**RAMCont**) surge de aplicar la siguiente fórmula:

$$RAMCont = max(minTamanioCont, \frac{RAMdisponible}{numCont})$$

En el Cuadro 5.3 se presentan los resultados para el caso específico del proyecto actual.

Cuadro 5.3: Parámetros asignación memoria contenedor YARN

Parámetro	**Valor**
numCont	3.6
minTamanioCont	512MB
RAMCont	1400MB

Cuadro 5.4: Configuración de memoria para MapReduce

Archivo configuración	**Parámetro configuración**	**Valor**
yarn-site.xml	yarn.nodemanager.resource.memory-mb	$numCont \times RAMCont$
yarn-site.xml	yarn.scheduler.minimum-allocation-mb	RAMCont
yarn-site.xml	yarn.scheduler.maximum-allocation-mb	$numCont \times RAMCont$
mapred-site.xml	mapreduce.map.memory.mb	RAMCont
mapred-site.xml	mapreduce.reduce.memory.mb	$2 \times RAMCont$
mapred-site.xml	mapreduce.map.java.opts	$0.8 \times RAMCont$
mapred-site.xml	mapreduce.reduce.java.opts	$0.8 \times 2 \times RAMCont$
yarn-site.xml	yarn.app.mapreduce.am.resource.mb	$2 \times RAMCont$
yarn-site.xml	yarn.app.mapreduce.am.command-opts	$0.8 \times 2 \times RAMCont$

Luego, con los valores definidos en el cuadro 5.3 y la información del Cuadro 5.4 se asigna la configuración de memoria en el método *Main* de cada tarea MapReduce, como se puede observar en la Figura 5.2. Estos valores se pueden asignar al momento de crear el clúster y se aplican globalmente a todas las tareas, pero es necesario reiniciar el clúster cada vez que se quieran modificar, dejándolo inutilizable por ese periodo. Asignando los valores a las variables definidas en el cuadro 5.4 mediante código, cada tarea utiliza sus parámetros de memoria y no se deja inutilizado el clúster.

5.4. Almacenamiento

Uno de los componentes principales del proyecto es el espacio de almacenamiento. Como ya se explicó en el capítulo 4, se utiliza una cuenta de almacenamiento perteneciente a la suscripción de Azure, estructurada en *blobs*. Dicha cuenta de almacenamiento cuenta con un sistema de replicación de datos que utiliza georedundancia provista por Microsoft Azure para asegurar durabilidad y un alto índice de disponibilidad (ver Sección 4.7).

En la arquitectura propuesta, todos los roles del *Cloud Service* acceden a la cuenta de almacenamiento, con diferentes propósitos, como ser almacenamiento y recuperación de información o creación y actualización de metadatos. El acceso a la cuenta de almacenamiento se realiza directamente desde el código. Para ello, primero se debe establecer un *string* de conexión en la sección de configuración, dentro de las propiedades de cada rol; utilizando este *string* es posible acceder a la cuenta de almacenamiento mediante la clase «CloudConfigurationManager». Una vez establecida la conexión, se utiliza un cliente que permite la recuperación de *blobs* dentro de un contenedor, siendo accedido utilizando su nombre. En el sistema desarrollado para el proyecto, el contenedor principal utilizado por la aplicación web y por el clúster HDInsight se denomina *scicluster*. Adicionalmente, se utiliza un contenedor secundario denominado *publicsci* (destinado a la obtención de resultados por parte del usuario como se explicará más adelante en este capítulo). El contenedor *publicsci* cuenta con permisos de acceso menos restrictivos que el contenedor principal.

Al recuperar un *blob*, el sistema devuelve una instancia de la clase *CloudBlockBlob* que permite determinar si el mismo existe en el contenedor o no, además de leer su contenido. Es posible realizar un manejo completo de los *blobs* obteniendo la información que contiene, modificarlo cargando datos de diferentes formatos (*strings*, bytes, datos numéricos, etc) o eliminarlo.

En la aplicación también se utilizan las tablas de Azure, pertenecientes a la misma cuenta de almacenamiento. Éstas no son tablas relacionales de tipo SQL, pero permiten almacenar información estructurada con campos variables de información. En el proyecto se accede a una única tabla denominada «WADPerformanceCountersTable», utilizada en el algoritmo de

balanceo de carga para almacenar las métricas de utilización de memoria y de procesador. Para recuperar dichas métricas se realiza un proceso similar al recién descrito para *blobs*, obteniendo un cliente de tablas a través del cual se puede accedera la tabla buscada. Para la recuperación de entradas de la tabla se hace uso del objeto «TableQuery» de Azure, que permite realizar una consulta similar a una consulta SQL.

5.4.1. Metadatos

Todos los metadatos de la aplicación son generados y mantenidos en *blobs* en la cuenta de almacenamiento principal, dentro del contenedor denominado *scicluster*. A continuación se describen los *blobs* utilizados como metadatos y la información brindada por cada uno:

- **userIncrement** - Contiene el identificador de la última simulación ejecutada; cada vez que un usuario lanza una nueva simulación este *blob* es recuperado y actualizado.
- **infoSimulation-IdSimulación** - Este *blob* contiene metadatos descriptivos acerca del estado de la simulación. Se utiliza cuando aún no se han lanzado las ejecuciones de sub-tareas en el clúster, contiene únicamente el identificador de simulación. Los dos posibles estados que contiene son «Creando simulación» e «Inicializando tareas en el clúster». Estos estados son actualizados por el rol web y el rol Job Creation, respectivamente.
- «**pruebaProgreso-IdSimulación.IdJob**» - Estos *blobs* contienen información sobre el estado actual de la simulación. Cada *blob* de este tipo hace referencia a una sub-tarea específica de la simulación. Los posibles mensajes de estado son «Ejecutándose», «Tarea finalizada con éxito» y «Tarea fallida».
- «**scicluster/test/output-IdSimulación.IdJob/controloutput-m-00000**» - Este *blob* puede ser considerado como información en sí misma, pero también es correcto clasificarlo como metadato, ya que contiene información acerca de la ejecución de MCell. El *blob* se genera con la ejecución del algoritmo MapReduce para cada sub-tarea y contiene la salida en consola generada por MCell. Este metadato es particularmente útil para el usuario, ya que la aplicación le brinda la opción de descargarlo y leer la salida en consola de MCell, pudiendo determinar la causa de error en caso de ocurrir un fallo.
- «**user/admin/errorMapper-IdSimulación.IdJob.txt**» - La existencia de este *blob* refleja una falla en las tareas MapReduce pero no en la ejecución de MCell o FERnet en sí; se crea cuando se obtiene una excepción en la ejecución del *mapper* de MCell o en el de FERnet.

5.5. Cloud Service

5.5.1. Configuración

El *Cloud Service* implementado para la aplicación cuenta con un componente de código y un componente de configuración, siendo este último muy importante para desplegar el servicio en la plataforma de Azure. Azure se encarga de mantener la infraestructura del *Cloud Service*, realizar tareas rutinarias de mantenimiento, realizar *patchs* del sistema operativo y brindar recuperación ante fallos del servicio o de hardware.

Para configurar y desplegar un *Cloud Service*, se deben seguir los siguientes pasos:

1. Crear el *Cloud Service* desde el portal de administración de Azure, indicando la URL para el sitio web y la región en la cual será desplegado.
2. Cargar desde el portal de Azure un certificado especificando el archivo .pfx y la clave del mismo, como se describe en la sección 5.14, para el acceso al servicio HDInsight por parte de los roles.
3. Desplegar los distintos roles que conforman al *Cloud Service* en el cloud. Para realizar el despliegue se debe crear el paquete de servicio (.cspkg) y el archivo de configuración (.cscfg). Para la creación de estos archivos desde el IDE de Visual Studio se debe seleccionar la opción «Paquetes» desde el proyecto de Azure y luego, en el diálogo presente, se debe informar si el servicio será local o en el cloud, así como definir la configuración de despliegue: desarrollo (*staging*) o producción.
4. Realizar el despliegue en el cloud desde el portal de administración de Azure con los archivos generados en el punto anterior.

El SDK de Azure para .Net provee herramientas para preparar los archivos de desarrollo requeridos, sin necesidad de definir elementos en archivos XML, y realizando las mínimas acciones indispensables. Una de estas herramientas es *Publish Wizard*, que permite publicar la aplicación desde el propio proyecto Azure en Visual Studio haciendo clic en «Publicar» desde el menú de atajos del proyecto. Una vez que comienza el despliegue es posible monitorear el progreso y las acciones que se realizan en el proceso [7].

5.5.2. Configuración de roles

Como ya se mencionó, un proyecto Azure que conforma un *Cloud Service* posee roles de tipo web o de trabajo que pertenecen a la solución. Además, incluye archivos de definición y configuración del servicio. Los archivos de definición permiten establecer configuraciones de ejecución de la aplicación, incluyendo los roles requeridos, puntos de acceso (*endpoints*), y tamaños de las máquinas virtuales. Por su parte, los archivos de configuración del servicio

permiten definir la cantidad de instancias que se ejecutan y los valores de configuración de cada rol en particular.

Azure provee plantillas para la creación de proyectos de ambos tipos de roles. En el proyecto actual se optó por la creación de un rol web de tipo *ASP.NET Web Role* y se utilizaron roles de trabajo simples. Durante la etapa inicial de desarrollo se utilizó un rol de trabajo con *Service Bus Queue*, que luego se descartó, como se explicó en la sección 4.2.2.

Agregar un rol al *Cloud Service* resulta muy simple; desde el menú de atajos del proyecto Azure en Visual Studio se selecciona la opción *Nuevo proyecto Web Role*, o bien, *Nuevo proyecto Worker Role*, y desde la ventana de creación se selecciona la plantilla deseada.

Cada rol se compone de una clase principal *WebRole.cs* ó *WorkerRole.cs*, según el tipo de rol, y además se crean una serie de archivos entre los que se encuentran *diagnostics.wadcfg*, *packages.config* y *app.config*. Adicionalmente, al agregar un nuevo proyecto en la configuración del *Cloud Service*, se modifica el archivo *ServiceDefinition.csdef* agregando el nuevo rol y definiendo el tamaño de la máquina virtual, los puntos de acceso y demás configuraciones del rol.

Los archivos de configuración (*ServiceConfiguration.Local.cscfg* y *ServiceConfiguration.Cloud.cscfg*) se modifican automáticamente y poseen la configuración de diversos *strings* de conexión e información sobre certificados. Para modificar las opciones de configuración, Visual Studio provee una interfaz amigable. Por detrás de esta interfaz el ambiente se encarga de actualizar los archivos mencionados, acorde a las opciones ingresadas por el desarrollador.

5.5.3. Diagnostics

Azure provee una herramienta denominada *Diagnostics* que permite recuperar información de diagnóstico sobre servicios que se ejecutan en Windows Azure. La herramienta puede ser utilizada como mecanismo de depuración (*debugging*) y solución de problemas, para medir rendimiento, para monitorear la utilización de recursos, para análisis de tráfico y para auditoría. Una vez que se recolecta la información deseada, la misma puede ser transferida y almacenada en una cuenta de almacenamiento de Azure. Esta operación puede planificarse o realizarse a demanda.

La herramienta Diagnostics puede ser incorporada directamente en un *Cloud Service* y configurada en cada uno de sus roles para obtener información sobre los recursos utilizados. Para su utilización se debe importar el módulo *Diagnostics* en el modelo de servicios de cada rol en el que se desee recuperar métricas. El módulo *Diagnostics* puede ser importado agregando un elemento de tipo «Import» en el archivo *ServiceDefinition.csdef*. Por defecto, la información no es persistida directamente en una cuenta de almacenamiento, por lo que es necesario definir un *string* de conexión que

contenga el nombre y clave de la cuenta en el archivo *ServiceDefinition.csdef*.

Una vez configurado el módulo, se deben configurar los recursos de información, a partir de los cuales se obtienen las métricas para el diagnóstico. Éstos se pueden especificar en el archivo *ServiceDefinition.csdef* o en las propiedades de cada rol, accediendo a la sección «Configuración − > Diagnosticos».

En el proyecto se obtienen métricas de los roles Message Processing y Job Creation, que son utilizadas por el algoritmo de balanceo de carga. Para ambos roles se utilizó la misma configuración: se seleccionó un plan personalizado de métricas y se añadieron métricas de utilización de procesador y consumo de memoria. Se optó por un plan de persistencia planificado en el que se recolectan y se almacenan métricas en la cuenta de almacenamiento cada 3 minutos, dado que el balanceador de carga verifica el historial de los últimos 10 minutos de utilización de recursos por parte del rol. Las métricas son almacenadas en el contenedor de tablas denominado «wad-control-container» de la cuenta de almacenamiento principal, y dentro del mismo, en la tabla «WADPerformanceCountersTable».

5.6. Web Role

El proyecto Web Role forma parte de la implementación del *Cloud Service* y su función principal es la de albergar la interfaz web de la aplicación (ver figura 5.4), a la vez que gestionar la comunicación primaria entre el usuario y el resto de la aplicación. El proyecto está conformado por un componente rol de trabajo *WebRole.cs* parte del *Cloud Service* y un .Net Web Form, este último se compone de una página web conducida por eventos, fusionando HTML y controles del lado del servidor. Adicionalmente, se realizó el diseño web de la aplicación utilizando Bootstrap, un marco de trabajo de código abierto, que utiliza HTML junto con Jquery y CSS3 permitiendo crear interfaces y diseños web simples, claros e intuitivos para el usuario. En la Figura 5.5 se pueden observar las clases que componen al rol web.

La aplicación web gestiona dos funcionalidades principales, tanto del lado cliente como del servidor, que se explican a continuación.

5.6.1. Simulación

La sección principal del portal web permite ingresar datos de entrada para la simulación (ver Figura 5.6), contando con un mecanismo de auto-completado con parámetros básicos predefinidos para un manejo más dinámico. El usuario puede optar por realizar la simulación con los parámetros básicos, o gestionar sus propios parámetros de entrada a través de formularios web y controles para la adición y remoción de componentes. En la tabla 5.5, se pueden observar los parámetros configurables por el usuario en la aplicación para la ejecución de MCell; el resto de la configuración es

Figura 5.4: Previsualización aplicación Web

establecida por defecto en base a los ejemplos de simulaciones brindados junto al código fuente de los componentes de software.

Para la ejecución de FERnet es posible seleccionar el modo de ejecución. Las opciones disponibles actualmente por la aplicación son *point* o *multi*. Como se explicó en la sección 5.2, tanto el sistema MCell como FERnet se modificaron para su correcta ejecución en Windows. La ejecución de los sistemas en Windows llevó a que los modos de ejecución de FERnet *line* e *image* no funcionen correctamente, por ello no están disponibles en la solución del proyecto.

Posteriormente al ingreso de los parámetros, el usuario presiona el botón *Simular* y los datos son enviados para su procesamiento a un método web del servidor a través de una función Jquery, que realiza un llamado ajax,

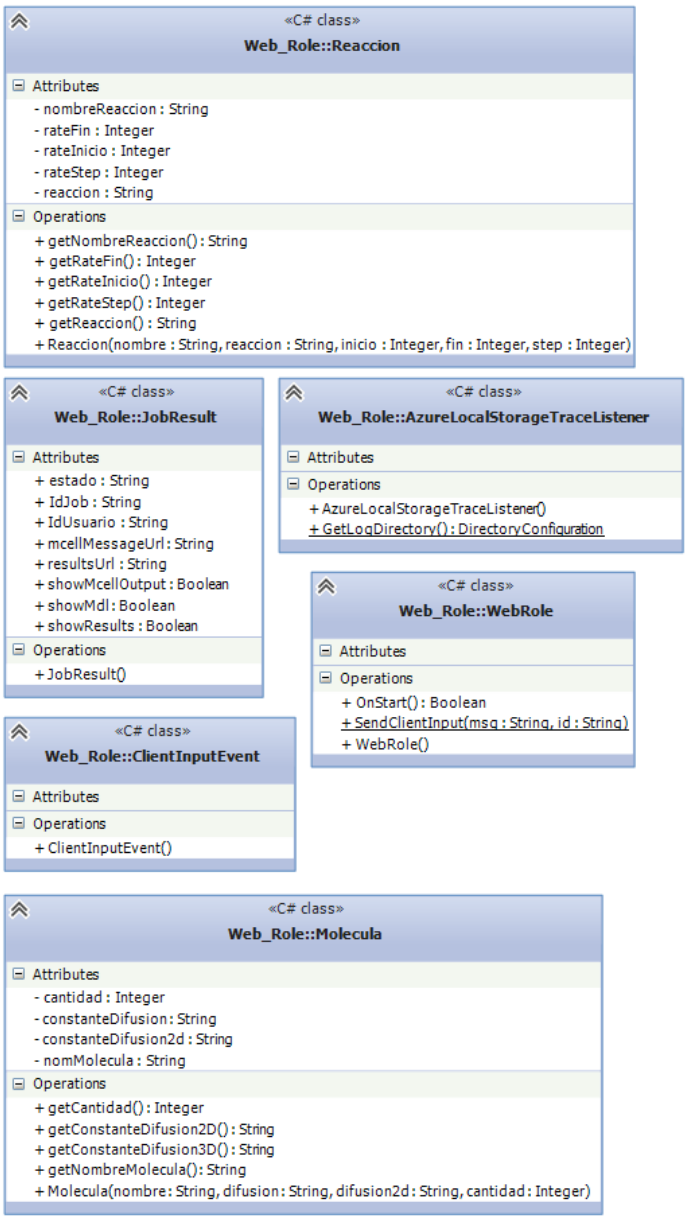

Figura 5.5: Diagrama de clases Web Role

como se puede observar en la Figura 5.7. En este llamado se envían los datos simples ingresados por el usuario, mientras que las moléculas y reacciones son mantenidas en listas a nivel de servidor. Una vez en el servidor, se genera un archivo en formato Json con todos los datos ingresados por el usuario y se

Cuadro 5.5: Parámetros de entrada MCell

Parámetro	**Descripción**
ITERATIONS	Número de iteraciones
TIME_STEP	Paso de tiempo en la simulación, en segundos.
COORDENADAS	Se ingresan dos pares de coordenadas en tres dimensiones que definen la superficie de acción de la simulación de MCell, se define la llamada *caja*
ASPECT_RATIO	Asegura que la relación del lado largo con lado corto de cada triángulo en la caja no es más que este ratio.
MOLECULAS	Permite ingresar una lista de moléculas definiendo para cada una de ellas el nombre, las constantes de difusión en 2D y en 3D y la cantidad
REACCIONES	Permite incluir una lista de reacciones que representan las interacciones entre las moléculas definidas en el punto anterior indicando para cada una nombre de la reacción, la reacción en sí y las tasas de inicio, fin y salto
NAME_LIST	Define la información de salida de MCell como cuáles moléculas se desean estudiar

envía a la clase *WebRole.cs* en un nuevo hilo del sistema, para no interrumpir la ejecución del portal web.

Como se muestra en la Figura 5.7, el código del servidor (*SciCloud.aspx.cs*) accede a la cuenta de almacenamiento y obtiene el *blob userIncrement* donde se lee e incrementa el identificador global de usuarios del sistema. Una vez enviado el Json a la clase *WebRole.cs*, se le devuelve el identificador al usuario, para monitorear el estado de sus ejecuciones y obtener los resultados. Además, se crea el *blob infoSimulation-Idsimulacion* en el que se carga el estado inicial de la simulación, que se almacena en el *Blob Storage* como metadato. Una vez que el usuario obtiene su identificador, queda liberado para lanzar otra simulación.

La clase *WebRole.cs* se encarga de la comunicación con el resto del *Cloud Service*. Cuando la clase recibe el archivo Json, comienza el establecimiento de la comunicación con el rol *Message Processing* utilizando los servicios de comunicación y aplicando el algoritmo de balanceo de carga que se describe en la sección 5.11, enviando el archivo a la instancia del rol *Message Processing* seleccionada por el balanceador de carga (ver Figura 5.7).

Figura 5.6: Aplicación web - Formulario de ingreso de parámetros

5.6.2. Obtención de resultados

La sección de obtención de resultados permite al usuario consultar el progreso de sus simulaciones, así como obtener los resultados de las mismas. Para poder consultar por resultados, el usuario debe haber realizado una simulación previamente. Como se observa en la Figura 5.8, cuando el usuario ingresa el identificador que le fue proporcionado al momento de completar los parámetros y enviar la petición, el sistema consulta el *Blob Storage* y despliega una tabla con información sobre sus simulaciones y resultados, en caso de existir los mismos.

Existen varios estados de salidas para las simulaciones, como se puede observar en la Figura 5.9. En las primeras etapas se muestra una única entrada en la tabla que indica que aún se están preparando los datos en la aplicación para ser enviados al clúster HDInsight (mientras se realiza el

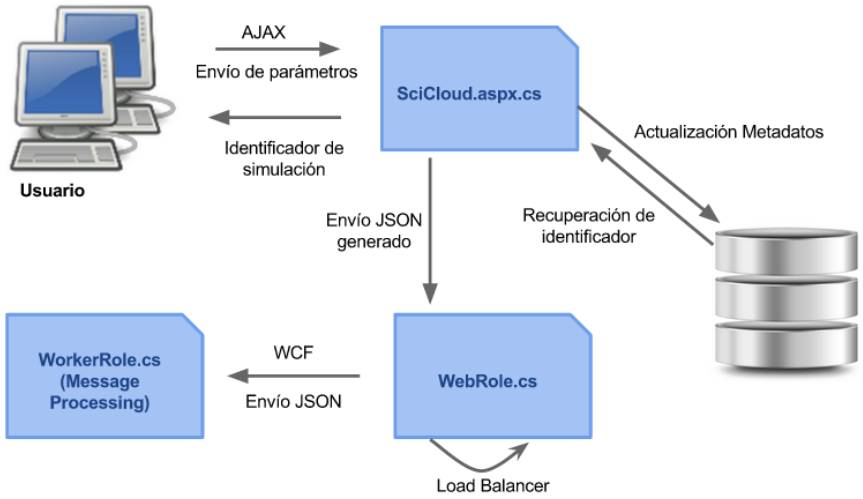

Figura 5.7: Diagrama comunicación - Simulación

Figura 5.8: Diagrama comunicación - Obtención Resultados

barrido paramétrico, crean los archivos de entrada para MCell y las tareas a ser ejecutadas por el clúster). Durante esta etapa no es posible la descarga de ningún archivo resultado. Como se observa en la Figura 5.9, los primeros estados son:

- *Creando simulación* - metadato asignado por el rol de trabajo Web Role.
- *Creando simulación* - metadato asignado por el rol de trabajo Message Processing.
- *Inicializando jobs en el clúster* - metadato asignado por el rol de trabajo Job Creation.

Las siguientes etapas muestran más información en la tabla de resultados, desplegándose una entrada por cada sub-tarea (simulación) que se haya lanzado al clúster (dependiendo de los datos de entrada y del algoritmo de

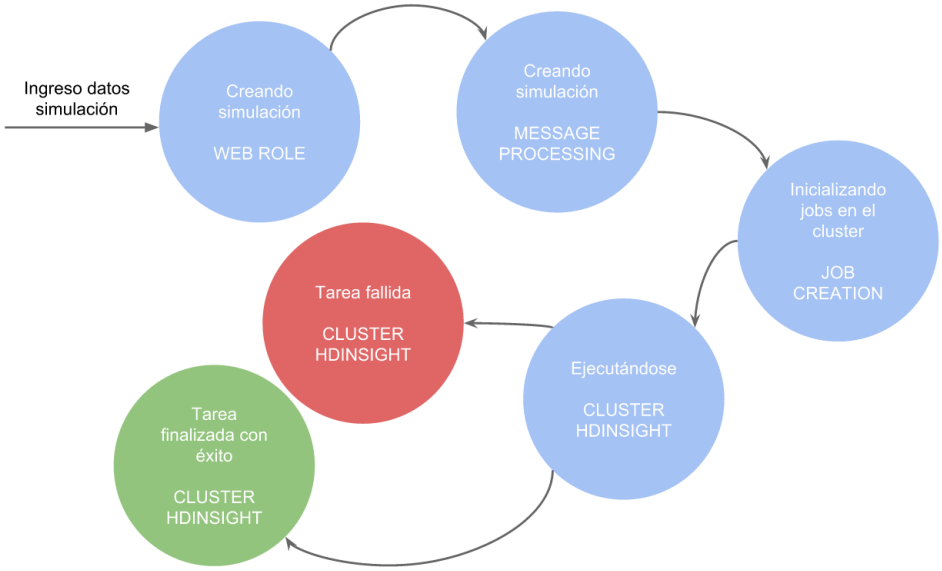

Figura 5.9: Diagrama de estados - Ejecución simulación

barrido paramétrico). Para cada sub-tarea se indica si aún se encuentra en ejecución, si finalizó con éxito o si falló. En cualquiera de estos tres estados es posible descargar el archivo con extensión MDL generado para la ejecución de la simulación. Cuando la tarea se encuentra finalizada es posible descargar el archivo conteniendo la salida en consola de MCell. Sólo en caso de éxito, al finalizar la ejecución de cada simulación, es posible descargar los resultados de la ejecución de FERnet en forma de archivo comprimido correspondiente a cada simulación.

Una vez que el usuario ingresa el identificador de simulación y realiza un pedido de resultados, el sistema se encarga de obtener los datos para esa simulación y generar la salida en el portal web. Esta acción se gestiona a través de la cuenta de almacenamiento, donde se encuentran guardados diferentes *blobs* que permiten determinar el estado en el que se encuentra la simulación. El comportamiento realizado por la clase *SciCloud.aspx.cs* para la obtención de metadatos y resultados se detalla a continuación.

- El servidor chequea la existencia del *blob pruebaProgreso-ID*, este *blob* es creado por el rol web a la hora de crear la tarea, el mismo indica el progreso de la simulación. Consultando la existencia de este *blob* el servidor logra verificar si el identificador ingresado por el usuario existe en el sistema o no. El *blob pruebaProgreso-ID* contiene un mensaje de estado y se actualiza en una serie de puntos clave de la aplicación para que el usuario pueda determinar el progreso con claridad.

- Si se verifica la existencia del *blob pruebaProgreso-ID*, se pasa a verificar la cantidad de simulaciones existentes para ese usuario. Cuando las simulaciones son lanzadas al clúster, se crea para cada una de ellas un *blob* llamado «*scicluster/test/status-ID.IdSubTarea*». Estos *blobs* contienen un mensaje de estado que puede tomar los siguientes valores: *Ejecutándose*, *Tarea finalizada con éxito* o *Tarea fallida* (ver Figura 5.9). A partir de este *blob* se gestionan los diferentes *links* de descarga a mostrar para cada simulación. Se contemplaron tres casos límite:

  1. Si la ejecución de MCell falla mostrando un mensaje en consola, no necesariamente la tarea *mapper* es fallida. Por lo que el metadato indica una ejecución exitosa cuando no lo es realmente. Para solucionar este problema, se almacena la salida de errores en consola de MCell en el *Blob Storage* y se consulta para verificar la existencia de los mismos. En caso de encontrarse un error en el archivo se genera un *blob* llamado *errormcell*. Si la clase *SciCloud.aspx.cs* verifica la existencia de este *blob*, indica que la tarea es fallida en la entrada correspondiente de la tabla de resultados.
  2. Cuando una instancia del rol Job Creation falla mientras se encuentra monitoreando una simulación, provoca que el estado de la tarea no sea actualizado y permanezca en *Ejecutándose* de forma indeterminada, cuando en realidad las simulaciones finalizan su ejecución en el clúster HDInsight. Para mitigar este problema, en el momento de obtener los resultados, si el estado actual es *Ejecutándose*, se verifica la existencia de *blobs* correspondientes a resultados de tareas o errores. Si dichos *blobs* no existen el estado *Ejecutándose* es correcto, caso contrario, se indica al usuario que la simulación se encuentra en estado *Tarea finalizada con éxito* o *Tarea fallida*, según corresponda.
  3. Cuando la ejecución de las tareas *mapper* lanza una excepción, no se genera la salida estándar de las tareas MapReduce y tampoco por tanto los archivos de existencia de errores, por lo que el sistema no es capaz de percibir el error. Para mitigar este problema, al capturar una excepción en la ejecución de la función *map*, se genera un *blob errorMapper-IdSimulacion.IdJob.txt*, que indica un fallo en la tarea MapReduce y no en la ejecución de MCell o FERnet en sí. El sistema muestra en la tabla el estado *Tarea fallida* en caso de corroborar la existencia de este *blob*.

Cuando existen archivos para descargar se genera en la tabla de resultados en el portal web, el ícono de descarga correspondiente (ver Figura 5.10). Si el usuario desea descargar uno de estos archivos, el sistema procede a buscar el contenido del mismo en el *blob* almacenado en contenedor principal del almacenamiento. Por motivos de seguridad, el nivel de acceso a dicho contenedor es restringido a usuarios que posean la clave de seguridad o el

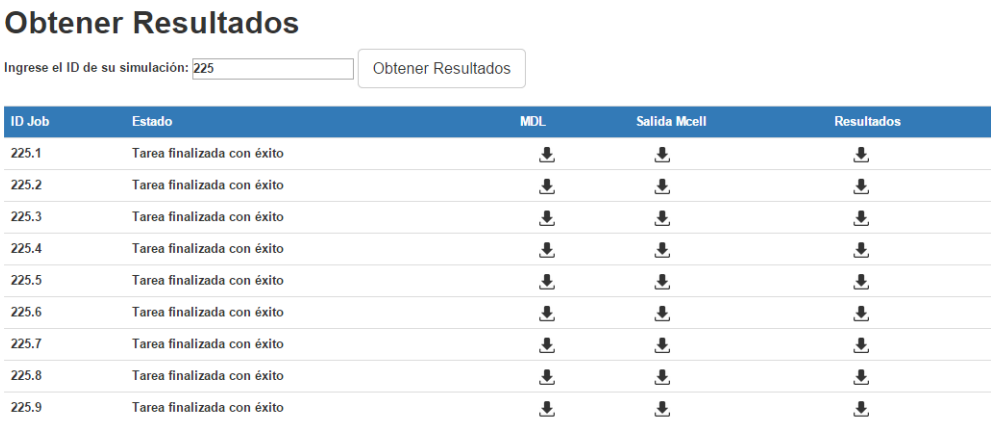

Figura 5.10: Aplicación web - Tabla de resultados para el ID 225

nombre de usuario y contraseña del administrador de Azure asociados. Por ello, para que el usuario pueda acceder al contenido del *blob* y descargarlo en su computadora, el sistema crea un nuevo *blob* con contenido idéntico en un contenedor en la misma cuenta de almacenamiento con permisos de acceso menos restrictivos, de esta forma con una simple dirección web (URL) el usuario puede realizar la descarga.

5.7. Message Processing

El rol de trabajo Message Processing es el componente de procesamiento de mensajes perteneciente al Cloud Service. El rol se conforma principalmente por tres clases: *ClientInputServiceHost.cs*, utilizada para la comunicación con otros roles que será explicada en la sección 5.10, *MDLParametros.cs* y *WorkerRole.cs*. En la Figura 5.12 se puede observar el diagrama de clases del rol.

La clase *WorkerRole.cs* es la encargada de inicializar y ejecutar el rol dentro del Cloud Service. Como se observa en la Figura 5.11, una de las tareas del rol es exponer un servicio WCF en un punto de acceso definido, para recibir mensajes del Web Role con la información ingresada por el usuario en formato Json.

El método de ejecución del rol, llamado *Run()*, cuenta con una particularidad: éste inicia el servicio y luego queda en un loop continuo; este aspecto se da ya que si se sobreescribe el método especificado y el mismo retorna (es decir, termina su ejecución), el rol es reiniciado por el *framework* de Azure, por lo que el servicio quedaría iniciándose una y otra vez [10].

Cuando la clase *WorkerRole.cs* recibe un pedido, se procede a deseria-

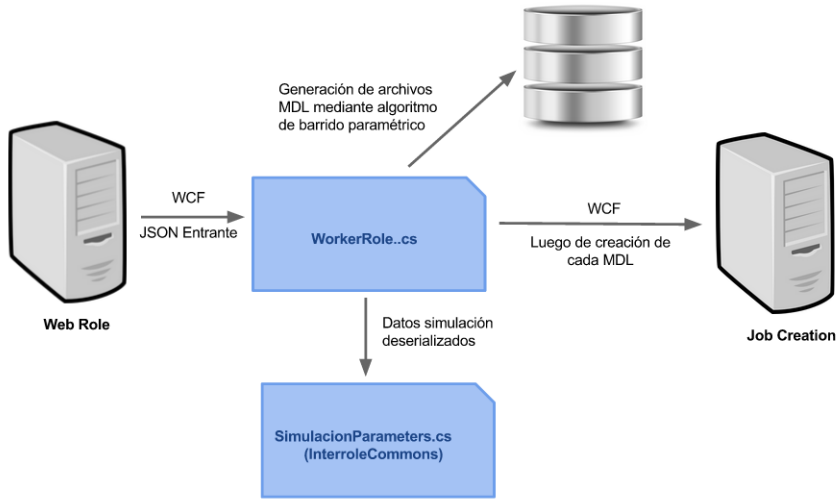

Figura 5.11: Diagrama comunicación - Message Processing

lizar el Json entrante en un objeto utilizando el marco de trabajo *Newtonsoft.Json*. Luego, se procede a la ejecución del algoritmo de barrido paramétrico detallado en la sección 5.11 sobre las reacciones definidas, en caso que hubiera alguna. Durante la ejecución de este algoritmo se generan y guardan en la cuenta principal de almacenamiento los archivos con extensión MDL correspondientes a las entradas para la ejecución de MCell (ver Figura 5.11).

Para la generación del archivo MDL se utiliza una biblioteca de generación de texto a partir de plantillas (*templates*) llamada Mustache, concretamente su versión para .Net. Se parte de un archivo base MDL en el proyecto, el mismo cuenta con una notación que indica los valores que deben ser completados programáticamente; los campos faltantes son completados por la biblioteca con un objeto que posee los mismos nombres de atributos que los solicitados en el *template* (ver Figura 5.13) [36].

El nombre de los archivos MDL generados sigue el siguiente formato: *Scene-main-Modo-IdSimulación.IdJob.MDL*, donde *Modo* es la configuración de ejecución de FERnet, *IdSimulación* es el identificador del usuario e *IdJob* es la sub-tarea dentro de la ejecución del usuario, es decir uno de los archivos generados a partir del barrido paramétrico. En caso de no existir reacciones y, por tanto, no existir ejecución del barrido paramétrico, se genera solamente un único archivo MDL y solo existe un valor para *IdJob*.

Dado que la creación de los archivos MDL toma algunos segundos, se genera un nuevo hilo para la creación de cada uno de ellos en el sistema y, de esta forma, es posible continuar con la creación de las entradas para las siguientes tareas MapReduce.

Al finalizar la creación de cada archivo de entrada para MCell se procede

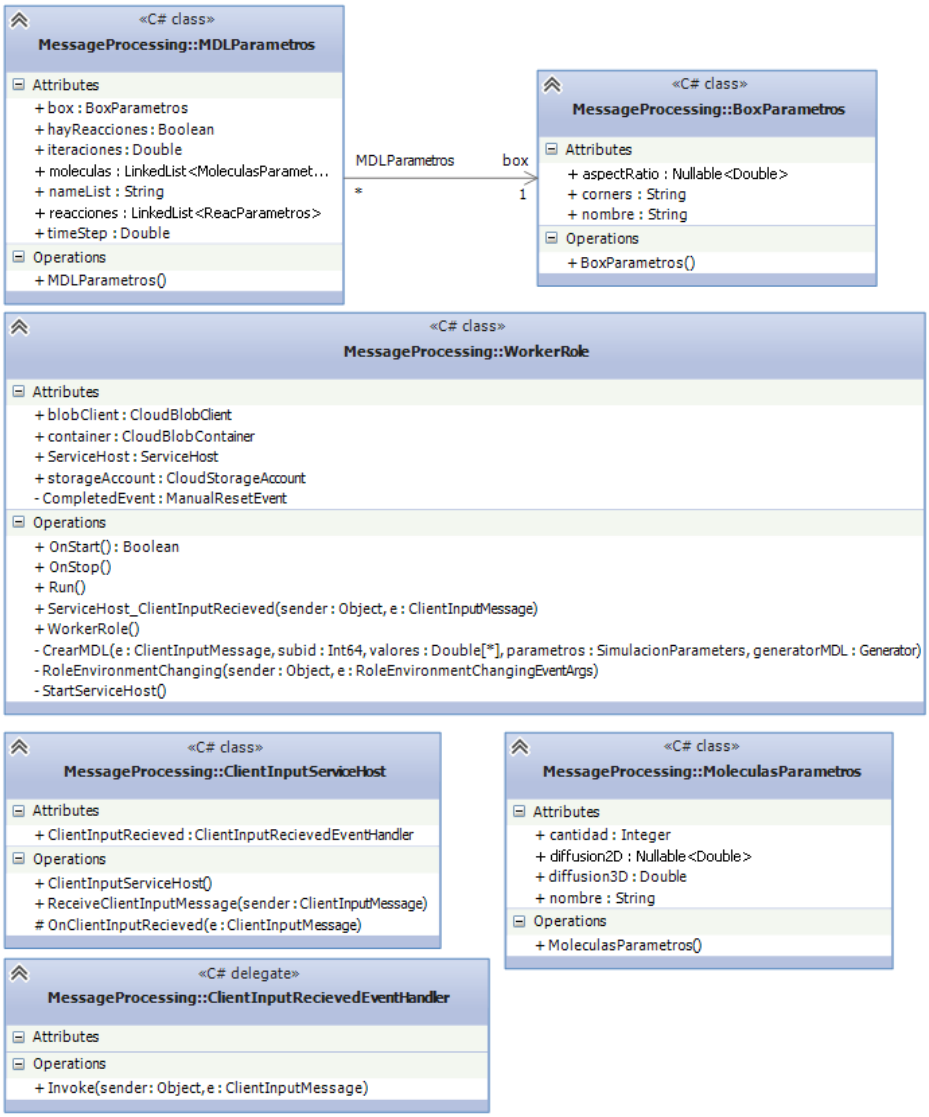

Figura 5.12: Diagrama de clases Message Processing

a realizar la comunicación con el siguiente nodo del *Cloud Service* denominado *Job Creation*. Para ello se utiliza el servicio de comunicación WCF descrito en la sección 5.10, con su correspondiente algoritmo de balanceo de carga (ver sección 5.12), y se envía el nombre del *blob* conteniendo el MDL a procesar.

Obj[r:$A + B \rightarrow C$] +′ La reacción: {{r}} ′ ⟶′ La reacción: $A + B \rightarrow C'$

Figura 5.13: Ejemplo de acción de *template* Mustache

5.8. Job Creation

El proyecto denominado *Job Creation* es un rol de trabajo perteneciente al *Cloud Service* desarrollado. Su objetivo principal consiste en recibir peticiones para la creación de tareas MapReduce y lanzar su ejecución en el clúster HDInsight. El proyecto contiene un componente para el establecimiento de la comunicación a través del servicio WCF que será explicado en la sección 5.10 y una clase principal *WorkerRole.cs*. Las clases que conforman el rol se pueden observar en la Figura 5.14.

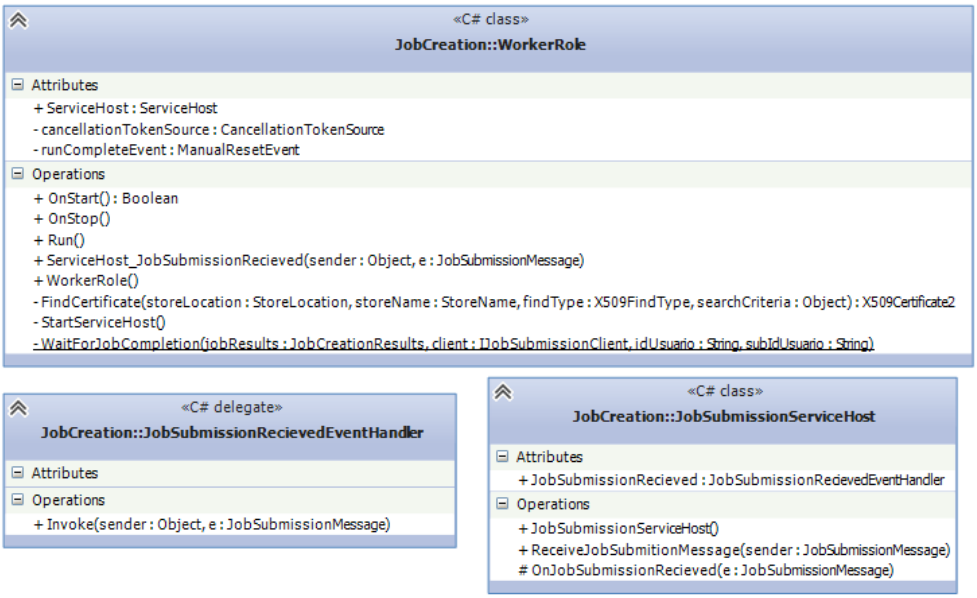

Figura 5.14: Diagrama de clases Job Creation

La clase *WorkerRole.cs* es la encargada de inicializar y ejecutar el rol dentro del *Cloud Service*, y una de sus tareas es exponer el servicio WCF en un punto de acceso definido para poder recibir mensajes del rol Message Processing con la información de la tarea a ejecutar. En el método de ejecución del rol, llamado *Run()*, se inicia el servicio y luego el rol queda en un loop continuo de la misma forma que sucede con el método *Run()*

del rol *Message Processing*. En esta misma clase, y como se ilustra en la figura 5.15, se encuentra desarrollada la funcionalidad principal del rol de trabajo en el que se procesan los pedidos entrantes, los cuales contienen los identificadores de simulación y de tarea, y el nombre del *blob* que alberga el archivo MDL necesario para la ejecución de cada simulación. Una vez extraídos estos datos del pedido se procede al envío de la tarea al clúster HDInsight y se actualizan los metadatos en los archivos correspondientes a la configuración de estados que se mostrarán en la página de resultados.

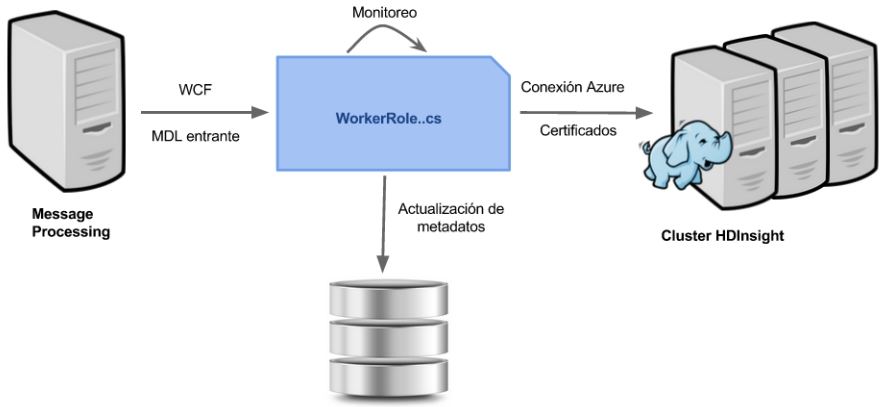

Figura 5.15: Diagrama comunicación - Job Creation

Para poder ejecutar una tarea MapReduce en el clúster, inicialmente deben configurarse los parámetros de ejecución y variables para dicha tarea. La configuración inicial incluye la ubicación en la cuenta de almacenamiento del *blob* que contiene el algoritmo MapReduce y la clase principal del algoritmo, para luego proceder al establecimiento de la conexión con el clúster HDInsight. En las siguientes sub-secciones se explica la configuración necesaria para el lanzamiento de la tarea MapReduce y la conexión segura con el clúster HDInsight.

5.8.1. Configuración de tarea MapReduce

Para la creación de una tarea MapReduce en el clúster se estudiaron dos estrategias diferentes como se detalla a continuación.

Hadoop en Streaming. Inicialmente se pensó en una configuración para la ejecución de cada tarea MapReduce con un formato en *streaming*. El mismo consiste en una modalidad para la ejecución de tareas provista por Hadoop que permite lanzar directamente cualquier archivo ejecutable o *script* como *mapper* o *reducer*. Esta configuración parecía adaptarse correctamente a los requerimientos del proyecto, donde se contaba con dos archivos ejecutables

con extensión .exe, que podían obrar como *mappers*. El problema surge a la hora de agregar los parámetros para el lanzamiento de cada tarea, sobre todo el nombre del archivo a procesar, pero estando fuera del alcance manejar el nombre de la partición del *split* a tan alto nivel. Luego de una exhaustiva investigación se llegó a la conclusión de que este mecanismo tiene la limitante de no poder configurar correctamente el pasaje de parámetros para lanzar los archivos ejecutables, y por tanto fue descartado.

Hadoop clásico. Ya que la opción anterior fue inviable, se continuó con la investigación de posibles alternativas. Finalmente se decidió por la utilización del mecanismo de ejecución de tareas tradicional o clásico. El mismo consiste en la ejecución de un algoritmo MapReduce desarrollado en Java, el lenguaje nativo de Apache Hadoop, a través de un archivo con extensión .jar. Para ello fue necesaria la implementación de un proyecto desarrollado en Java, conteniendo el algoritmo deseado. En el mismo se incluyeron dos *mappers* y un *reducer*. Las funciones *map* se encargan de lanzar las ejecuciones de cada uno de los componentes de software biológicos, con sus respectivos parámetros y el *reducer* se encarga de agrupar las salidas generadas en un único archivo comprimido, como se explicó en la sección 5.3.1. Este mecanismo de ejecución de tareas MapReduce tiene la desventaja de ser más difícil de implementar, especialmente en el escenario del proyecto donde el resto del código está escrito en C# utilizando .Net. Se debió utilizar un ambiente de desarrollo independiente con su respectiva instalación y configuración. Un aspecto positivo de Hadoop clásico es que el proyecto Java cuenta con un nivel de personalización y flexibilidad mucho más alto respecto al mecanismo en *streaming*, lo que permite realizar más acciones de control y monitoreo dentro del propio algoritmo. A continuación se muestra una porción del *script* requerido para la creación de la tarea utilizando este mecanismo.

$HadoopJob = New-AzureHDInsightMapReduceJobDefinition -JarFile "wasb:///example/Hadoop.jarClassName "MainArguments "wasb:///Scene.main.MDL-1.1.MDL", "wasb:///scicluster/output"

En el script presentado se crea una tarea, a modo de ejemplo, utilizando el mecanismo clásico de Hadoop, estableciendo la ubicación dentro del contenedor de la cuenta de almacenamiento del archivo .jar para la ejecución. Además, se configuran el nombre de la clase principal del proyecto que contiene el algoritmo MapReduce y los parámetros. Estos últimos representan el nombre del archivo MDL necesario para la ejecución de MCell y el nombre del *blob* que contendrá la salida de la tarea.

5.8.2. Comunicación con el clúster HDInsight

Continuando con el lanzamiento de la tarea en el clúster HDInsight, luego de crear la tarea desde el código se procede al establecimiento de la conexión con el clúster. Inicialmente se optó por una configuración utilizando HCatalog, una herramienta propia de Hadoop, que permite el acceso al clúster y la manipulación del sistema de archivos de Hadoop (HDFS). Se implementó un proyecto utilizando HCatalog y se logró establecer la conexión con el clúster de manera exitosa. Pero, como contrapartida, se encontró una gran limitante a la hora de acceder a los *blobs* albergados en el contenedor de almacenamiento, como consecuencia de que HCatalog no se encuentra específicamente diseñado para HDInsight de Azure. Con la utilización de HCatalog, por tanto, surgieron inconvenientes a la hora de realizar operaciones sobre la cuenta de almacenamiento de Azure, como por ejemplo: obtención de resultados, monitoreo de progresos, etc.; es por este motivo que se decidió migrar y comenzar con la utilización de un mecanismo alternativo.

Azure provee un sistema de comunicación propio del SDK para el *framework* .NET, que permite la conexión con el clúster HDInsight. Para el uso de este sistema de comunicación se debe instalar la librería *Management.HDInsight* a través de la consola de instalación de paquetes Nuget de Visual Studio; esta librería permite llevar a cabo la acción de conexión con el clúster. Luego de instalada la librería, para continuar con el proceso de conexión se deben proporcionar los siguientes parámetros:

- Identificador de la subscripción de Azure, obtenido a través de Azure Powershell.
- Huella digital, es decir, una clave que permite el acceso a través de un certificado que garantiza seguridad y confiabilidad (también puede ser gestionada a través de Azure Powershell).
- Nombre del clúster HDInsight con el que se desea establecer la conexión.
- Nombre y clave de acceso a la cuenta de almacenamiento de Azure asociada al clúster.
- Nombre del contenedor vinculado al clúster HDInsight y que funciona como sistema de archivos HDFS.

Respecto al acceso seguro, es necesario crear y configurar un certificado de administración para la comunicación con el clúster a partir del código, este aspecto se detalla en la Sección 5.14. Finalmente, se procede a la instanciación de un cliente Hadoop, con el cual se lanza la ejecución de la tarea recién descripta en el clúster HDInsight como tarea MapReduce.

5.8.3. Monitoreo

Mientras la tarea MapReduce se ejecuta en el clúster, el hilo del rol de trabajo Job Creation, que se encargó de lanzar la ejecución, queda a la espera de su finalización realizando un continuo monitoreo del estado de la tarea, a la vez que actualiza los metadatos en los *blobs* de estado para la tarea en cuestión. En esta porción de código se determina si la tarea se encuentra en ejecución o si ya ha finalizado, y de ser así es posible determinar si finalizó exitosamente o no. Una vez finalizada la ejecución de la tarea en el clúster, el hilo queda liberado.

5.9. Interrole Commons

El proyecto Interrole Commons fue creado con la intención de albergar todas las estructuras de datos, funciones, constantes y demás elementos comunes a todo el *Cloud Service*. Se compone de un rol de trabajo que se encuentra ejecutando continuamente para proporcionar accesibilidad a los elementos compartidos durante todo el ciclo de ejecución de la aplicación. Adicionalmente, el proyecto Interrole Commons cuenta con diversas clases utilizadas con propósitos variados que se describen a continuación (ver Figuras 5.16 y 5.17).

- **ClientInputMessage** - Esta clase define los datos (*Data Contract* de WCF) utilizados para establecer la comunicación entre el rol web y el rol de trabajo Message Processing. La clase consta de un identificador de usuario y el mensaje enviado, el cual está formado por el Json con los parámetros especificados por el usuario.
- **IReceiveMessageFromClient** - Interfaz que define el contrato de servicio WCF expuesto por el rol Message Processing, donde se especifica una operación *ReceiveClientInputMessage* que recibe un objeto de la clase *ClientInputMessage*. Esta función será llamada por el Web Role para enviarle los parámetros del usuario para armar las entradas a las simulaciones.
- **JobSubmissionMessage** - Esta clase define los datos (*Data Contract* de WCF) utilizados para establecer la comunicación entre el rol Message Processing, y el rol Job Creation. Estos datos son el identificador de simulación, el identificador de tarea y el nombre del archivo de entrada con extensión MDL.
- **IReceiveJobSubmission** - Interfaz que define el contrato de servicio WCF expuesto por el rol Job Creation donde se especifica una operación *ReceiveJobSubmitionMessage*, que recibe un objeto de la clase *JobSubmissionMessage*.
- **LoadBalancerCommons** - Esta clase alberga constantes de configuración del algoritmo de balanceo de carga y constantes útiles para la

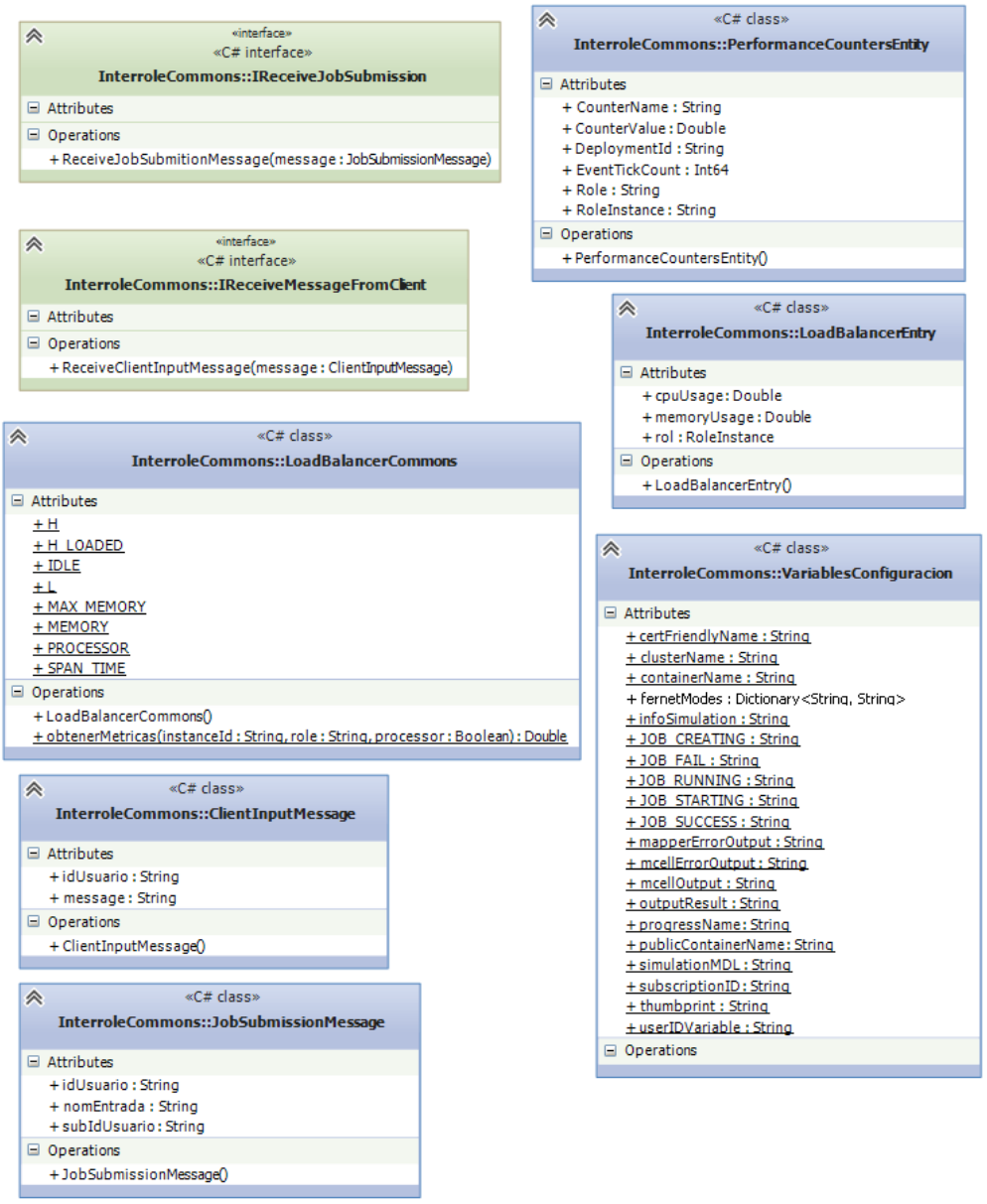

Figura 5.16: Diagrama de clases Interrole Commons A

obtención de métricas. A su vez, la clase contiene una función genérica utilizada por el algoritmo para obtener el promedio de una de las métricas para una instancia dada, los detalles específicos se presentan en la Sección 5.12.

- **LoadBalancerEntry** - Esta clase contiene las entradas necesarias para almacenar las métricas promedio de utilización de procesador y memoria para una instancia de rol, en un instante dado de la simulación.
- **PerformanceCountersEntity** - Esta clase representa una entrada

en la tabla *WADPerformanceCountersTable* de la cuenta principal de almacenamiento, que es utilizada para guardar cada una de las métricas de utilización de recursos configuradas en la aplicación. Cada variable de la clase coincide con un atributo de dicha tabla, y se utiliza para obtener registros de la misma en el algoritmo de balanceo de carga.

- **SimulacionParameters** - En este archivo están contenidas todas las estructuras de datos necesarias para almacenar la información ingresada por el usuario en el sistema para las diferentes configuraciones de MCell y FERnet.
- **VariablesConfiguracion** - El objetivo de este archivo es el de almacenar constantes generales de la aplicación como el nombre del clúster HDInsight, los nombres de los *blobs*, la información sobre los certificados utilizados, etc. El archivo también se utiliza para albergar variables utilizadas en otros roles del sistema, como el modo de ejecución de FERnet, por ejemplo.

5.10. Servicios de comunicación

Las instancias de los roles en un *Cloud Service* de Azure se comunican a través de conexiones internas y externas, dependiendo del tipo de comunicación. Una conexión externa que sirve para comunicar al rol con clientes externos se llama punto de acceso de entrada (*input endpoint*), y una conexión interna que se utiliza para comunicar a los roles internamente se llama punto de acceso interno (*internal endpoint*). Se puede utilizar los protocolos HTTP, HTTPS o TCP para establecer una conexión con un punto de acceso de entrada, mientras que para el establecimiento de la conexión un punto de acceso interno se permite la utilización de los protocolos HTTP o TCP.

Los puntos de acceso se asocian a direcciones IP y números de puerto, donde los puertos de los puntos de acceso de entrada pueden ser especificados por el administrador y los puertos de los puntos de acceso internos son asignados por la plataforma Azure.

Cada punto de acceso de entrada para un rol define un único puerto donde el rol escucha llamados. Este puerto definido para la comunicación del rol con el mundo exterior es utilizado por el componente de balanceo de carga de Azure, de manera que el servicio queda expuesto y disponible en Internet.

En un *Cloud Service* se pueden desplegar un máximo de 25 roles de trabajo o roles web y un máximo de 25 puntos de acceso de entrada que se distribuyen entre los posibles roles. Es decir, se puede incluir en un *Cloud Service* 25 roles de trabajo o roles web con un punto de acceso de entrada cada uno o un solo rol web o rol de trabajo con 25 puntos de acceso de entrada.

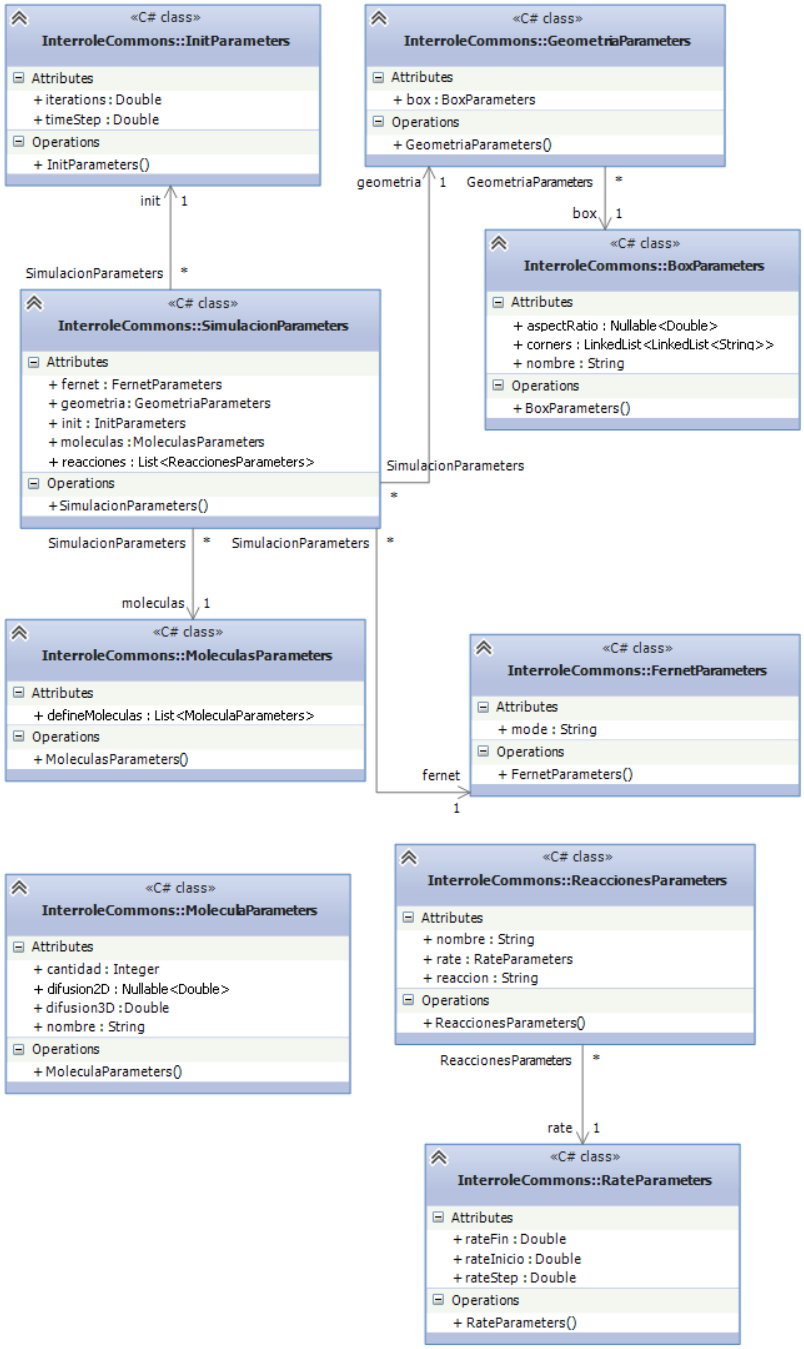

Figura 5.17: Diagrama de clases Interrole Commons B

Cabe destacar que las conexiones internas entre roles se deben generar entre los componentes del rol que forman parte del *Cloud Service* (las clases

que implementan el rol web o el rol de trabajo). Por ejemplo, el código que implementa el cliente web en el rol web no puede comunicarse con otro rol, sino que debe hacerlo a través del componente Web Role (*WebRole.cs*).

5.10.1. Puntos de acceso de entrada

De todos los roles que conforman el *Cloud Service*, el rol web es el único que posee un punto de acceso de entrada, donde se escuchan llamados HTTP en el puerto 80, y que hace disponible para su acceso a través de Internet al sitio web accedido por el usuario para iniciar las simulaciones. Este punto de acceso de entrada se crea por defecto al crear un rol de tipo web en Visual Studio.

5.10.2. Puntos de acceso internos

Para la comunicación interna entre roles se definen puntos de acceso internos en los cuales se enlazan servicios WCF que escuchan en ese puerto. Cuando un rol desea realizar un llamado a un servicio WCF expuesto por otro rol, obtiene mediante una clase denominada *RoleEnvironment* (clase provista por el SDK de Azure para obtener parámetros de configuración, recursos locales e información de puntos de acceso) las diferentes instancias de los roles. Posteriormente, el rol aplica el algoritmo de balanceo de carga implementado y realiza el llamado WCF a la instancia seleccionada.

El *Cloud Service* implementado para el proyecto cuenta con dos puntos de acceso internos:

- **ClientInputServiceEndPoint** - Es el punto de acceso interno expuesto por el rol de trabajo Message Processing. En este punto de acceso se exhibe un servicio WCF *oneway* (un solo sentido), donde se reciben los parámetros ingresados por el usuario para correr la simulación.
- **JobSubmissionServiceEndPoint** - Es el punto de acceso interno expuesto por el rol de trabajo Job Creation. En este punto de acceso se exhibe un servicio WCF en un solo sentido (*oneway*) donde se reciben el nombre del archivo MDL generado, el identificador de simulación (usuario) y el identificador de tarea para crear y ejecutar la tarea MapReduce en el clúster HDInsight.

Creación de punto de acceso en un rol de trabajo

Para la creación de un punto de acceso en un rol de trabajo se debe acceder desde Visual Studio a las propiedades del proyecto y particularmente a la sección de puntos de acceso. Como se puede observar en la Figura 5.18, se crea un nuevo punto de acceso de tipo interno con soporte de protocolo TCP y se brinda un nombre. El nombre, *ClientInputServiceEndPoint* en el

caso de la Figura 5.18, es utilizado posteriormente para realizar el llamado WCF por el rol emisor. El puerto se configura como dinámico ya que es asignado por la plataforma de Azure.

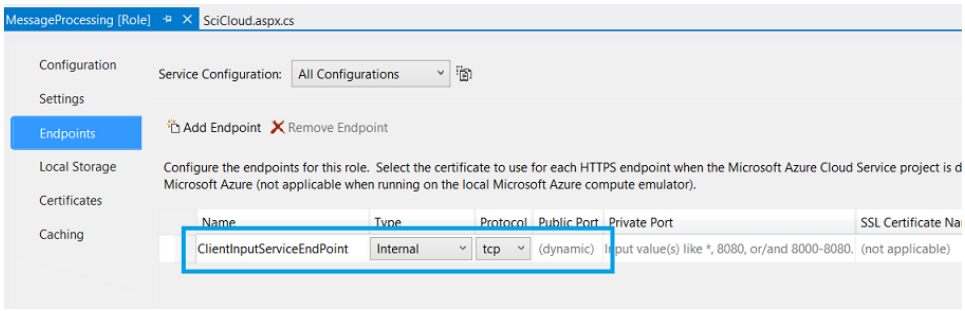

Figura 5.18: Creación de punto de acceso en rol de trabajo con Visual Studio.

5.10.3. Servicios WCF

Para la definición de los servicios WCF se utilizó un enfoque basado en eventos. Se definen en el rol Interrole Commons los contratos de servicio y de datos para WCF, ya que se utilizan tanto por el rol que envía el mensaje como por el rol que lo recibe.

Servicio WCF Recibir mensaje de usuario

Se define en el proyecto del rol Interrole Commons el contrato de servicio en la clase *IReceiveMessageFromClient.cs*, donde se especifica el método que se podrá llamar mediante WCF. En el proyecto Interrole Commons, se define también el *Data Contract* especificado en la clase *ClientInputMessage.cs*, que define la estructura del mensaje a enviar. En el rol Message Processing se implementa la interfaz *IReceiveMessageFromClient.cs* mediante la clase *ClientInputServiceHost.cs*, donde se define el evento disparar cuando se recibe un mensaje.

En la clase *WorkerRole.cs* del rol se define una instancia del *ClientInputServiceHost* y se mapea al punto de acceso interno definido para este servicio (*ClientInputServiceEndPoint*), dejando así al servicio disponible para la comunicación con otros roles. Se define también el *handler ServiceHost_ClientInputRecieved* (manejador de eventos) que será llamado cuando se reciba un mensaje en el servicio expuesto. En la Figura 5.19 se pueden apreciar las clases mencionadas.

Se define en el proyecto del rol Interrole Commons el contrato de servicio en la clase *IReceiveJobSubmission.cs*, donde se especifica el método que se

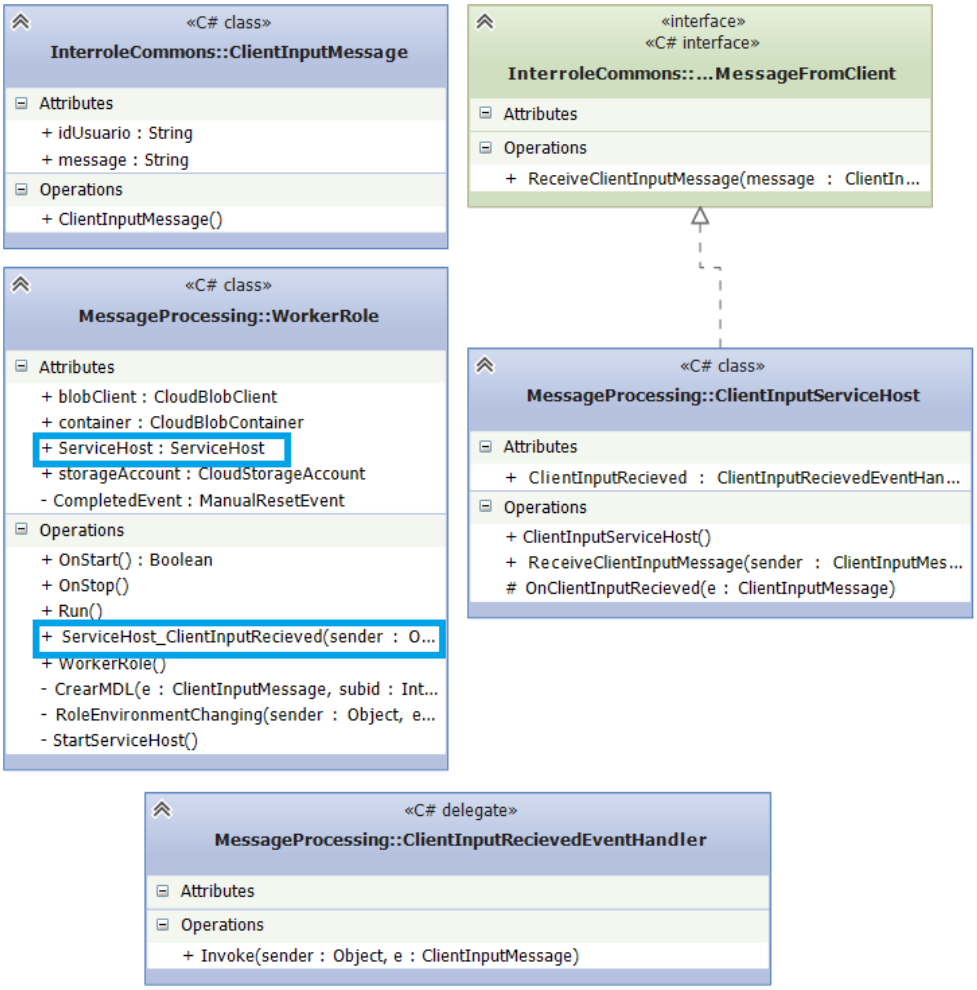

Figura 5.19: Diagramas de clase para la definición del servicio WCF para recepción del mensaje del usuario.

podrá llamar mediante WCF. El *Data Contract* especificado en la clase *JobSubmissionMessage.cs* del proyecto Interrole Contracts, define la estructura del mensaje a enviar. En el proyecto del rol Job Creation se implementa la interfaz *IReceiveJobSubmission.cs* mediante la clase *JobSubmissionServiceHost.cs*, donde se define un evento el cual se dispara cuando se recibe un mensaje.

En la clase *WorkerRole.cs* del rol, se define una instancia del *JobSubmissionServiceHost* y se mapea al punto de acceso interno definido para este servicio (*JobSubmissionServiceEndPoint*), dejando así el servicio disponible para la comunicación con otros roles. Se define también el *handler ServiceHost_JobSubmissionRecieved* (manejador de eventos) que será llama-

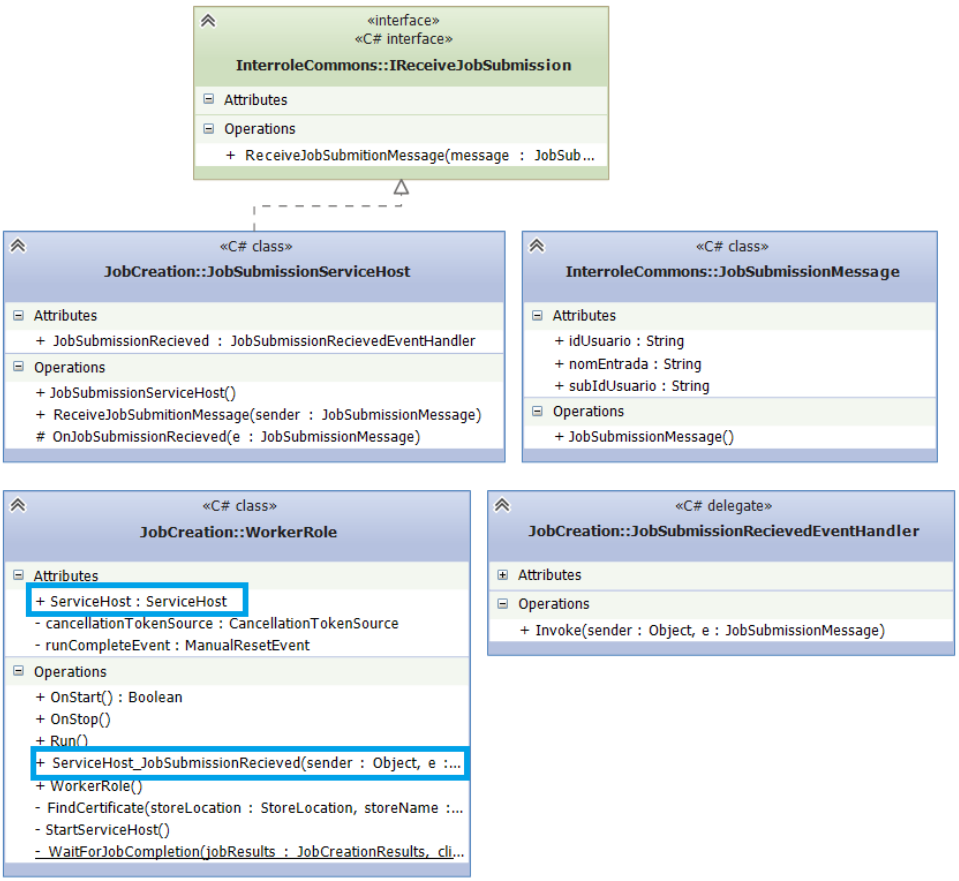

Figura 5.20: Diagramas de clase para la definición del servicio WCF envío de tarea.

do cuando se reciba un mensaje en el servicio expuesto. En la Figura 5.20 se pueden apreciar las clases mencionadas.

Envío de mensajes utilizando WCF

Los dos llamados a WCF que se realizan en el sistema se dan cuando el Web Role invoca al servicio WCF expuesto por el rol de trabajo Message Processing para el envío de parámetros brindados por el usuario, y posteriormente cuando el rol Message Processing invoca al servicio expuesto por el rol de trabajo Job Creation para enviarle la información de la tarea que debe lanzar en el clúster HDInsight. Ambos llamados se manejan de la misma manera, tal como se explica genéricamente a continuación:

1. Se elige una instancia del rol al cual se desea realizar el llamado WCF aplicando el algoritmo de balanceo de carga especificado en la Sección 5.11.

2. Se obtiene la información del punto de acceso a partir de su nombre y se crea un canal que se conecta a la dirección del punto de acceso.
3. Se llama a la función que envía el mensaje utilizando la estructura definida en el *Data Contract* correspondiente mediante el canal.

Servicio WCF Recibir envío de tarea

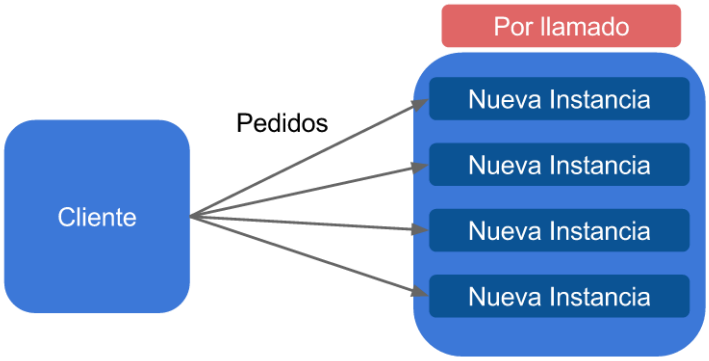

Figura 5.21: Modelo de envío por llamado

Para el envío de mensajes se utilizó la configuración de WCF denominada por llamado («PerCall»). Como se muestra en la Figura 5.21, cada vez que se requiere el envío de un mensaje se crea una nueva instancia del servicio *ClientInputServiceHost*. Una vez que el pedido es enviado, dicha instancia se elimina automáticamente. Este tipo de llamado permite un envío de mensajes donde no se requiere un servicio con preservación de estado, como es el caso del sistema desarrollado. Adicionalmente, cuando la información que se envía requiere de elevado tiempo de transferencia, este mecanismo resulta apropiado ya que permite el envío concurrente sin generar *overhead* por la espera de la instancia *singleton* única y central, como sería el caso al utilizar envíos de tipo «Single». Utilizando servicios «PerCall» se logra una arquitectura ampliamente escalable al aumentar la cantidad de pedidos entrantes.

5.11. Barrido paramétrico

Dentro de los parámetros que el usuario puede definir a la hora de lanzar una simulación MCell se encuentra la definición de las distintas reacciones, es decir cómo actúan las moléculas o especies cuando se acercan entre sí. Cada reacción tiene una probabilidad de suceder (*rate*), es decir, una reacción $A + B \longrightarrow C$ no implica que el %100 de las veces que una molécula A se acerque a una B, ambas reaccionan para formar una molécula C. El

parámetro *rate* podría no ser bien conocido. Es por tanto de interés brindarle al usuario la posibilidad de definir un rango de valores y el paso para ir del valor inferior al superior para estos parámetros *rate* dentro de una definición de reacción. Para ello es necesario brindar un algoritmo de barrido paramétrico que obtenga distintas combinaciones de los rangos de valores para así ejecutar las posibles simulaciones en paralelo corriendo en el clúster HDInsight.

El algoritmo de barrido paramétrico sólo se ejecuta en caso que el usuario haya definido al menos una reacción, y que esta reacción o reacciones definidas sean unidireccionales, como el ejemplo brindado al comienzo de esta sección. Al ser las reacciones unidireccionales, existe solamente un parámetro a barrer (el *rate*) por cada una de ellas. En caso de definir una reacción bidireccional para la ejecución de MCell se brindan dos valores de *rate*, pero esta funcionalidad no es soportada en la versión actual del proyecto.

5.11.1. Algoritmo desarrollado

Se realizó un relevamiento de trabajos donde se plantean problemas resueltos utilizando barrido paramétrico, pero fue de especial interés una tesis de doctorado realizada en la Universidad Politécnica de Cataluña [39], donde se busca definir un workflow genérico para el modelado de problemas de barrido paramétrico en sistemas distribuidos. En el trabajo de doctorado se definen ciertas variables para describir y resolver el problema de barrido paramétrico para modelos científicos:

- **v** - define un vector de resultado de aplicar una determinada función a una selección de parámetros. En el caso de la implementación actual representa el resultado de la ejecución de MCell utilizando una cierta combinación de parámetros.
- **n** - es el número de parámetros a los que se desea aplicar el barrido.
- **p** - es un vector cuyos componentes son los valores de los **n** parámetros que producen los resultados en **v**. En la implementación actual, corresponde a la especificación de los distintos valores de *rate* para cada reacción que desencadenan el resultado **v** al ejecutar MCell. $v = v(p)$
- **j** - Un determinado parámetro sólo puede adquirir uno de varios valores. Cada parámetro se corresponde con una serie de **m** valores que se pueden considerar identificados por un índice, con 0 para el valor mínimo y **m**−**1** para el valor máximo. Se define por tanto **j** como un vector que especifica el valor del índice para cada uno de los **n** parámetros. $p = p(j)$ y por tanto $v = v(p(j))$, es decir que **v** queda determinado por el vector de índices de los posibles valores que pueden tomar los parámetros. Este vector de índices se puede utilizar para definir las distintas combinaciones de valores de parámetros.

Se define por otro lado una matriz **P** de **nxm** que especifica todos los posibles valores que puede tomar cada parámetro, donde **n** es el número de parámetros y **m** es el número de valores de cada parámetro por cada índice **j**.

$$P = \begin{bmatrix} p_{00} & \cdots & p_{0(n-1)} \\ \cdots & \cdots & \cdots \\ p_{(n-1)0} & \cdots & p_{(n-1)(m-1)} \end{bmatrix}$$

Es importante mencionar que el valor de **m** no tiene porqué ser igual para cada parámetro, ya que el valor de **m** dependerá del rango de valores y el paso especificado por el usuario. **P** es por tanto un *array* (arreglo) multidimensional.

Algorithm 1 Utilización del vector lógico L

if $(L_k \neq j_k, \forall k = 0, (n-1))$ **then**
  P, barrido paramétrico total
else
  subconjunto de P, barrido paramétrico parcial
end if

Dependiendo de la cantidad de reacciones definidas y del rango de valores brindado, la cantidad de posibles combinaciones de estos valores puede llegar a ser muy grande. Por lo tanto, en el trabajo relevado se presenta una manera de realizar una selección parcial utilizando un vector lógico L que especifica, para cada parámetro **n**, un índice de los posibles valores de **n** a no tener en cuenta para armar las distintas combinaciones. Si **j** presenta algún valor de **n** que coincide con el valor de L para ese parámetro **n**, ese vector **j** se descarta. El Algoritmo 1 muestra el uso del vector lógico L en la implementación del barrido paramétrico.

Se implementa entonces el barrido paramétrico sugerido por la tesis [39] adaptándolo al problema del proyecto actual, mediante el procedimiento descrito en el Algoritmo 2.

En las líneas $1-3$ se inicializan los vectores de valores iniciales, finales y pasos que definen los posibles valores para cada parámetro, iterando sobre estos valores se obtiene la matriz P mencionada anteriormente. En la línea 4 se define el vector lógico **L** que filtra combinaciones para realizar un barrido selectivo. En las líneas $5-11$ se inicializa el vector **j** con los valores iniciales de cada parámetro y se realiza el filtrado con el vector lógico **L**, implementando en estas líneas el Algoritmo 1.

Algorithm 2 Barrido paramétrico

```
1: Se inicializa vector con el valor inicial de cada reacción
2: Se inicializa vector con el valor final de cada reacción
3: Se inicializa vector con los pasos (steps) de cada reacción
4: Se define el vector lógico L
5: j ← [n]
6: for k = 0 to n − 1 do
7:     if k = vectorLogico[k] then
8:         iniciosArray[k] ← iniciosArray[k] + stepsArray[k]
9:     end if
10:    j[k] = iniciosArray[k]
11: end for
12: subid = 1
13: z = n − 1
14: while z >= 0 do
15:    if (j[z] − finesArray[z]) ∗ stepsArray[z] > 0 then
16:        j[z] = iniciosArray[z]
17:        z = z − 1
18:    else
19:        NuevoHilo(CrearMDL(subid, parametros))
20:        z = n − 1
21:        subid = subid + 1
22:    end if
23:    if z >= 0 then
24:        j[z] = j[z] + stepsArray[z] +
25:        if z = vectorLogico[z] then
26:            j[z] = j[z] + stepsArray[z]
27:        end if
28:    end if
29: end while
```

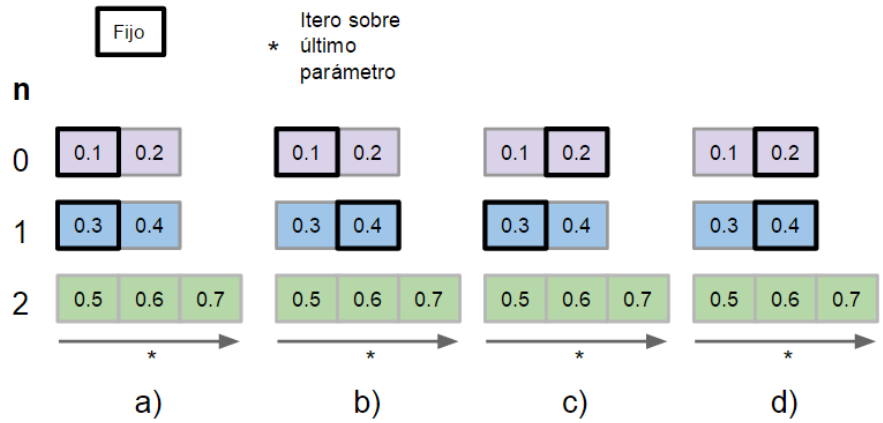

Figura 5.22: Iteraciones en el algoritmo de barrido paramétrico.

La modalidad seguida es iterar sobre un parámetro con índice z (donde z varía entre 0 y $n-1$), dejando todos los parámetros $z-1$ anteriores fijos, como se puede observar en la Figura 5.22 - diagrama *a)*, donde z es el último parámetro. Cuando se finaliza la iteración sobre el parámetro z, se incrementa el parámetro *z-1* con el paso definido y se procede a realizar nuevamente la iteración sobre z (ver Figura 5.22 - diagrama *b)*). Se decrementa de igual manera z hasta abarcar todos los parámetros (Figura 5.22 - diagrama *c)* y *d)*).

Para implementar esta modalidad se realiza un *loop* (línea 14) comenzando a iterar sobre el último parámetro, $z = n-1$ (línea 13), donde se van procesando las distintas combinaciones de la siguiente manera:

- En las líneas $15-17$ se verifica que el valor de **j** para el parámetro **z** no haya sobrepasado el valor final especificado en el arreglo definido en la línea 2, es decir, verifica si se procesaron todos los valores posibles para el parámetro **z**. Si se sobrepasa el valor final, se reinicia la variable **j** con el valor inicial del parámetro **z** definido en el arreglo de valores iniciales especificado en la línea 1. Se decrementa **z** para trabajar sobre el parámetro anterior.
- En caso de no sobrepasarse el valor final, en las líneas $18-22$, se lanza la ejecución de la creación del archivo MDL en un nuevo hilo con los valores del vector **j**, asignándole un *subid* que identificará a la tarea. Luego, se reinicia el valor de **z** al último parámetro para comenzar una nueva iteración.
- Dado cualquiera de los dos puntos anteriores, si el valor de **z** es mayor que cero, es decir, si quedan parámetros para procesar, se incrementa el valor de **j** para ese valor **z**, utilizando el valor de paso definido en el arreglo especificado en la línea 3, como se puede observar en las líneas

$23 - 28$. En este incremento se toma en cuenta el vector lógico **L** para realizar el filtrado (líneas $25 - 27$).

Se muestra a continuación un ejemplo simple de ejecución del algoritmo.

Cuadro 5.6: Ejemplo de barrido paramétrico

Reacción	**Inicio**	**Fin**	**Paso**
$A + B \rightarrow AB$	0.4	0.5	0.1
$AB + B \rightarrow A$	0.6	0.7	0.1

En el caso de ejemplo definido en el Cuadro 5.6, se tienen dos reacciones y un vector lógico con valores lo suficientemente altos para no filtrar ninguno de los valores especificados.

1. Inicialmente el vector **j** se inicializa como $j = [0.4, 0.6]$ y se comienza a iterar sobre la segunda reacción.
2. Se crea el MDL para el vector **j** inicial y se pasa a incrementar el segundo parámetro utilizando el paso definido, obteniendo $j = [0.4, 0.7]$.
3. Se crea el MDL para el nuevo vector **j**, y como se llega al final de los valores posibles para el último parámetro, se vuelve a setear en su valor inicial y se incrementa el primero utilizando el paso definido, obteniendo $j = [0.5, 0.6]$.
4. Se crea el MDL para el nuevo vector **j** y se pasa a incrementar el segundo parámetro utilizando el paso definido, obteniendo $j = [0.5, 0.7]$.
5. Se crea el MDL para el nuevo vector **j** y debido a que se llega al final de los valores posibles para el primer parámetro, el algoritmo finaliza.

La creación del archivo MDL para cada combinación de parámetros se realiza concurrentemente utilizando hilos, ya que el procesamiento de cada archivo es independiente de los demás (*trivialmente paralelo*). Cada hilo se encarga de crear el archivo MDL y enviarlo mediante un mensaje WCF al rol *Job Creation* para la configuración y creación de la tarea en el clúster HDInsight.

5.12. Balance de carga

Cuando los sistemas distribuidos se vuelven más complejos, los desafíos de estructurar una aplicación que utiliza servicios WCF se vuelven mayores. Existen muchos aspectos a tener en cuenta en este sentido. Algunos de ellos involucran: cómo distribuir el tráfico de la red, cómo balancear la carga de los servidores, cómo realizar un correcto manejo de fallos y cómo implementar un sistema seguro.

Si muchos clientes se conectan de forma simultánea, la carga que se genera sobre los servicios WCF comienza a crecer, y más aún en el caso

de la aplicación implementada en este proyecto, donde para cada cliente se efectúan diversas comunicaciones WCF. A raíz de esto, se plantea una pregunta clave: "¿cómo manejar aquellas circunstancias donde la demanda es muy alta?". Una solución común consiste en agregar a la infraestructura la implementación de un balanceador de carga. La tarea de un balanceador de carga consiste en tomar los pedidos entrantes y distribuirlos a través de un número de servidores que ejecutan un mismo servicio. Esto permite evitar que un servidor específico se vea sobrecargado con pedidos, lo que podría afectar de forma significativa el rendimiento de un servicio WCF. De esta forma se busca distribuir el tráfico entre los distintos servidores de forma equitativa.

A la hora de implementar un balanceador de carga para servicios WCF se deben tener en cuenta diversas consideraciones, entre las cuales se incluyen: requerimientos de desempeño, aspectos de seguridad y utilización de sesiones [13]. A continuación se describe el algoritmo de balanceo de carga empleado en la aplicación.

5.12.1. Investigación

Se procedió a investigar diferentes algoritmos de balanceo de carga y se identificaron tres grandes categorías.

- **Algoritmos estáticos** Los algoritmos estáticos realizan su decisión de selección de servidor de forma previa a la ejecución del algoritmo. Se basan en información sobre el comportamiento general del sistema y la selección del servidor es independiente del estado actual y de las posibles fluctuaciones en tráfico y carga. Esto podría llevar a que en determinados momentos la carga en algunos servidores se vuelva muy alta, mientras que en otros se mantenga cercana al mínimo. Los algoritmos estáticos utilizan de forma estricta un conjunto de reglas fijas y pre-configuradas. Ejemplos de este algoritmo incluyen análisis por características generales del tráfico en la red, asignación aleatoria y Round Robin.
- **Algoritmos dinámicos** Los algoritmos dinámicos distribuyen en tiempo de ejecución la carga de trabajo entre los servidores. Realizan decisiones de transferencia en base al estado actual del sistema a través de la obtención y el monitoreo de métricas dinámicas, como ser: niveles de carga en canales de salida o consumo de recursos. Luego realizan una distribución de pedidos en tiempo real en base a dichas métricas con el objetivo de equilibrar la carga. Es por ello que la implementación de un algoritmo dinámico se vuelve naturalmente más compleja. Algunos ejemplos de este tipo de algoritmo son grafos de particionamiento paralelo y el método de selección del más veloz [24].

- **Algoritmos híbridos** Los algoritmos híbridos utilizan estrategias que combinan características genéricas del sistema con métricas dependientes de la ejecución actual. Un ejemplo de algoritmo híbrido es Round Robin con asignación de peso.

Al realizar el relevamiento de las tecnologías Microsoft disponibles, se encontró que Azure provee al usuario de una implementación de balanceador de carga para gestionar el tráfico entre las instancias de sus roles. El mismo se denomina ILB (siglas en inglés para 'Balanceador de Carga Interno') y consiste en un algoritmo de tipo Round Robin. Este tipo de algoritmo se encarga de obtener todos los servidores que se encuentran ejecutando un mismo servicio y distribuir los pedidos entrantes de forma circular entre dichos servidores. Además, el algoritmo de balanceo de carga Round Robin es estático. En el caso del proyecto, se consideró que la utilización del ILB no era la solución apropiada. Esto se debe a que en el caso del problema abordado las simulaciones pueden requerir tiempos de cómputo disímiles, dependiendo de los parámetros de entrada brindados por el usuario. Es por ello que ante una alta carga de simulaciones, la eficiencia del algoritmo puede ser limitada, al asignar los pedidos de forma secuencial a cada instancia de servidor sin tener en cuenta el consumo o utilización de recursos.

5.12.2. Algoritmo desarrollado

Dadas las características del problema, con tiempos de ejecución variables, alto consumo de procesamiento y ambiente homogéneo (es decir, todos los servidores que ejecutan un determinado tipo de servicio poseen igual poder de cómputo) se optó por la implementación de un algoritmo dinámico. Cada rol de trabajo tiene una o más instancias de sí mismo, donde cada una de ellas cuenta con su propio procesador y memoria. Para cada instancia del rol que se quiere balancear se obtienen y almacenan las métricas de consumo de procesador y de utilización de memoria, para determinar la carga de cada instancia. Se considera que el análisis del recurso procesador ($proc_u$) y de la utilización de la memoria (mem_u) es fundamental para que el algoritmo de balanceo de carga pueda determinar el estado actual de cada instancia en cada momento de la ejecución en que sea requerido. Se consideró que el tráfico en la red no resulta representativo, ni permite al algoritmo de balanceo de carga realizar una adecuada selección de instancia, ya que todos los nodos se encuentran dentro de la misma red, por lo que el tráfico entrante es homogéneo para todas las instancias. Se puede dividir la implementación del algoritmo en dos grandes secciones, la determinación de la carga de trabajo actual en cada nodo y el algoritmo que realiza la asignación de instancia para cada pedido [54].

Obtención de métricas Para lograr estimar la carga de trabajo de un nodo es necesario recolectar y analizar métricas de consumo de recursos que

se consideren representativas. Como ya se mencionó, en este caso se optó por estudiar el consumo del recurso procesador y la utilización de la memoria. Para la obtención de dichas métricas se utiliza una herramienta provista por Microsoft Azure que se denomina *Diagnostics*. La configuración de esta herramienta se describe con detalle en la Sección 5.5.3. Cabe destacar que para el correcto funcionamiento y convergencia del algoritmo se requiere de un consumo de espacio de almacenamiento extra para guardar las métricas en cuestión. De todas formas, dada la mejora de desempeño que produce el algoritmo de balanceo de carga dinámico, aún con este consumo adicional, el algoritmo continúa resultando beneficio para el sistema.

Algoritmo El algoritmo propuesto utiliza un período de diez minutos para calcular el promedio de las métricas obtenidas. Para obtener este valor en cada recurso y cada instancia de rol se aplica la ecuación

$$c_n(par) = \frac{m_1 + m_2 + ... + m_t}{t} \tag{5.1}$$

En la ecuación 5.1, c_n representa la carga promedio para la métrica especificada, en una instancia dada, durante los últimos t segundos. *par* es el parámetro de carga que se quiere medir, puede tomar el valor de $proc_u$ o mem_u. $m_1 \dots m_t$ es el valor para un parámetro dado en algún instante dentro del intervalo de tiempo. t representa el intervalo de tiempo, en el caso de este proyecto está configurado con el valor de diez minutos. n es el identificador de cada una de las instancias de rol.

El algoritmo de balanceo de carga utiliza dichos promedios para determinar el nivel de carga de cada instancia. En base a esto se toma la decisión de determinar a qué instancia será dirigido cada pedido entrante. Como primer paso se realiza una clasificación por nivel de carga y se asigna cada instancia a unas de las cuatro siguientes categorías: ociosa, baja, normal y alta. Dentro del primer paso, se utilizan dos umbrales para medir el consumo general del procesador. Estos dos umbrales permiten obtener un algoritmo de balanceo de carga más estable, ya que evita que una instancia cambie constantemente su clasificación de carga alta a baja y viceversa. Para realizar el cálculo de estos umbrales primero debe calcularse la carga promedio de consumo de procesador sobre todas las instancias. Para ello se utiliza la ecuación $C_{prom}(CPU) = \frac{c_1+c_2+...+c_n}{n}$, en la que $C_{prom}(CPU)$ es la carga promedio del recurso procesador sobre todas las instancias, $c_1 \dots c_n$ representan la carga promedio para cada instancia calculadas según la ecuación 5.1 y n es el identificador de cada instancia.

Determinación de umbrales Para calcular los umbrales alto y bajo del consumo de procesador se debe multiplicar el promedio general explicado en el párrafo anterior por una constante.

$$U_a = C_{prom}(CPU) \times A \tag{5.2}$$

$$U_b = C_{prom}(CPU) \times B \tag{5.3}$$

En las ecuaciones 5.2 y 5.3, U_a y U_b representan los umbrales para el consumo de procesador alto y bajo respectivamente y A y B son constantes. A debe ser mayor a uno y B menor que uno. En el algoritmo desarrollado, A toma el valor de 1.3 y B de 0.7 como se sugiere en el artículo referenciado en [54]. Esto significa que cuando el consumo de procesador en una instancia se encuentra 30 % por encima del promedio general, esta instancia se considera como altamente cargada en cuanto a este recurso. Si por el contrario el consumo del procesador en la instancia es del 70 % o menor, se considera que la instancia está poco cargada. Si no se cumple ninguna de estas dos condiciones se considera que la instancia está realizando un consumo de procesador normal. Cabe destacar que esta clasificación es para el uso de procesador y no para la instancia en general. También se debe tener en cuenta el consumo de memoria, pero en este caso no se utilizaron umbrales, sino que directamente se emplea el promedio de diez minutos calculado en la ecuación 5.1.

Una vez determinados los umbrales alto y bajo, se procede a clasificar la instancia en una de las cuatro clases de categorías. Usando los umbrales para el consumo de procesador y el promedio de utilización de memoria se categoriza el nivel de carga de la instancia en ociosa, baja, normal y alta, según corresponda, de acuerdo a las siguientes condiciones.

- **Ociosa** $proc_u \leq 30\,\%$. En cualquier servidor existen tareas que se encuentran ejecutando de forma permanente o por cortos períodos de tiempo (como ser servicios de monitoreo, rutinas propias del sistema operativo y procesos del *Cloud Service*). Dado este hecho, se utiliza un 30 % como margen para contemplar la ejecución de este tipo de tareas y concluir que la instancia no está ejecutando ningún pedido. A raíz de este razonamiento es que se utiliza el consumo de procesador para determinar si la instancia se encuentra ociosa o no.
- **Baja** $proc_u \leq U_b$ and $mem_u < 85\,\%$. Se clasificará una instancia con carga baja si se cumplen simultáneamente las siguientes dos condiciones.

    1. El consumo de procesador está por debajo del umbral bajo (U_b).
    2. La utilización de memoria no alcanza el 85 % de consumo. Se considera que si una instancia tiene menos del 15 % de memoria disponible corre el riesgo de comenzar un sistema de paginado, lo cual disminuiría la velocidad de procesamiento de forma dramática. Es por ello que si no se alcanza este nivel de consumo no habrá riesgo de paginado, por lo que no se verá afectada la velocidad de procesamiento.

- **Normal** $mem_u < 85\,\%$ and $proc_u < U_a$. La instancia está considerada dentro de un estado normal si la utilización de memoria no alcanza el nivel de riesgo mencionado en el punto anterior y además el consumo de procesador no supera el umbral alto (U_a).
- **Alta** Si la instancia no cumple con ninguna de las clasificaciones anteriores será clasificada como de alta carga.

Por último se procede a la decisión de selección de instancia. El algoritmo de balanceo de carga, fue implementado teniendo en cuenta las métricas que se utilizan para el escalado automático del sistema. Esto permite asumir que siempre existirá al menos una instancia cuya carga no sea alta, es decir, que se encuentra categorizada como ociosa, con baja carga o con carga normal. Es por ello que se descartan automáticamente las instancias cuya carga es alta.

Algorithm 3 Pseudo-código de selección de instancia en algoritmo de balance de carga

```
1: if hay instancias ociosas then
2:     Enviar pedido a instancia ociosa
3: else if hay instancias con carga baja then
4:     Enviar pedido a instancia con baja carga
5: else
6:     Enviar pedido a instancia con carga normal
7: end if
```

Como se explica en el pseudo-código descrito en el algoritmo 3, el balanceador de carga brinda preferencia a aquellas instancias ociosas. Si se encuentra alguna instancia en estado ocioso, el pedido entrante será dirigido a la misma. En caso de que no encuentre instancias ociosas, el algoritmo verificará si existe alguna cuya carga sea baja y de ser así enviará el pedido a una de estas. Por último, si no hay instancias ociosas, ni tampoco instancias con baja carga se procede a enviar el pedido a una instancia con carga normal. Esto es posible si se tiene en cuenta la afirmación que establece que siempre debe existir al menos una instancia cuya carga no sea alta.

5.13. Tolerancia a fallos

En el proyecto se implementó una aplicación distribuida en el cloud en comunicación con un clúster HDInsight. En un desarrollo de este tipo, la implementación de mecanismos para mitigar y recuperarse ante fallos se vuelve fundamental, ya que es imprescindible para proporcionar al usuario un servicio confiable, disponible y que proporcione información legible y clara sobre errores. Como se explicó en el capítulo 4, la aplicación provee varios mecanismos de tolerancia a fallos, algunos provistos por el *framework* o por

herramientas utilizadas para el desarrollo como WCF y otros implementados específicamente.

Uno de los principales aspectos del proyecto es el clúster HDInsight, que ejecuta todas las simulaciones de la aplicación y es de donde se obtienen las salidas que se le proporcionan al usuario. En las tareas que ejecuta el algoritmo de MapReduce pueden ocurrir fallos esporádicos, ya sea por fallos en instancias del clúster o por problemas internos, pero estos fallos son manejados y resueltos por el mismo *framework* Azure HDInsight. Sin embargo, los errores en los *mappers* también podrían deberse a una configuración errónea en los parámetros ingresados por el usuario. Para el caso de MCell, el *mapper* que lo ejecuta se encarga de obtener la salida que se produce en consola durante la ejecución, luego, en caso de que el algoritmo no pueda finalizar de forma exitosa, el usuario podrá descargar la salida y así comprender el motivo del fallo. En el caso de FERnet, también se almacena la salida en consola, pero la aplicación no permite descargarla. Esta decisión se tomó considerando que en la aplicación se puede configurar un único parámetro para la ejecución de FERnet y el tiempo y la complejidad de cómputo de FERnet son mucho menores en comparación a MCell.

5.14. Seguridad y Certificados

En el proyecto se utilizan certificados para la conexión segura con el clúster HDInsight. Un certificado de administración de Azure es un certificado *X.509* utilizado para autenticar un agente, como *Visual Studio Tools* para Windows Azure o una aplicación cliente que utilice la API de administración de servicios, y que actúa en nombre del propietario de la suscripción para administrar los recursos de ésta. Los certificados de administración se cargan en Azure y se almacenan en el nivel de suscripción. El almacén de certificados de administración puede contener hasta cien certificados por suscripción. Estos certificados se utilizan para autenticar la implementación de Windows Azure.

Los certificados de administración deben tener una longitud de clave de 2.048 bits como mínimo, y deben residir en el almacén personal de certificados. Cuando el certificado se instala en el cliente, debe incluir la clave privada del mismo. Para cargarlo en el portal de administración de Windows Azure es necesario exportarlo como un archivo con formato .cer que no contiene la clave privada [4].

En el caso del proyecto fue necesario obtener la huella digital (o *thumbprint*) para un certificado. La huella digital es utilizada para identificar el certificado para autenticar operaciones realizadas en un servicio en el cloud. En el caso del proyecto, mediante la huella digital es posible obtener el certificado *X.509* que permite la conexión con el clúster HDInsight, y a partir de ese momento es posible la realización de acciones sobre el mismo, como subir

y ejecutar tareas MapReduce. Para obtener la huella digital de un certificado se accede al administrador de certificados del sistema (local), ingresando en el cuadro de texto del menú inicio de Windows el comando «certmgr.msc». Desde el administrador, se importa el certificado y luego se abre haciendo clic derecho sobre el mismo; luego se accede a la sección «Detalles» y en la misma se puede obtener la propiedad huella digital deseada [5].

5.14.1. Certificados X.509 para acceder a HDInsight

Para que el rol de trabajo *Job Creation* pueda acceder al servicio HDInsight brindado por Azure, es necesario configurar los certificados correspondientes. Primero se debe obtener un certificado SSL que se puede descargar desde la plataforma Azure utilizando la herramienta Powershell, mediante el comando «Get- AzurePublishSettingsFile». Este comando descarga en el sistema dos archivos con extensiones .cer y .pfx respectivamente. Otra opción es crear un certificado auto-firmado en la máquina local utilizando IIS. Para crear el certificado auto-firmado se deben realizar los siguientes pasos utilizando Windows (en inglés):

1. Start → Run → escribir «inetmgr» y Ok (Este paso abre el administrador de IIS) → Seleccionar el nodo raíz en el panel de la izquierda → doble clic en «Server Certificate» → «Create Self–Signed Certificate» en el extremo derecho del panel lateral → Completar el formulario y obtener el certificado auto-firmado
2. Para generar el archivo .pfx del certificado: en el administrador IIS, doble clic en el certificado creado → En la ventana de propiedades del certificado, ir a la pestaña de detalles → Clic en «Copy to File» → Next → Seleccionar «Yes, export the Private key» (exportar clave privada) → Next → Seleccionar «Personal Information Exchange.» → Next → Ingresar una contraseña → Next → Elegir la ubicación del archivo → Next → Finish
3. Para generar el archivo .cer del certificado: se realizan los mismos pasos que en el punto anterior pero esta vez seleccionando el punto «Export without private key».

El paso siguiente es subir al portal de Azure el certificado creado. Para ello hay que tener en cuenta ciertos puntos:

- El archivo .cer creado se debe subir en la sección «managment certificates» del portal de Azure. Se accede a la misma en el menú izquierdo del portal bajo el nombre de «settings».
- El archivo .pfx creado debe ser subido en la sección «certificates» en la plataforma de administración del *Cloud Service* en el portal de Azure. Una vez copiado el archivo se puede obtener el valor de la huella digital

que debe ser utilizado por el rol de trabajo para obtener el certificado al momento de llamar un servicio Azure (HDInsight en este caso).

Luego se debe configurar el rol de trabajo para que utilice el certificado creado, para ello se deben abrir las propiedades del rol en el *Cloud Service* y bajo la pestaña de *Certificates* se debe agregar una nueva entrada donde se especifica el nombre amigable (*friendly name*) y huella digital del certificado obtenido en el portal de Azure en el paso anterior. Finalmente, desde el código del rol de trabajo se obtiene el certificado utilizando como filtro la huella digital y se llama al servicio requerido utilizando el mismo [28].

5.15. Despliegue en el cloud

Con la creación de un *Cloud Service*, Azure ofrece dos entornos de despliegue de servicio en el cloud: un entorno de prueba y ensayo (*staging*) en el que el desarrollador puede probar su implementación antes de liberar el producto al entorno de producción y el ambiente de producción que contiene la aplicación final que será utilizada por los usuarios. Los dos ambientes se distinguen sólo por las direcciones IP virtuales (VIP) a través de las cuales se accede al servicio en el cloud. El entorno de prueba es identificado a través del identificador único y global del servicio en el cloud (GUID) que se encuentra en las URLs de la forma «GUID.cloudapp.net». En el entorno de producción, la URL está basada en el prefijo «DNS amigable» asignado al servicio en el cloud (por ejemplo, myservice.cloudapp.net).

Para intercambiar una implementación desde el entorno de prueba Azure al entorno de producción, se realiza una conmutación de los VIPs a través de las cuales se accede a las dos implementaciones. Después del intercambio, el nombre DNS para el ambiente de producción apunta a la dirección que solía representar el ambiente de prueba.

Visual Studio 2012 junto con el SDK de .Net para Azure permiten una integración directa y sencilla de la aplicación desarrollada de forma local a los ambientes de ensayo y producción. Una vez que se asocia el *Cloud Service* desarrollado en Visual Studio con una suscripción de Azure, es posible realizar el despliegue en la nube. Para ello hay dos opciones, la primera es crear un paquete de configuración de servicio conteniendo el código de aplicación y la configuración. Este paquete es utilizado desde el portal de administración de Azure para realizar el despliegue. La segunda opción consiste en seleccionar la opción «Publicar» desde el menú de atajos del proyecto web y tras seleccionar la configuración de servicio y el ambiente de despliegue comenzarán a realizarse las acciones necesarias para desplegar el sistema en el cloud. De todas formas, para el correcto funcionamiento de la aplicación en todos los entornos es necesario contar con una correcta configuración de los servicios de comunicación, de los accesos al espacio central de almacena-

miento y del acceso al clúster a través de certificados como se explicó en la sección 5.14.

Una vez desplegada la aplicación en el cloud, es posible acceder a través del portal web de administración de Azure a cada entorno de ejecución y realizar diversas acciones. Entre las más utilizadas en el proyecto se destacan la obtención de la URL para acceder al sitio, el monitoreo de diferentes métricas de uso, como ser el consumo de cada instancia de cada rol o resumen de utilización, y la configuración del sistema de autoescalado.

Capítulo 6

Análisis experimental

El objetivo de este capítulo consiste en la descripción detallada y análisis de las pruebas que se realizaron durante el desarrollo del proyecto, explicando en cada caso los métodos y herramientas empleados.

6.1. Plataforma de ejecución

A lo largo de este capítulo se plantean pruebas realizadas sobre la plataforma Azure con las tecnologías definidas en el capítulo 4. Los análisis experimentales se ejecutaron en un *Cloud Service* desplegado en el cloud conformado por 4 instancias de máquinas virtuales de tamaño pequeño (*small*, CPU con 1 núcleo, 1,75 GB de RAM), 2 para el Web Role y 2 para el rol *Interrole Commons*, 4 instancias de máquinas virtuales de tamaño mediano (*medium*, 2 núcleos y 3,5GB de RAM), 2 para el rol de trabajo *Message Processing* y 2 para el rol de trabajo *Job Creation*. Para los casos de prueba que no utilizan esta configuración, se especifican los tamaños y número de instancias utilizados en cada caso.

Por otro lado, el clúster HDInsight utilizado para las pruebas está conformado por 2 nodos maestros (*head nodes*) y 8 nodos esclavos, donde cada uno de estos nodos se instancia mediante una máquina virtual de clase A3 que cuenta con una CPU de 2.20GHz con 4 núcleos y 7GB de memoria RAM. Como sistema de archivos HDFS se utilizó un contenedor de *blobs* con una capacidad máxima de 500TB.

6.2. Análisis experimental de las simulaciones

La principal funcionalidad de la aplicación es la ejecución de simulaciones utilizando MCell y FERnet en un clúster HDInsight. Por este motivo, la ejecución de tareas en un clúster HDInsight fue lo primero en ser desarrollado y testeado.

Desde el portal de administración de Azure se creó un clúster HDInsight con las características especificadas sobre el que se ejecutaron las primeras pruebas. Inicialmente se hizo uso de la herramienta Azure Powershell, utilizada para aprender las configuraciones y manejo básico de tareas en el clúster. Se ejecutaron varias tareas de ejemplo brindadas por Azure, como ser *Contador de palabras* o *Estimador del número pi*. Azure proporciona la implementación de los algoritmos de ejemplo que utilizan Hadoop, y provee de todos los archivos necesarios para la ejecución almacenados en forma de *blobs* en la cuenta de almacenamiento de Azure. Luego, desde la consola de comandos de Azure Powershell es posible lanzar dichas ejecuciones.

Una vez que se pusieron en funcionamiento las tareas de ejemplo con cada parámetro y configuración, se procedió a ejecutar la simulación. Para la ejecución de MCell se utilizó Azure Powershell en forma de una tarea en *streaming*, que recibe como entrada el archivo .exe ejecutable de MCell. Sin embargo, como ya se describió en el capítulo 5, este mecanismo de ejecución debió ser descartado por la incapacidad de proporcionar los parámetros necesarios y en el formato esperado para ser ejecutados por *MCell.exe* y *FERnet.exe*.

Al no poder utilizar los archivos ejecutables de forma directa, se debió implementar una tarea en Java que utilice MapReduce de Hadoop para ejecutar los programas MCell y FERnet, cuya implementación se describió en el capítulo 5. Para llevar a cabo la implementación utilizando Java, se construyó el proyecto de forma progresiva y modular, siguiendo los pasos que se describen a continuación.

1. Se implementó un proyecto donde FERnet era ejecutado en un (y único) *mapper* y no contaba con una implementación de *reducer*. Se optó por probar FERnet en primera instancia ya que su ejecución es más simple e insume menor tiempo de cómputo.
2. Se ejecutó MCell localmente desde consola y se obtuvo la salida de dicha ejecución para utilizarla como entrada de FERnet en el clúster HDInsight.
3. Se subieron a la cuenta de almacenamiento de Azure asociada al clúster el ejecutable de FERnet, las bibliotecas requeridas, la salida de MCell con extensión .dat y el .jar conteniendo la implementación del algoritmo que utiliza MapReduce para la ejecución de FERnet.
4. Desde Azure Powershell se realizó la ejecución de la tarea MapReduce, logrando el correcto funcionamiento de la tarea y almacenando en forma de *blob* el resultado de la ejecución.

De forma análoga a los puntos descritos, se creó un algoritmo MapReduce en Java, donde el método *mapper* se encarga de ejecutar MCell, y de igual manera se subieron los archivos al clúster y se probó su correcto funcionamiento utilizando Azure Powershell. Para la ejecución de dichas simu-

laciones de prueba se utilizó uno de los ejemplos provisto junto con el código de ambos componentes de software. La entrada de dicho ejemplo se compone de cinco archivos .mdl: «*Scene.geometry.mdl*», «*Scene.initialization.mdl*», «*Scene.main.mdl*», «*Scene.molecules.mdl*» y «*Scene.viz_output.mdl*» que por simpleza fueron fusionados en un único archivo «*Scene.main.mdl*». El cuadro 6.1 resume los parámetros de entrada de MCell utilizados y de relevancia para el proyecto, en el caso de FERnet se utilizó el modo de ejecución punto (*point*).

Cuadro 6.1: Parámetros de entrada

Parámetro	**Valor**
ITERATIONS	5000
TIME_STEP	$1E-05$
COORDENADAS	$[-1.5,-1.5,-1.5]$ y $[1.5,1.5,1.5]$
ASPECT_RATIO	-
MOLECULAS	A (constante de difusión = $5.5E-07$) y B (constante de difusión= $3E-07$)
REACCIONES	Sin reacciones

Una vez que se pusieron ambos proyectos en funcionamiento, se procedió a testear la integración del clúster HDInsight desde el código en C#. Para ello se creó un proyecto de tipo «Cloud» en Visual Studio, el cual tiene como propósito la ejecución de una simulación en el clúster que recibe la salida de MCell «*joined.dat*» y ejecuta la simulación de FERnet explicada en el punto anterior. Luego de implementado el código para la conexión (primero utilizando HCatalog y luego el mecanismo de conexión intrínseco de Azure) se realizaron pruebas de integración para verificar la ejecución de la simulación en el cloud. Para ello se ejecutó el programa y se verificaron los resultados a través de la cuenta de almacenamiento de Azure. Adicionalmente, se realizó una prueba similar utilizando el mismo proyecto, pero en este caso se utilizó como entrada el archivo .mdl y se lanzó en el clúster una ejecución de MCell.

Una vez que la conexión desde el código con el clúster se estableció de forma exitosa se procedió a integrar ambas tareas Hadoop en una sola. En este caso la integración se probó directamente en el programa desarrollado en Visual Studio. Nuevamente se probó la ejecución de la simulación en el cloud, verificando los resultados en la cuenta de almacenamiento de Azure. La última etapa de integración consistió en incorporar a la aplicación web el proyecto dedicado a la conexión y lanzamiento de tareas en el clúster. Para ello se agregó a la implementación del *Cloud Service* un proyecto de tipo rol de trabajo conteniendo el código a integrar (*Job Creation*), ya que posteriormente se adicionó el lanzamiento de simulaciones para los distintos archivos .mdl entrantes. Para la evaluación experimental de la simulación se

utilizaron dos casos de pruebas. El primero de ellos utilizando la ejecución básica ya descrita, el segundo utilizando una ejecución intensiva en cómputo que se describe en la sección 6.6.

6.3. Métodos de depuración

Durante el desarrollo de la aplicación se emplearon diversas técnicas de depuración. Para el testeo de la aplicación web, se empleó la herramienta de depuración (o *debugger*) propia de Visual Studio, que es muy completa y permite un manejo de flujo de ejecución muy práctico. Al ejecutar la aplicación web de forma local se utiliza un emulador de instancias que representa el entorno de cada rol de forma virtual en el ambiente local de ejecución y se utiliza la verdadera cuenta de almacenamiento y el clúster HDInsight desplegado en la nube.

La aplicación alojada en la nube, además de hacer uso de la cuenta de almacenamiento y el clúster, ejecuta instancias reales (aunque continúan siendo virtualizaciones) de cada rol. En algunas ocasiones fue necesaria la depuración del ambiente desplegado en la nube, ante la ocurrencia de errores que no sucedían localmente. En este caso el ambiente de Visual Studio permite ser configurado para ejecutar desde el propio entorno la aplicación real en la nube y de esta forma se puede depurar el entorno al igual que se haría de forma local; la diferencia radica en el tiempo requerido para realizar la depuración, el cual puede llegar a ser muy elevado.

Para la depuración y monitoreo del algoritmo de simulación ejecutado en el clúster HDInsight se utilizan los *blobs* de datos y metadatos generados durante la ejecución. Además, el clúster cuenta con un mecanismo de conexión remota que permite acceder mediante la herramienta de «Conexión a escritorio remoto» de Windows al *head node* del clúster. Para ello se accede a la configuración del clúster desde el portal de administración de Azure, se solicita una ejecución remota para la cual es necesario configurar un usuario y contraseña que tiene un cierto período de caducidad; y con las credenciales creadas y la herramienta antes mencionada es posible acceder remotamente al *head node*. La configuración de dicho usuario y contraseña tiene una validez de una semana para prevenir fallas de seguridad, por lo que después de dicho período es necesario repetir la operación para crear un nuevo usuario para el acceso remoto. Con la ejecución de cada simulación en el clúster se genera un *blob* de nombre «*sciclúster/test/status–115.0/stderr*» que contiene mensajes de información, advertencias y errores relacionados a la ejecución y su URL correspondiente. Al acceder al *head node* es posible introducir la URL de la tarea en ejecución en un navegador local y acceder a los *logs* e información general de cada *mapper* y *reducer*, para consultar el progreso o la salida producida, por ejemplo.

6.4. Desempeño del balanceador de carga

Con el objetivo de evaluar el rendimiento del balanceador de carga se realizaron pruebas en el ambiente de producción. La idea fue sobrecargar la primera de ellas con una ráfaga de simulaciones obligando al rol de trabajo Job Creation a monitorear las simulaciones y consumir recursos, logrando así que el balanceador de carga tenga que elegir la segunda instancia para ejecutar una nueva simulación. La ejecución de simulaciones se realizó de forma manual. Se ejecutaron cerca de 45 simulaciones separadas en un intervalo de tiempo suficiente para que se generen métricas de uso de recursos y así sobrecargar al rol, pero solo se consiguió aumentar el uso de CPU de 50 % a 56 %, no entrando en la categoría de *Alta* brindada por el balanceador.

En las 45 simulaciones realizadas se encontraron tiempos variables de ejecución del algoritmo de balanceo de carga. El tiempo promedio con esta configuración de ejecución se encuentra en los 572ms, siendo el mínimo 76ms y el máximo 998ms. Es decir, el resultado del análisis del algoritmo de balanceo de carga muestra que no genera un *overhead* considerable si se evalúa el tiempo total de simulación que varía entre varios minutos y horas.

6.5. Análisis de simulaciones de barrido paramétrico

Para estudiar el desempeño del barrido paramétrico se realizaron pruebas en un ambiente local (CPU con 2 núcleos de 3 GHz, 12GB RAM) ejecutando diferentes combinaciones de reacciones. Se observaron los resultados obtenidos y se evaluaron los tiempos que se consumen al armar las distintas combinaciones de parámetros.

Se realizaron varias pruebas para verificar el correcto funcionamiento del barrido paramétrico creando diversas reacciones, definiendo rangos de *rate* para cada una de ellas y observando el resultado obtenido por el algoritmo. Se presenta en la tabla 6.2 las reacciones definidas y los rangos de rate para cada una de ellas.

Cuadro 6.2: Definición de reacciones

Reacción	**Rate inicial**	**Rate final**	**Paso**
$A + B -> A$	$1.1e-6$	$1.2e-6$	$0.1e-6$
$B + B -> A$	0.3	0.4	0.1
$A + A -> B$	0.06	0.08	0.01

Como se puede apreciar en la tabla 6.3, el algoritmo genera las distintas combinaciones de rates para las reacciones dadas.

Cuadro 6.3: Resultados barrido paramétrico para valores de tabla 6.2.

Combinación	**Rate reacción 1**	**Rate reacción 2**	**Rate reacción 3**
1	$1.1e-6$	0.3	0.06
2	$1.1e-6$	0.3	0.07
3	$1.1e-6$	0.3	0.08
4	$1.1e-6$	0.4	0.06
5	$1.1e-6$	0.4	0.07
6	$1.1e-6$	0.4	0.08
7	$1.2e-6$	0.3	0.06
8	$1.2e-6$	0.3	0.07
9	$1.2e-6$	0.3	0.08
10	$1.2e-6$	0.4	0.06
11	$1.2e-6$	0.4	0.07
12	$1.2e-6$	0.4	0.08

Para estudiar los tiempos de ejecución del algoritmo de barrido paramétrico se toman diferentes vectores de rangos de *rate*, variando los índices de inicio y fin de manera de aumentar la cantidad de posibles combinaciones.

Cuadro 6.4: Tiempos de ejecución del barrido paramétrico (en milisegundos)

# reacciones/ # combinaciones	**4**	**9**	**90**	**200**	**10000**
2	102	120	720	14400	49380
6	–	72	660	12000	62400
10	–	–	660	1020	61200
20	–	–	900	1020	57000

Los resultados expuestos en la tabla 6.4 muestran que el factor que altera más significativamente los tiempos de realización del barrido es el número de combinaciones posibles de parámetros y no el número de reacciones, ya que los tiempos se mantienen aproximadamente iguales al variar la cantidad de reacciones. Al momento de realizar la prueba se pudo apreciar que el algoritmo es más sensible a la cantidad de posibles valores dentro de una reacción que a la cantidad de reacciones, es decir, una simulación demorará mas si hay una reacción que posee 1000 posibles combinaciones que una combinación de varias reacciones que se interpretan en 1000 combinaciones de parámetros posibles.

Por otro lado, se puede apreciar en los resultados que los tiempos de ejecución del algoritmo no generan un gran *overhead* en relación al tiempo total entre que el usuario lanza la simulación hasta que obtiene los resultados, ya que en el caso de generar 10000 combinaciones (caso poco probable), se genera un *overhead* de 1 minuto aproximadamente.

6.6. Análisis experimental de una gran simulación

Dada las características del sistema implementado y el propósito de poder ejecutar grandes simulaciones, se realizó una prueba donde se ejecutó una única simulación de gran porte con un millón de iteraciones de MCell. Los parámetros utilizados se presentan en la tabla 6.5. Este caso de prueba se ejecutó en un ambiente local y se pudo observar que el archivo generado por MCell y correspondiente entrada de FERnet es de un tamaño de más de 10GB, por lo que fue de sumo interés comprobar el comportamiento de su ejecución en el cloud.

Cuadro 6.5: Parámetros de entrada

Parámetro	**Valor**
ITERATIONS	$1E06$
TIME_STEP	$1E-05$
COORDENADAS	$[-1.5,-1.5,-1.5]$ y $[1.5,1.5,1.5]$
ASPECT_RATIO	-
MOLECULAS	A {Constante de difusión $= 5.5E-07$} y B {Constante de difusión= $3E-07$}
REACCIONES	Sin reacciones
NAME_LIST	A B

El primer resultado obtenido no fue el esperado, ya que la tarea *mapper* encargada de ejecutar MCell y enviar al segundo *mapper* el archivo de salida generado lanzó una excepción de exceso de tamaño de archivo en memoria. La estrategia utilizada al momento era leer todo el archivo de salida de MCell y enviarlo a la salida estándar de las tareas *mappers* en Hadoop. Este enfoque es erróneo, ya que no se tiene en cuenta la necesidad de mantener todo el archivo en memoria para poder enviarlo a través de la salida estándar, ocasionando problemas de memoria al manejar archivos más grandes.

Para resolver el problema se decidió utilizar otra estrategia, en la cual se almacena el archivo de salida en un *blob* en el contenedor asociado al clúster HDInsight, de manera que todas las tareas que componen el trabajo MapReduce puedan acceder a él. Por tanto, se realiza una lectura del archivo de salida de MCell almacenado localmente al nodo, utilizando un *buffer* y copiando su contenido al *blob*. Así, se evita generar un exceso de memoria utilizada por la aplicación. Luego, en vez de enviar los bytes del archivo al siguiente *mapper*, se le envía el nombre del *blob* que debe obtener de la cuenta de almacenamiento.

Se ejecutó tres veces la misma simulación y se obtuvo un tiempo promedio de ejecución total de 5 horas y 25 minutos. La misma ejecución en el ambiente local tomó 1 hora y 23 minutos. El tiempo obtenido con la eje-

cución en el cloud es relativamente mayor que la ejecución en un ambiente local, ya que el archivo creado por la ejecución de MCell, con un tamaño de 13.94GB, es leído dos veces en el flujo. En primera instancia el archivo es leído en el *mapper* de MCell para almacenarlo en el *blob*, y en segunda instancia es leído por el segundo *mapper* para copiarlo al ambiente local y realizar la ejecución de FERnet.

El *overhead* generado por la lectura de datos ocasiona un tiempo de ejecución mayor para grandes simulaciones. Sin embargo, este *overhead* se contrarresta con el tiempo ahorrado al realizar la ejecución de una gran cantidad de simulaciones en paralelo en comparación a la ejecución serial de las mismas en un ambiente local; al menos en lo que respecta a un gran número de simulaciones y contando con los recursos suficientes. Si se desea ejecutar la misma simulación descrita en la tabla 6.5 10 veces, en el ambiente local de manera serial demoraría casi 14 horas, mientras que en el ambiente cloud seguirá demorando 5 horas y 25 minutos promedio, ya que se ejecutan las 10 simulaciones en paralelo.

Los resultados muestran que es posible liberar el entorno local del usuario utilizando el portal desarrollado en el proyecto para ejecutar la simulación en el cloud, aún cuando se trata de simulaciones de gran porte. Esto es de vital relevancia ya que este tipo de simulaciones son las que insumen mayor tiempo de cómputo. Además, si bien la simulación en el cloud requiere mayor tiempo que la simulación local debido a la lectura de archivos de gran tamaño, se logra compensar este tiempo mediante la paralelización de las simulaciones.

6.7. Escalado de la infraestructura

Al completar la implementación del *Cloud Service* y cuando se desplegó el servicio en el cloud, los roles *Message Processing* y *Job Creation* tenían asignados máquinas virtuales del tipo A1. Dichas máquinas virtuales poseen una CPU de un núcleo y 1.75BG de memoria RAM. Este factor es importante ya que el rol por el simple hecho de estar activo y esperando posibles llamados consume recursos de CPU. La CPU brindada por esta máquina virtual no era suficiente para mantener al rol activo y a su vez procesar pedidos del usuario y es por ello que se pudo observar al desplegar el *Cloud Service* en el cloud que unos instantes después de encontrarse en estado estable, el *framework* de Azure comenzaba a iniciar una nueva instancia para el rol ya que su uso de CPU excedía el porcentaje definido en la configuración del escalado automático.

Se procedió a aumentar el tamaño de las máquinas virtuales a A3 (CPU con 4 núcleos y 7GB de RAM), para las cuales se observó un decremento en el uso de CPU en estado de espera del rol (sin pedidos de usuario) a un 25 % promedio. Para cumplir con los requisitos expuestos en el SLA sobre la disponibilidad de los roles en línea, se crearon dos instancias de cada rol en el

Cloud Service. Al tiempo de haber desplegado esta nueva configuración en el cloud, el sistema contaba con tan solo una instancia de cada rol; este hecho se debió al bajo uso de CPU y la configuración de escalado definida, la cual apaga instancias de roles hasta dejar tan solo una si las mismas se encuentran utilizando un bajo porcentaje de CPU. Luego de observar este hecho se modificó la regla de escalado automático para que el mínimo de instancias de un rol en el *Cloud Service* sea dos. Utilizando como métrica el uso de CPU y utilizando el portal de administración de Azure para monitorear la carga en cada rol, fue posible observar el comportamiento configurado del autoescalado.

6.8. Estudio de la tolerancia a fallos

Para las pruebas de tolerancia a fallos se buscó crear fallas en los diferentes roles y observar el comportamiento de la aplicación.

Inicialmente existen dos instancias de cada rol ejecutando en el *Cloud Service*; Azure garantiza un 99,95 % de accesibilidad, ya que en caso de falla mientras una instancia se reinicia todo el flujo se redirige a la segunda instancia, y en caso de sobrecarga del procesador se inicia una nueva instancia por la configuración de autoescalado del *Cloud Service*. Aún así se decidió experimentar la reacción del sistema frente a una falla general de cada rol. Para simular esta falla se desplegó en el cloud toda la infraestructura del *Cloud Service* y se accedió desde un navegador al portal web que permite ingresar parámetros y ejecutar las simulaciones. Los cuatro casos de fallo estudiados se describen a continuación.

6.8.1. Falla del rol Message Proccesing

Para generar la falla se dejó temporalmente el rol *Message Proccesing* deshabilitado, forzando el reinicio de las máquinas virtuales para ambas instancias. En este estado, se envió una simulación desde el sitio web. En primera instancia se recibió un mensaje de "'Error interno del servidor', ya que ninguna de las dos instancias se encontraba disponible para atender el pedido. Inmediatamente después de cerrar el mensaje de error, se Al re-lanzó la tarea y la aplicación devolvió un identificador de usuario y continuó ejecutando de manera normal. Las instancias se reinician en pocos segundos; es por esta razón que el sistema se recupera rápidamente y continúa su ejecución. Ya que las instancias no mantienen estado, no es necesario realizar ninguna tarea una vez que finaliza de reiniciar, sólo exponer el servicio que atiende pedidos del usuario. Cabe destacar que la probabilidad que ambas instancias sean reiniciadas es muy pequeña, dado el contrato de servicio de Azure.

6.8.2. Falla del rol Job Creation

Para estudiar los fallos del rol *Job Creation* se procedió de manera análoga al caso descrito para el rol *Message Processing*. Al lanzar la ejecución el sistema devolvió un ID para corroborar el estado de tareas y resultados. Al verificar los resultados con el ID, se encontró que el sistema devuelve todas las tareas en estado "Ejecutándose". Aunque en el momento exacto del lanzamiento de las simulaciones el rol Job Creation no se encontraba disponible, el rol *Message Processing* consiguió comunicarse con el mismo en un período corto de tiempo.

Los resultados muestran que el periodo de tiempo que tardan las instancias en ser reiniciadas en la plataforma Azure es muy breve. Por esta razón la comunicación fue posible, por más que la simulación se lanzó apenas un segundo después de reiniciar las instancias en el caso de Job Creation, y que el segundo intento de ejecución de simulación fue exitoso en el caso de la falla en el rol *Message Processing*.

Otra posible falla del rol se puede dar cuando está verificando el estado de ejecución de la tarea y actualizando la variable de estado utilizada como metadato *"pruebaProgreso-IDsimulacion.IDjob"*. Se realizó una prueba local en la cual se lanzó una simulación y se obtuvo el estado de las tareas hasta que todas se encontraron en estado "Ejecutándose". Al llegar a ese estado se mató al proceso local de ejecución del *Cloud Service*, por lo que el metadato quedó para siempre con el valor "Ejecutándose". Al corroborar los resultados se encontró que el sistema devolvía siempre el mismo estado por más que se pudo observar en el Azure Storage Explorer que la tarea había finalizado con éxito. Se procedió por tanto a verificar la finalización de una tarea no solo por el metadato especificado, sino también corroborando tanto la existencia de archivos de salida como archivos utilizados para indicar errores en la ejecución, como se explicó en el capítulo 5. Una vez realizado el cambio en la implementación, se procedió a realizar los mismos pasos recién descritos. El resultado de esta ejecución fue satisfactorio, ya que el sistema devolvió los resultados una vez finalizadas las ejecuciones, por más que el rol que las monitoreaba no se encontraba activo y actualizando el metadato correspondiente.

6.8.3. Ejecución de MCell fallida

Para lograr una ejecución fallida de MCell se utilizaron los datos expuestos en la tabla 6.6, donde se puede observar la definición de varias reacciones $A + B -> C$, pero sin una definición de la molécula C.

Para esta simulación el sistema realiza el barrido paramétrico generando ocho combinaciones con los rangos brindados, obteniendo por tanto ocho subtareas MapReduce como se muestra en la Figura 6.1.

Al descargar la salida de MCell se observa que la razón de la falla es la

Cuadro 6.6: Parámetros de entrada para una simulación fallida

Parámetro	**Valor**
ITERATIONS	1000
TIME_STEP	$1E-05$
COORDENADAS	$[-1.5,-1.5,-1.5]$ y $[1.5,1.5,1.5]$
ASPECT_RATIO	-
MOLECULAS	A {Constante de difusión $= 5.5E-07$} y B {Constante de difusión$= 3E-07$}
REACCIONES	**Reacción** **Rate** $A+B->C$ de 0.1 a 0.2 con paso 0.1 $A+B->C$ de 0.3 a 0.5 con paso 0.1 $A+B->C$ de 0.5 a 0.6 con paso 0.1
MODO FERNET	POINT

falta de definición de la molécula C en la configuración inicial, como se puede apreciar en la captura de imagen de la salida en la Figura 6.2. Al observar la salida de la ejecución de MCell el usuario puede corregir los errores en la definición de parámetros y volver a lanzar la simulación en el sistema.

Figura 6.1: Resultados al ejecutar una simulación errónea.

6.8.4. Falla de la tarea MapReduce

Para generar una falla en la ejecución de la tarea MapReduce se alteró el código para que lance una excepción en el primero de los dos *mappers* y posteriormente se subió al cloud el nuevo .jar por medio del Azure Storage

```
point-139.3 MCell initializing simulation...
MCell[0]: random sequence 1
MCell 3.2.1 [unofficial revision]
  Running on workernode3 at SOH
  Copyright (C) 2006 - 2013 by
    The National Center for Multiscale Modeling of Biological Systems,
    The Salk Institute for Biological Studies, and
    Pittsburgh Supercomputing Center, Carnegie Mellon University,


************************************************************************
MCell development is supported by the NIGMS-funded (P41GM103712)
National Center for Multiscale Modeling of Biological Systems (MMBioS)
Please acknowledge MCell in your publications.
************************************************************************

Defining molecules with the following theoretical average diffusion distances:
  l_r_bar=0.0529256743 um for A
  l_r_bar=0.039088201 um for B

Fatal error: On line: 50 of file entradaMap.mdl
Undefined molecule: C
```

Figura 6.2: Salida de Mcell al ejecutar una simulación errónea.

Explorer, sin existir la necesidad de desplegar nuevamente el *Cloud Service*. Para la ejecución de la prueba se utilizaron los valores descritos en la Tabla 6.6, existiendo una única diferencia: no se define ninguna reacción, provocando la generación de una única tarea MapReduce. Al terminar la ejecución se pudo observar en la tabla de resultados como la tarea fue catalogada como "Tarea fallida", sin existir la posibilidad de descargar el MDL, ya que el mismo no fue generado por el *mapper*. Esto significa que el *mapper* y sus reintentos lanzados por el clúster HDInsight fallaron y es posible que exista una falla en la infraestructura o en la configuración de las tareas MapReduce, por ejemplo.

6.9. Resumen

Se realizaron pruebas sobre los algoritmos desarrollados y la integración de los distintos componentes que conforman la solución implementada, así como de los servicios de autoescalado y de disponibilidad brindados por Azure.

Los resultados muestran que los tiempos de ejecución de los algoritmos de barrido paramétrico y balanceo de carga no presentan un *overhead* considerable si se evalúa el tiempo total de simulación que varía entre varios

minutos y horas. Por otro lado, se verificó la gran escalabilidad obtenida al utilizar los servicios de autoescalado de Azure, así como la disponibilidad de los componentes desplegados en el cloud siguiendo las condiciones del contrato de servicio del proveedor. Se realizaron ejecuciones de simulaciones de gran porte obteniendo resultados satisfactorios, y demostrando así que es posible liberar el entorno local del usuario, utilizando el portal desarrollado en el proyecto para ejecutar grandes simulaciones en el cloud.

Se realizaron pruebas exhaustivas en los diferentes roles para generar fallas y observar la reacción del sistema frente a éstas. Fue necesario modificar la estrategia de mitigación de fallos en algunos escenarios, pero las pruebas realizadas mostraron finalmente una buena respuesta del sistema frente a la falla de los distintos roles que lo conforman. En caso de finalizar la ejecución de la simulación se le brinda al usuario los resultados correspondientes, y en caso de generarse errores en las ejecuciones se le brinda herramientas al usuario para lograr localizar el punto de fallo y la razón del mismo.

Capítulo 7

Conclusiones y trabajo futuro

7.1. Conclusiones

Este proyecto de grado presentó una solución para la ejecución de sistemas biológicos que permiten develar el complejo proceso de desarrollo embrionario empleando fluorescencia por microscopía aplicando el paradigma de Cloud Computing.

Se utilizaron tecnologías de computación distribuida sobre la plataforma Microsoft Azure para implementar una arquitectura altamente escalable y accesible para las simulaciones biológicas. Se integraron los sistemas de simulación MCell y FERnet a la arquitectura distribuida mediante el modelo de programación MapReduce, utilizando el servicio HDInsight de Microsoft Azure.

Como principales contribuciones del proyecto se encuentran brindar una interfaz amigable en un portal web para el ingreso de parámetros y lanzamiento de ejecuciones de MCell y FERnet, así como una sección dentro del portal para el monitoreo de tareas y obtención de resultados de las ejecuciones de las simulaciones biológicas. Se brinda también la funcionalidad de realizar barridos paramétricos para la exploración de parámetros en las simulaciones. Además, la implementación cuenta con algoritmos de balanceo de carga de servidores según métricas de utilización de recursos. La implementación del algoritmo de barrido paramétrico y balanceo de carga son un aporte fundamental para la resolución de problemas de computación de alto desempeño en el paradigma de Cloud Computing.

Se logró construir una arquitectura distribuida altamente escalable y con un alto nivel de tolerancia a fallos en los distintos roles. Además, esta arquitectura presenta un gran índice de disponibilidad de sus componentes gracias al contrato de nivel de servicio brindado por Microsoft Azure. La arquitectura construida y las pruebas realizadas sobre la misma muestran

resultados alentadores en la ejecución de grandes simulaciones y sistemas científicos de manera paralela en el cloud.

La evaluación experimental del proyecto se llevó a cabo en un ambiente de desarrollo de Microsoft Azure. Los resultados obtenidos muestran que es posible adaptar sistemas científicos como simulaciones biológicas en una arquitectura distribuida en el cloud, logrando una paralelización de ejecuciones mediante el modelo de programación MapReduce.

Como aporte general del proyecto se destaca la ejecución de simulaciones en paralelo desplegadas en una infraestructura distribuida de alto desempeño, la facilitación de la configuración y usabilidad de los sistemas de simulación MCell y FERnet, y la innovación en el área científica bajo el paradigma de Cloud Computing. Un aporte relevante del proyecto consiste en la propuesta de una arquitectura de software flexible, que puede ser utilizada para adaptar distintos tipos de sistemas al paradigma de Cloud Computing. El modelo abstracto que se presenta permite el desarrollo y la adaptación de sistemas biológicos que realicen simulaciones estocásticas, como en el caso de estudio planteado, pero también es aplicable a otros sistemas de computación científica que demandan un gran poder de cómputo y que posean características que permitan la aplicación de técnicas de paralelismo y computación de alto desempeño para mejorar sus tiempos de ejecución.

7.2. Trabajo futuro

Las principales líneas de trabajo futuro incluyen mejorar la usabilidad de la aplicación, aumentar el número de modelos y escenarios a simular, así como realizar experimentos más realistas. Estas líneas de trabajo futuro se describen brevemente a continuación.

Mejorar ejecución y configuración de FERnet. Una posible mejora se puede realizar brindando al usuario la salida producida en consola al realizarse una ejecución de FERnet, mejorando la visibilidad de este componente. Por otro lado, resta realizar la configuración correspondiente para permitir la ejecución de FERnet en los modos *line* y *frame*. Para ello, se sugieren dos posibles soluciones:

1. Optimizar la adaptación de MCell y FERnet para permitir su ejecución en el sistema operativo Windows.
2. Utilizar el servicio de HDInsight con Linux como sistema operativo huésped, que Windows Azure lanzó en forma preliminar en febrero de 2015. Con esta herramimenta se permitiría ejecutar las aplicaciones MCell y FERnet en su implementación nativa, permitiendo la ejecución de todos los modos de FERnet.

Mejoras en el modelo biológico y simulación. Un requerimiento deseable de la aplicación, consiste en la posibilidad de lanzar ejecuciones de simulaciones partiendo de su estado de equilibrio (conocido como *steady state*). Para ello, se debería comenzar la ejecución y localizar el momento en el que se llega al estado equilibrio, almacenando la información necesaria para poder lanzar simulaciones futuras partiendo de ese escenario.

Por otro lado, interesa brindar herramientas para realizar un modelado más cercano a la realidad. Para ello, el sistema debería aceptar reacciones bidireccionales. Esta mejora implica realizar un ajuste en el algoritmo de barrido paramétrico, permitiendo la definición de dos *rates* por cada reacción. Este nuevo algoritmo podría crear las posibles combinaciones de *rate* dentro de una reacción, para luego combinarla con las demás reacciones obteniendo el barrido paramétrico global. También se puede realizar en este algoritmo un refinamiento en el filtrado en los rangos de valores que se *barren*. Esta mejora se realizaría brindando una implementación precisa y dinámica del vector lógico que forma parte del algoritmo de barrido paramétrico.

Mayor seguridad. Un aspecto esencial para mejorar la seguridad de la aplicación consiste en la implementación de un sistema de registro de usuarios, con la utilización de contraseñas. De esta manera, solamente usuarios registrados pueden lanzar la ejecución de simulaciones y cada usuario podrá acceder solamente a los resultados de las ejecuciones que él mismo lanzó.

Optimización de pruebas. Si bien en la aplicación actual se realizaron diversas pruebas de rendimiento como se describió en el capítulo 6, aún resta la realización de pruebas de carga y estrés. En el primer caso, sería conveniente evaluar la aplicación bajo el escenario en el cual un gran número de usuarios utilicen la aplicación de manera concurrente. En el segundo caso, se sugiere realizar varias pruebas de carga intensivas en relación al porte de las simulaciones, permitiendo evaluar el rendimiento real. Ambas evaluaciones se pueden realizar utilizando un tipo de proyecto de pruebas brindado por Visual Studio denominado “proyecto de pruebas de carga y rendimiento web”, donde se graba una serie de pasos utilizando el portal web y se indica la cantidad de usuarios concurrentes que se desea simular.

Bibliografía

[1] J.M. Arévalo Navarro. Cloud computing: fundamentos, diseño y arquitectura aplicados a un caso de estudio. Master's thesis, Universidad Rey Juan Carlos, 2011.

[2] Microsoft Azure. Blob Service REST API. https://msdn.microsoft.com/en-us/library/azure/dd135733.aspx. [Online]. Accedido Marzo 2015.

[3] Microsoft Azure. Contratos de nivel de servicio. http://azure.microsoft.com/es-es/support/legal/sla/. [Online]. Accedido Enero 2015.

[4] Microsoft Azure. Crear y cargar un certificado de administración para Azure. https://msdn.microsoft.com/es-es/library/azure/gg551722.aspx. [Online]. Accedido Enero 2015.

[5] Microsoft Azure. Create a service certificate for Azure. https://msdn.microsoft.com/en-us/library/azure/gg432987.aspx. [Online]. Accedido Enero 2015.

[6] Microsoft Azure. HDInsight's security patterns and practices. https://msdn.microsoft.com/en-us/library/dn768256.aspx. [Online]. Accedido Enero 2015.

[7] Microsoft Azure. How to create and deploy a Cloud Service. http://azure.microsoft.com/en-us/documentation/articles/cloud-services-how-to-create-deploy/#prepare. [Online]. Accedido Setiembre 2014.

[8] Microsoft Azure. Pague solo por lo que usa. http://azure.microsoft.com/es-es/pricing/calculator/?scenario=full. [Online]. Accedido Marzo 2015.

[9] Microsoft Azure. Release notes for the Service Bus. https://msdn.microsoft.com/en-us/library/azure/hh667331.aspx. [Online]. Accedido Febrero 2015.

[10] Microsoft Azure. Roleentrypoint.Run method. https://msdn.microsoft.com/en-us/library/microsoft.windowsazure.serviceruntime.roleentrypoint.run.aspx. [Online]. Accedido Diciembre 2014.

[11] Microsoft Azure. Tamaños de máquinas virtuales y servicios en la nube de Azure. https://msdn.microsoft.com/es-es/library/azure/dn197896.aspx. [Online]. Accedido Octubre 2014.

[12] Microsoft Azure. Windows Azure Management Portal. manage.windowsazure.com. [Online]. Accedido Marzo 2015.

[13] B. Collette. Things to consider when implementing a Load Balancer with WCF. https://msdn.microsoft.com/en-us/library/vstudio/hh273122(v=vs.100).aspx. [Online]. Accedido Noviembre 2014.

[14] M. Da Silva, S. Nesmachnow, M. Geier, E. Mocskos, J. Angiolini, V. Levi, and A. Cristobal. Efficient fluorescence microscopy analysis over a volunteer grid/cloud infrastructure. In *High Performance Computing*, volume 485 of *Communications in Computer and Information Science*, pages 113–127. Springer Berlin Heidelberg, 2014.

[15] L. De la Peña. *Introducción a la Mecánica Cuántica (3ra Edición)*. UNAM, Facultad de Ciencias, 2006.

[16] M. Digman and E. Gratton. Lessons in fluctuation correlation spectroscopy. In *Annual Review of Physical Chemistry*, pages 645–668, 2011.

[17] T. Erl, R. Puttini, and Z. Mahmood. *Cloud Computing: Concepts, Technology & Architecture*. Prentice Hall Press, Upper Saddle River, NJ, USA, 1st edition, 2013.

[18] A. Galindo and P. Pascual. *Mecánica cuántica Ed. Eudema*. Eudema, 1989.

[19] D. Garber, J. Malik, and A. Fazio. *Windows Azure Hybrid Cloud*. Wrox Press Ltd., Birmingham, UK, 1st edition, 2013.

[20] S. García, S. Iturriaga, and S. Nesmachnow. Scientific computing in the Latin America-Europe GISELA grid infrastructure. In *Proceedings of the 4th High–Performance Computing Latin America Symposium HPCLatAm*, pages 48–62, 2011.

[21] D. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, 22(4):403–434, 1976.

[22] M. González. Fluorescencia. `http://quimica.laguia2000.com/`. [Online]. Accedido Marzo 2015.

[23] E. Haustein and P. Schwille. Ultrasensitive investigations of biological systems by fluorescence correlation spectroscopy. *Methods*, 29:153–166, 2003.

[24] E.A. Inria and N. Shimkin. Individual equilibrium and learning in processor sharing systems. *Oper. Res.*, 46(6):776–784, 1998.

[25] P. Jakovits and S.N. Srirama. Adapting scientific applications to cloud by using distributed computing frameworks. *Cluster Computing and the Grid, IEEE International Symposium on*, pages 164–167, 2013.

[26] R.A. Kerr, T.M. Bartol, B. Kaminsky, M. Dittrich, J.-C.J. Chang, S.B. Baden, T.J. Sejnowski, and J.R. Stiles. Fast Monte Carlo Simulation Methods for Biological Reaction-Diffusion Systems in Solution and on Surfaces. *SIAM Journal on Scientific Computing*, 30(6):3126–3149, 2008.

[27] I. Landa Martín and U. Zorrilla Castro. Introducción a Windows Azure. `http://es.slideshare.net/jhoBGP/parte-1-introduccion-a-windows-azure`. [Online]. Accedido Agosto 2014.

[28] G. Mantri. Consuming Windows Azure service management API from web/worker role–the easy way. `goo.gl/hbDFlt`. [Online]. Accedido Noviembre 2014.

[29] D.C. Marinescu. *Cloud Computing: Theory and Practice*. Elsevier Science, 2013.

[30] Mell, P. and Grance, T. The NIST Definition of Cloud Computing. [Online]. Accedido Mayo 2014.

[31] J. Nilsson. Hadoop MapReduce in Eucalyptus Private Cloud. Master's thesis, Umeå University, 2011.

[32] Oracle. What are RESTful Web Services? `http://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html`. [Online]. Accedido Marzo 2015.

[33] M. Ortega and S. Nesmachnow. HPC applied to fluorescence fluctuation analysis: contributing to unravel hidden dynamical processes. Technical report, Centro de Cálculo, Facultad de Ingeniería, Universidad de la República, 2013.

[34] M. Ortega, S Nesmachnow, J. Angiolini, V. Levi, and E. Mocskos. HPC applied to fluorescence fluctuation analysis: contributing to unravel hidden dynamical processes. In *5th International Supercomputing Conference*, Ensenada, Baja California, Mexico, 2014.

[35] S.L. Pallickara, M. Pierce, Q. Dong, and C.H. Kong. Enabling large scale scientific computations for expressed sequence tag sequencing over Grid and Cloud Computing clusters. In *Eighth International Conference on Parallel Processing and Applied Mathemathics (PPAM 2009)*, pages 13–16, Wroclaw, Poland, 2009. Springer-Verlag.

[36] T. Parks. Mustache Sharp. `https://github.com/jehugaleahsa/mustache-sharp`. [Online]. Accedido Noviembre 2014.

[37] C.L. Philip Chen and C.-Y. Zhang. Data-intensive applications, challenges, techniques and technologies: A survey on big data. *Information Sciences*, 275:314–347, 2014.

[38] Hortonworks Data Platform. Determine YARN and MapReduce Memory Configuration Settings. `http://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.0.6.0/bk_installing_manually_book/content/rpm-chap1-11.html`. [Online]. Accedido Enero 2015.

[39] S. Reyes Ávila. Desarrollo de un workflow genérico para el modelado de problemas de barrido paramétrico en sistemas distribuidos. Master's thesis, Universitat Politécnica De Catalunya, 2012.

[40] R. Richman, H. Zirnhelt, and S. Fix. Large-scale building simulation using cloud computing for estimating lifecycle energy consumption. *Canadian Journal of Civil Engineering*, pages 252–262, 2014.

[41] J. Roth. Azure subscription and service limits, quotas, and constraints. `http://azure.microsoft.com/en-us/documentation/articles/azure-subscription-service-limits/`. [Online]. Accedido Febrero 2015.

[42] J.C. Ruiz Pacheco. What is Windows Communication Foundation? `https://msdn.microsoft.com/library/ms731082%28v=vs.110%29.aspx`. [Online]. Accedido Marzo 2015.

[43] Amazon Web Services. AWS compliance. `http://aws.amazon.com/es/compliance/`. [Online]. Accedido Febrero 2015.

[44] Amazon Web Services. Precios de Amazon S3. `http://aws.amazon.com/es/s3/pricing/`. [Online]. Accedido Febrero 2015.

[45] S.N. Srirama, V. Ivanistsev, P. Jakovits, and C. Willmore. Direct migration of scientific computing experiments to the cloud. In *HPCS*, pages 27–34. IEEE, 2013.

[46] S.N. Srirama, P. Jakovits, and E. Vainikko. Adapting scientific computing problems to clouds using mapreduce. *Future Gener. Comput. Syst.*, 28(1):184–192, 2012.

[47] J. Stiles and T. Bartol. Monte Carlo methods for simulating realistic synaptic microphysiology using MCell. In *Computational Neuroscience: Realistic Modeling for Experimentalists*, pages 87–127, Boca Raton, FL, USA, 2001. CRC Press, Inc.

[48] B. Swan. Maximizing HDInsight throughput to Azure Blob Storage. http://blogs.msdn.com/b/brian_swan/archive/2013/11/25/maximizing-hdinsight-throughput-to-azure-blob-storage.aspx. [Online]. Accedido Enero 2015.

[49] Eucalyptus Systems. Eucalyptus. https://www.eucalyptus.com/. [Online]. Accedido Julio 2014.

[50] Thraka. Compute hosting options provided by Azure. http://azure.microsoft.com/en-us/documentation/articles/fundamentals-application-models/. [Online]. Accedido Febrero 2015.

[51] J.S. van Zon and P.R. ten Wolde. Simulating Biochemical Networks at the Particle Level and in Time and Space: Green's Function Reaction Dynamics. *Phys. Rev. Lett.*, 94:128103, 2005.

[52] J. Varia. Amazon Web Services - architecting for the cloud: Best practices. White paper, Amazon Web Services, Enero 2010.

[53] T. Velte, A. Velte, and R. Elsenpeter. *Cloud Computing, A Practical Approach.* McGraw-Hill Education, 2009.

[54] P. Werstein, H. Situ, and Z. Huang. Load balancing in a cluster computer. In *Proceedings of the Seventh International Conference on Parallel and Distributed Computing, Applications and Technologies*, PDCAT '06, pages 569–577, Washington, DC, USA, 2006. IEEE Computer Society.

[55] T. White. *Hadoop: The Definitive Guide.* O'Reilly Media, Inc., 2nd edition, 2009.

Apéndice A

Anexo I: Compilar MCell y FERnet en Windows

A.1. Compilar MCell en Windows

Para compilar MCell en Windows se deben seguir los pasos que se detallan a continuación.

1. Instalar MinGW y cygwin.
2. Instalar librerías necesarias para Mcell en MinGW: flex, gawk, bison, byacc.
3. Con todas las clases de Mcell generar un proyecto en codeblocks.
4. Desde el terminal cygwin ejecutar el archivo *version.sh* (que es parte del proyecto), este genera un archivo *version.h*, renombrarlo a *version2.h* y agregarlo al proyecto de Codeblocks.

A.1.1. Configuración del compilador

En CodeBlocks ir al menú *Configuración* seleccionar la opción *Compilador* y en la pestaña *Configuraciones del compilador* seleccionar las opciones que se muestran en la figura A.1.

Hacer clic con el botón derecho sobre el proyecto y seleccionar *Opciones de Compilación* y en la pestaña *Banderas del compilador* seleccionar la opción *Producir símbolos de debug [-g]* como se muestra en la figura A.2

Realizar los siguientes cambios en el código:

Cambios a realizar en el código			
Nombre de Archivo	**Codigo nuevo**	**Codigo anterior**	**Línea**
argparse.c	#include «config-win.h»	#include «config.h»	23

bootstrap	#! /bin/perl	#! /bin/sh	1
chkpt.c	#include «config-win.h»	#include «config.h»	30
config-win.h	#include ¡Windows.h¿	#include ¡windows.h¿	60
config-win.h	#include ¡ctype.h¿	-	66
config-win.h	#include "limits.h"	-	linea 66
config-win.h	typedef int errno_t	typedef int int	71
config-win.h	todas las ocurrencias de errno_t	por int	-
config-win.h	errno_t err = _ctime32_s(buf, buflen, timep)	int err = 0	114
config-win.h	#define ARRAYSIZE(a) sizeof(a)/sizeof(a[0])	-	120
config-win.h	en método _win_rename new	new2	495
count_util.c	#include «config-win.h»	#include «config.h»	31
diffuse.c	#include «config-win.h»	#include «config.h»	31
diffuse_util.c	#include «config-win.h»	#include «config.h»	32
grid_util.c	#include «config-win.h»	#include «config.h»	31
init.c	#include «config-win.h»	#include «config.h»	23
init.c	mover #include "mdlparse_aux.h.ª la sección superior de las inclusiones	-	62
isaac64.c	#include «config-win.h»	#include «config.h»	29

libmcell.h	#include «config-win.h»	#include «config.h»	27
logging.c	#include «config-win.h»	#include «config.h»	23
macromolecule.c	#include «config-win.h»	#include «config.h»	23
mcell_engine.c	#include «config-win.h»	#include «config.h»	23
mcell_engine.h	#include «config-win.h»	#include «config.h»	26
mcell_structs.h	#include «config-win.h»	#include «config.h»	26
mdlparse.y	#include «config-win.h»	#include «config.h»	2
mdlparse_util.c	#include «config-win.h»	#include «config.h»	23
mem_util.c	#include «config-win.h»	#include «config.h»	32
minrng.c	#include «config-win.h»	#include «config.h»	23
react_cond.c	#include «config-win.h»	#include «config.h»	31
react_outc.c	#include «config-win.h»	#include «config.h»	31
react_output.c	#include «config-win.h»	#include «config.h»	23
react_trig.c	#include «config-win.h»	#include «config.h»	31
react_util.c	#include «config-win.h»	#include «config.h»	31
rng.c	#include «config-win.h»	#include «config.h»	23
sched_util.c	#include «config-win.h»	#include «config.h»	31
strfunc.c	#include «config-win.h»	#include «config.h»	25
sym_table.c	#include «config-win.h»	#include «config.h»	23
util.c	#include «config-win.h»	#include «config.h»	23
vector.c	#include «config-win.h»	#include «config.h»	48

version_info.c	#include «config-win.h»	#include «config.h»	23
version_info.c	#include "version2.h"	#include "version.h"	25
version_info.c	#include "winver.h"	-	26
viz_output.c	#include «config-win.h»	#include «config.h»	23
viz_output.c	comentar "mkfifo(fname,0666)"	-	7389
vol_util.c	#include «config-win.h»	#include «config.h»	31
volume_output.c	#include «config-win.h»	#include «config.h»	23
wall_util.c	#include «config-win.h»	#include «config.h»	31

A.2. Compilar FERnet en Windows

Para compilar FERnet en Windows se deben seguir los pasos que se detallan a continuación.

1. Crear proyecto en CodeBlocks con los archivos fuente.
2. Obtener las librerias libconfig, libtiff y argtables.
3. Agregar las librerías al proyecto.
4.

A.2.1. Configuración del compilador

En la sección "Configuración" seleccionar la opción "Compilador" y luego marcar las opciones que se muestran en la figura A.1 como se muestra en la pestaña "Configuración del Compilador"

En la pestaña "Directorios de búsqueda" agregar las ubicaciones donde se encuentran las carpetas denominadas "include" y "Debug Win32" de de GnuWin32 y de la librería "libconfig" respectivamente.

Haciendo clic derecho en el proyecto, seleccionar "Propiedades" y luego "Opciones del compilador". En la pestaña "banderas del compilador" seleccionar la opción que se muestra en la figura A.3.

Luego, en la pestaña "Configuración de Link" agregar las ubicaciones donde se encuentran las carpetas denominadas "include" y "Debug Win32" de de GnuWin32 y de la librería "libconfig" respectivamente.

Modificar el código del archivo fernet.h de la siguiente forma.

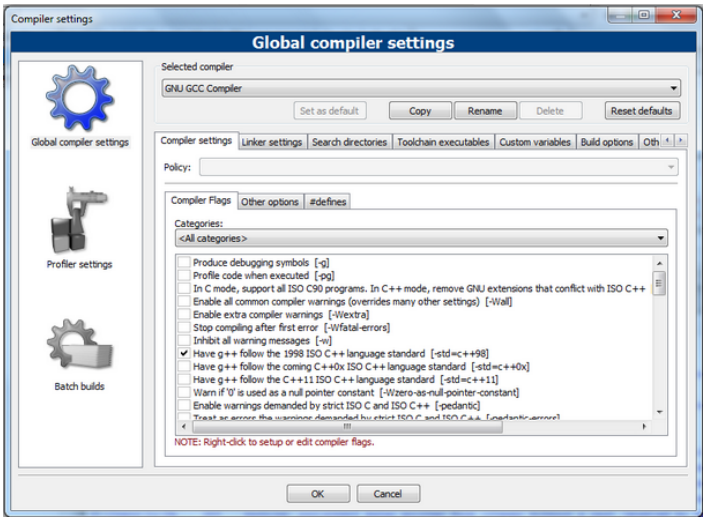

Figura A.1: Opciones del compilador

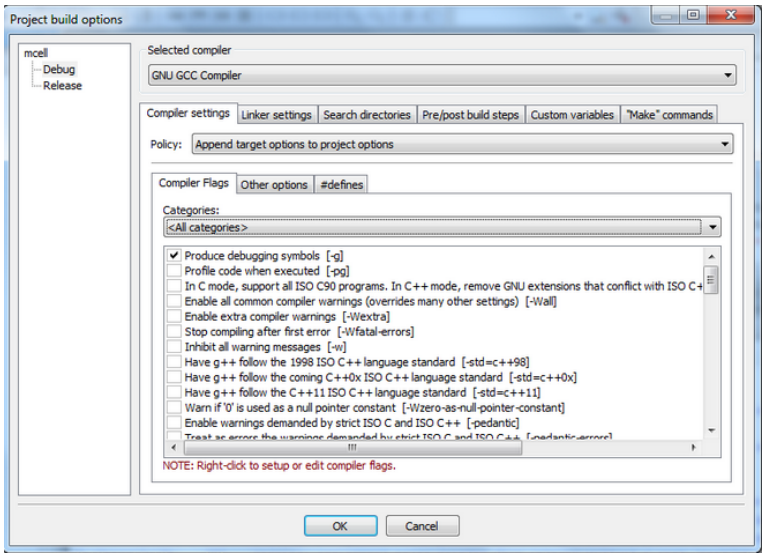

Figura A.2: Opciones del compilador

- Modificar #include “argtable2.h” por #include ¡argtable2.h¿Linea 28
- Modificar #include “libconfig.h” por #include ¡libconfig.h¿Linea 29

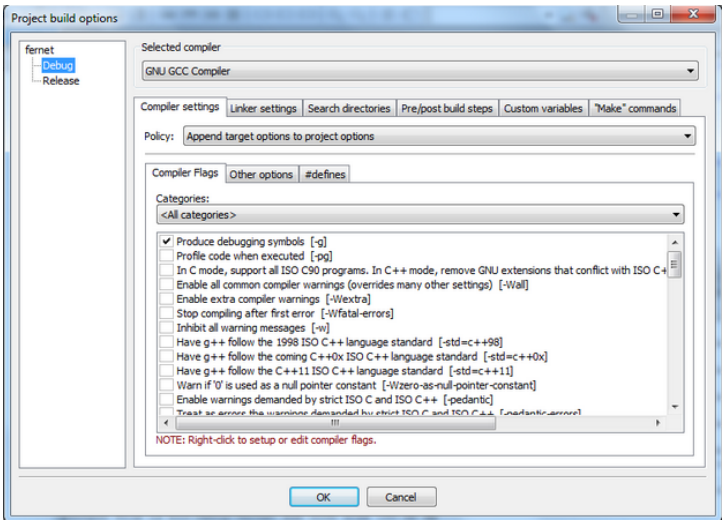

Figura A.3: Opciones del compilador