Facultad 6
Departamento de Programación y Sistemas Digitales

Modelado y simulación de un Sistema de Almacenamiento Distribuido en un entorno de estaciones de trabajo.

Trabajo final presentado en opción al título de Máster en Informática Aplicada

Autora: Ing. Mónica Teresa Llorente Quesada
Tutor: MSc. Longendri Aguilera Mendoza
Tutora: MSc. Yeleny Zulueta Véliz

Ciudad de La Habana, Noviembre del 2013

TESIS EN OPCIÓN AL GRADO DE MÁSTER EN
INFORMÁTICA APLICADA

Modelado y simulación de un Sistema de Almacenamiento Distribuido en un entorno de estaciones de trabajo

Autora
Ing. Mónica Teresa Llorente Quesada [1]
E-mail: mllorente@uci.cu

Tutor
Msc. Longendri Aguilera Mendoza [1]
Profesor Asistente
E-mail: loge@uci.cu

Tutora
Msc. Yeleny Zulueta Véliz [1]
Profesora Auxiliar
E-mail: yeleny@uci.cu

[1] Universidad de las Ciencias Informáticas
Carretera San Antonio de los Baños Km. 2 $\frac{1}{2}$
Torrens, Boyeros,
Ciudad de La Habana, Cuba.

"Año 55 de la Revolución"

Dedicatoria

A Loge: Mi Esposo, mi Amor, mi Vida, mi Fuerza, mi Todo.

A mi madre: Donde quiera que esté gracias por darme fuerzas.

A mi hermosa familia: Por su amor incondicional y su confianza sin límites.

Agradecimientos

A mi familia y mis amigos por apoyarme en todo momento.

En especial a Yele, por ser mi defensora contra vientos y mareas.

A mis suegros porque siempre estuvieron al tanto de todo.

A mis vecinos, por estar pendientes y preocupados en todo momento.

Declaración jurada de autoría

Declaro que yo, Mónica Teresa Llorente Quesada, con carné de identidad 84072724337, soy la autora principal del trabajo final de maestría: Modelado y simulación de un Sistema de Almacenamiento Distribuido para un clúster de estaciones de trabajo, desarrollado como parte de la Maestría en Informática Aplicada y que autorizo a la Universidad de las Ciencias Informáticas a hacer uso de la misma en su beneficio, así como los derechos patrimoniales con carácter exclusivo.Y para que así conste, firmo la presente declaración jurada de autoría en:

Ciudad de La Habana a los días del mes de del año 2013.

Mónica Teresa Llorente Quesada

Firma de la Autora

MSc. Yeleny Zulueta Véliz

Firma de la Tutora

MSc. Longendri Aguilera Mendoza

Firma del Tutor

Resumen

En la actualidad la capacidad de almacenamiento de los discos duros se ha incrementado de Gigabytes a Terabytes, razón por lo cual es frecuente encontrar una cantidad considerable de espacio libre en las computadoras de una institución. Con el fin de que se pueda aprovechar este espacio de almacenamiento en una institución como la Universidad de las Ciencias Informáticas, donde no se tiene el control de la disponibilidad de las estaciones de trabajo, en el presente trabajo se realiza el modelado y simulación de un Sistema de Almacenamiento Distribuido (SAD) que tiene en cuenta las características del entorno donde se desea desplegar la solución y los fallos aleatorios provocados por la pérdida de datos y el apagado de las máquinas a voluntad de los usuarios. El modelo construido se basó en la arquitectura de software y la lógica de funcionamiento del producto informático *Hadoop Distributed File System* (HDFS), un sistema muy popular de Apache. Su implementación se realizó en la herramienta CPN Tools utilizando las Redes de Petri Coloreadas como lenguaje de modelado y el lenguaje de programación CPN ML para enriquecer las especificaciones del modelo. Las simulaciones realizadas permitieron diagnosticar problemas que pueden surgir en el despliegue del sistema y se propusieron acciones correctivas para lograr un mejor funcionamiento del SAD. Los resultados obtenidos en las simulaciones muestran que es necesaria una modificación en el código fuente de HDFS para garantizar un correcto desempeño del sistema en un entorno donde no se tiene el control de la disponibilidad de las estaciones de trabajo.

Palabras Claves: *Hadoop Distributed File System*, modelado, Redes de Petri Coloreadas, simulación, Sistema de Almacenamiento Distribuido.

Abstract

Currently the storage capacity of hard drives has increased from Gigabytes to Terabytes, for this reason is common to find a considerable amount of free space on the computers of an institution. In order to be able to use this storage space in an institution like the University of Information Sciences, where there is no control over the availability of their workstations, a modeling and simulation of a Distributed File System (DFS) is presented in this work that takes into account the characteristics of the environment where you want to deploy the solution and random failures caused by data loss and power off. The model built is based on the software architecture and the programming logic of product Hadoop Distributed File System, very popular software of Apache. It was implemented in the tool CPN Tools using Coloured Petri Nets as the modeling language and the programming language CPN ML to enrich the model specifications. Simulations allowed diagnose problems that may arise in the deployment of the system and proposed corrective actions to achieve a better functioning of DFS. The results of the simulations show that a modification is necessary in HDFS source code to ensure proper system performance in an environment where there is no control over the availability of the workstations.

Keywords: Coloured Petri Nets, Distributed Storage System, Hadoop Distributed File System, modeling, simulation.

Índice general

Índice de figuras

Introducción

Los medios de almacenamiento evolucionan en forma notable desde las primeras computadoras [1, 2]. A lo largo de los años los discos duros han disminuido de tamaño y precio, aumentando significativamente la cantidad de datos que pueden albergar. En la actualidad la capacidad de almacenamiento disponible en el disco duro de las estaciones de trabajo se ha incrementado notablemente de GBs a TBs[1].

Según estudios realizados [3,4], es frecuente encontrar cantidades significativas de espacio libre de almacenamiento en las computadoras (PCs) de una institución. Lo cual puede ser interpretado como un desaprovechamiento de los recursos computacionales en los que se ha invertido para ponerlos en función de los usuarios.

Con el propósito de aprovechar la cantidad de espacio no utilizado en las estaciones de trabajo, se desarrollan los Sistemas de Almacenamiento Distribuido (SAD): una combinación de hardware y software, para tomar ventajas a través de la red de esta capacidad de almacenamiento disponible [5]. Para ejemplificar el éxito de este tipo de soluciones, un estudio realizado sobre 330 000 computadoras pertenecientes a usuarios comunes muestra las enormes potencialidades de estos sistemas cuando son aplicados a gran escala [6].

Motivación

En la Universidad de las Ciencias Informáticas (UCI), creada en Cuba en el año 2002, se encuentra la infraestructura a nivel nacional con el mayor número de PCs disponibles en una red de

[1] 1 TB = 10^3 GB = 10^6 MB = 10^9 KB = 10^{12} bytes

área local. Actualmente cuenta, aproximadamente, con un poco más de 8 000 estaciones de trabajo distribuidas por todo el campus universitario.

Una gran parte de estas máquinas se encuentran ubicadas en los laboratorios docentes y en las aulas, utilizándose para impartir clases, estudiar, consultar sitios web y revisar el correo electrónico en la mayoría de los casos, siendo los documentos de textos, hojas de cálculos y presentaciones digitales, los archivos más comunes de encontrar en el disco duro de estas computadoras. En estas nos detendremos para desarrollar el presente trabajo.

Un estudio realizado durante 40 días (del 1 de noviembre al 10 de diciembre de 2012) en las PCs dedicadas a la docencia del docente 4, permitió analizar el comportamiento (figura 1) del espacio libre en el disco duro de las estaciones de trabajo durante un determinado periodo de tiempo. Para ello, se recogieron métricas utilizando el Sistema de Monitoreo y Explotación de las Estaciones de Trabajo (SIMON) desarrollado por el departamento de Almacenes de Datos y Bioinformática del Centro de Tecnologías y Gestión de Datos (DATEC).

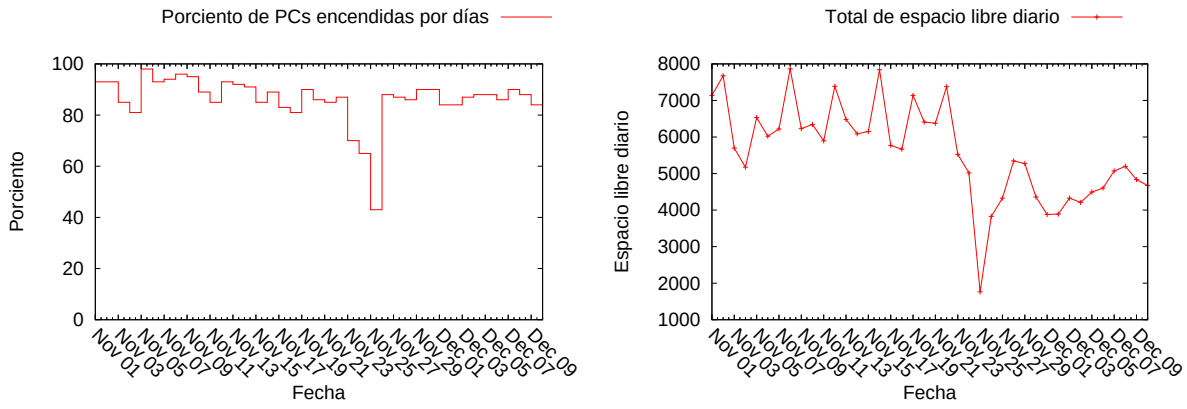


Figura 1: Cantidad de PCs encendidas por días (a) y total de espacio libre en GB por día (b).

En la figura 1(a) aparece el número de PCs que se reportaron, al menos una vez, diariamente durante la recogida de los datos y en la figura 1(b) se muestra la capacidad total de espacio libre, en GB, con la que se pudo contar cada día. Es lógico que a medida que pasa el tiempo el espacio libre disminuya. Sin embargo, sigue siendo considerable la cantidad que no está siendo aprovechada. El valor mínimo de la gráfica, que se obtiene cuando solo se reportan menos de 60 PCs, llega a aglutinar más de 1 TB de almacenamiento y el valor máximo casi alcanza los 8 TB.

Considerando que cada vez la capacidad de los discos es mayor y que por reposición estas máquinas pueden llegar a tener 500 GB o más de capacidad, se puede decir que una gran parte de todo el espacio disponible que se tiene de almacenamiento no está siendo utilizada. Esta es la principal motivación que se tuvo para comenzar a estudiar cómo podría funcionar un Sistema de Almacenamiento Distribuido en la universidad.

Planteamiento del Problema

En un SAD para el entorno UCI estarían presentes las estaciones de trabajo como principales componentes del sistema, por la gran cantidad de PCs que existen en los laboratorios y aulas de dicha institución. Es por ello, que se debe tener en cuenta que la red local puede sufrir pérdida de conectividad, las estaciones de trabajo se pueden apagar cuando no se hace uso de ellas y que algunos datos se pudieran perder por fallos en los discos duros o por necesidad de reinstalar una computadora.

Para enfrentar estas situaciones los SAD plantean una solución: la replicación de datos, donde se trata de que los archivos guardados siempre estén disponibles en más de una computadora y que se tenga acceso a los mismos, de ser posible, por más de una ruta de red. Esto aunque parezca sencillo realmente no lo es y una mala estrategia de replicación influye directamente en la tolerancia a fallos del sistema y en la mala utilización del espacio libre en disco duro.

A modo de ejemplo, si se trata de algo tan simple como el tamaño en el que se fragmentan los datos para ser guardados, fragmentos pequeños y numerosos pueden tener como inconveniente que implique más trabajo y tiempo la localización y unión para formar el archivo guardado; mientras que los fragmentos muy grandes, aunque fueran pocos, consumen mayor tiempo al ser transferidos por la red e incluso pudieran no encontrar el suficiente espacio disponible para ser almacenados o replicados.

Varias son las preguntas que hay que contestarse para realizar una adecuada replicación de datos: ¿Qué tamaño de bloque deben tener los fragmentos de los archivos? ¿Cuándo se debe realizar la réplica de dichos fragmentos? ¿Cuántas réplicas hacer de cada uno? ¿Dónde colocar las réplicas?,

entre otras. Las respuestas a estas interrogantes en un entorno no controlado como la UCI, donde las computadoras se reinstalan periódicamente y se apagan y encienden a voluntad de los usuarios, requieren un estudio previo para determinar bajo qué condiciones puede funcionar correctamente un SAD.

Sería muy arriesgado desplegar un sistema de esta envergadura sin realizar un estudio previo para analizar su futuro comportamiento. Además, no existe un lugar con características similares donde pueda ser desplegado el sistema para evaluar su funcionamiento. Las consecuencias de no hacer tal estudio y experimentar, pudieran ser la pérdida definitiva de los datos almacenados y la sobre carga innecesaria de la red.

Formulación del Problema

Teniendo en cuenta lo anteriormente planteado se presenta como problema de la investigación: ¿Cómo determinar el comportamiento que tendría un SAD, con antelación a su despliegue, en un entorno donde no se tiene el control de la disponibilidad de las estaciones de trabajo, con el fin de contribuir en la toma de decisiones para su correcto funcionamiento?

Objeto de Investigación

Para dar respuesta al problema planteado se definió el siguiente **objeto de investigación:** El almacenamiento distribuido de datos

Objetivo General

Para darle solución al problema planteado se define como objetivo general de la investigación: Desarrollar un modelo de simulación que permita determinar el comportamiento que tendría un SAD, con antelación a su despliegue, teniendo en cuenta las características y disponibilidad de las estaciones de trabajo, con el fin de contribuir en la toma de decisiones para su correcto funcionamiento.

Objetivos Específicos

Para definir detalladamente lo que se desea obtener el objetivo general se desglosó en los siguientes objetivos específicos:

- Elaborar el marco teórico conceptual de la investigación relacionado con los SAD, lenguajes de modelado y herramientas de simulación.
- Implementar el modelo de un SAD reportado en la literatura, teniendo en cuenta las características y disponibilidad de las estaciones de trabajo del entorno donde se desea desplegar el sistema.
- Validar el modelo desarrollado comparando los resultados de las simulaciones con el sistema desplegado en un entorno real de prueba.
- Evaluar el comportamiento del sistema modelado mediante la realización de simulaciones y análisis de sus resultados.

Campo de Acción

Como parte del objeto de investigación, se definió como **campo de acción:** Modelado y simulación de los Sistemas de Almacenamiento Distribuido.

Hipótesis de investigación

Al concluir la revisión de la literatura y luego de elaborar el marco teórico, se formula la siguiente **hipótesis de investigación**:Si se desarrolla un modelo de simulación de eventos discretos para un SAD, utilizando variables y funciones estadísticas de distribución para representar las características y disponibilidad de las estaciones de trabajo, se logrará determinar el comportamiento que tendría el sistema, con antelación a su despliegue, y contribuir en la toma de decisiones para su correcto funcionamiento.

En la realización de la investigación se destaca la utilización de los siguientes **métodos científicos**:

Métodos teóricos:

Analítico-sintético, para la realización del análisis, la valoración y evaluación del sistema, en concordancia a los conocimientos expuestos en la literatura consultada.

El método **histórico-lógico** y el **dialéctico** para el estudio de trabajos anteriores que sentaron las bases para el desarrollo de la investigación.

El método **hipotético-deductivo** para elaborar la hipótesis de investigación como resultado del marco teórico de la investigación.

La **modelación** que contribuye al desarrollo del modelo, para describir el funcionamiento del sistema.

Métodos empíricos:

Estudio de casos, para la recolección de los datos provenientes de un caso real y la comparación de resultados reales con los del modelo.

Importancia de la investigación

Un SAD para la UCI sería muy útil debido a la gran cantidad de computadoras presentes en la infraestructura de esa institución, principalmente si se tiene en cuenta que está siendo subutilizado el espacio libre del disco duro de las computadoras que se encuentran en los laboratorios docentes y aulas. Con una solución de este tipo, se podría prestar en un futuro el servicio de almacenamiento para otras instituciones con menos recursos computacionales y para los proyectos de investigación que manejan grandes volúmenes de datos. Sin embargo, lograrlo no es una tarea fácil, el desarrollo y despliegue de un sistema de este tipo es complejo por la cantidad de variables que se deben tener en cuenta para garantizar su buen desempeño.

El hecho de modelar y realizar simulaciones previas al despliegue de un sistema de este tipo permite:

- Evaluar el desempeño del sistema sin interrumpir el funcionamiento de la universidad ni cargar

la red de datos innecesariamente.

- Diagnosticar problemas que pueden ocurrir en un futuro.
- Estudiar el efecto de cambios haciendo alteraciones al modelo.
- Facilitar la exploración de varias alternativas de configuración y seleccionar la que garantice un buen desempeño del sistema.

Resultados esperados

En la presente investigación se obtendrá el modelo de un SAD para analizar su comportamiento en un entorno no dedicado, como el que se tiene en la UCI. Además se puede estudiar su comportamiento en otros entornos gracias a todas las configuraciones que se pueden explorar con solo modificar sus parámetros.

Para la obtención del modelo, se tendrán en cuenta varios parámetros que inciden en el comportamiento del sistema, por ejemplo: el número de solicitudes hechas por los usuarios, el número de subredes y máquinas por subredes, valores que determinan el tamaño de los archivos a almacenar, capacidad de espacio libre en las computadoras y el factor de replicación de los bloques, entre otros.

Con el modelo implementado, se realizarán varias simulaciones y los resultados a obtener podrían demostrar que se puede lograr un sistema de alta disponibilidad o, en caso contrario, mostrar las causas que afectan su funcionamiento.

Estructura del documento

En el primer capítulo se presenta el marco teórico de la investigación donde se hace un análisis crítico de la literatura y una presentación de la información recopilada que está estrechamente relacionada con el tema tratado. Se selecciona el SAD a utilizar en la investigación, así como el lenguaje de modelado y la herramienta computacional que permita la implementación y estudio del modelo.

En el segundo capítulo se describe el modelo implementado, para explicar cómo se realizó este trabajo de manera que el lector pueda repetir el estudio. El tercer y último capítulo está dedicado a ilustrar y discutir los resultados alcanzados mediante la ejecución de varias simulaciones del sistema modelado. Se muestran las conclusiones y recomendaciones que sugiere el autor para darle una continuidad a la presente investigación. Finalmente se muestran las referencias bibliográficas, así como los Anexos que apoyan la comprensión y dan información adicional sobre el trabajo realizado.

Capítulo 1

Marco teórico referencial de la investigación

En el presente capítulo se realiza una revisión bibliográfica con el fin de seleccionar un SAD para analizar su desempeño en la UCI. Debido a que el funcionamiento del sistema será analizado mediante la técnica de modelado y simulación, se expondrán las herramientas y lenguajes de modelado a utilizar para modelar y evaluar el comportamiento de este tipo de sistemas con la realización de varias simulaciones.

1.1. Sistemas de Almacenamiento Distribuido

Los SAD datan de la década del 80 del siglo pasado [7, 8] y a pesar de haber transcurrido más de 30 años desde entonces, siguen siendo objeto de investigación para mejorar aspectos esenciales tales como la transparencia, el rendimiento, la tolerancia a fallas y la escalabilidad.

Un SAD está compuesto por un grupo de computadoras conectadas entre sí, con el fin de almacenar de forma distribuida archivos de cualquier índole y sobre todo de manera transparente a los usuarios. Generalmente los usuarios se conectan a una computadora denominada servidor, la cual se encarga de atender las solicitudes, ya sea de almacenamiento o recuperación. Cada archivo a almacenar se divide en fragmentos a los que se les añade redundancia generando archivos denominados

réplicas de los datos.

Las réplicas de un mismo archivo se almacenan en distintas computadoras y subredes. De esta manera, el sistema es capaz de tolerar algunas fallas de rotura en las computadoras y redes. Para la recuperación se realiza la operación inversa; a partir de los archivos dispersos se recupera el archivo original.

1.1.1. Principales características a tener en cuenta

Para el estudio y selección de un SAD que pueda adaptarse a las condiciones de la UCI se tuvieron en cuenta las siguientes características, según las necesidades y políticas de la institución.

- **Licencia de código abierto.** El sistema debe ser de libre adquisición y uso para poder inspeccionar su código fuente, entender su funcionamiento y poder realizar el modelado y las modificaciones necesarias que garanticen un mejor desempeño en la universidad.

- **Escalabilidad.** Es una propiedad deseable en cualquier sistema distribuido, que indica su habilidad para crecer sus dimensiones sin que esto afecte la calidad del servicio prestado. En el caso de los SAD, debe aumentarse la capacidad de almacenamiento a medida que se aumenta el número de computadoras sin que esto tenga incidencias negativas en el desempeño.

- **Interfaces de programación**. El SAD seleccionado debe brindar una Interfaz de Programación de Aplicaciones (API según sus siglas en Inglés) para el desarrollo de futuros programas que hagan uso de las bondades del sistema, dada las características que tiene la institución de ser una universidad con énfasis en la producción de software.

- **Tolerancia a fallos.** No se concibe un sistema distribuido que no sea tolerante a fallos, es decir, que no tenga la capacidad de recuperarse ante un problema de conexión en la red o en las computadoras que forman parte del sistema. Esto toma mayor importancia en el entono UCI, ya que las estaciones de trabajo entran y salen del sistema de una forma que no se puede controlar.

En el caso de la tolerancia a fallos, esta se puede garantizar mediante la implementación de **estrategias de replicación.** La replicación de archivos es una redundancia útil para mejorar la disponibilidad de los datos. Además de distribuir la carga a la hora de leer los archivos.

1.1.2. Revisión de los principales sistemas existentes

Desde los inicios del presente siglo se han desarrollado más de una docena de productos o soluciones informáticas para el almacenamiento distribuido de datos [9–21]. Unos tienen más ventajas que otros, en dependencia del entorno y los propósitos para los cuales han sido creados. A continuación se mencionan algunos de estos sistemas, principalmente aquellos que se mantienen activos con actualizaciones y publicaciones más recientes.

Tahoe

Tahoe [22, 23] fue desarrollado para servir como soporte tecnológico del almacenamiento de *Allmydata* [24], una empresa que ofrece alojamiento de datos para usuarios comunes y empresas, con garantías de privacidad, redundancia y disponibilidad. Es un software liberado bajo licencia de código abierto y demuestra su escalabilidad mediante su puesta en práctica por *Allmydata*. Cuenta con un API que facilita el desarrollo de aplicaciones que interactúen con el sistema en el lenguaje de programación Python. Para garantizar la tolerancia a fallos implementa un mecanismo de *erasure coding* [25], que consiste en una alternativa a la replicación de datos. Está disponible para varias distribuciones de Linux y para otros Sistemas Operativos, dentro de los que está *Windows*.

Debido a que está siendo utilizado para brindar un servicio de almacenamiento *on-line* a través de internet, sus desarrolladores sacrifican el rendimiento para potenciar la encriptación, un alto precio a pagar para aquellos clientes que quieren garantizar la seguridad de sus datos. Por lo que no es recomendable su uso para el entorno universitario, teniendo en cuenta que los datos que se proponen almacenar en las estaciones de trabajo no son confidenciales. Además, encriptar los datos pudiera causar que estos no se pudieran recuperar a largo plazo en caso de perdida de claves para desencriptarlos.

GlusterFS

GlusterFS [26] es un sistema para aglutinar varios servidores de almacenamiento en un único sistema de archivos. Se puede utilizar en entornos donde cada servidor funciona como un espejo de los otros [27]. Los nodos clientes pueden acceder al sistema distribuido como si se tratara de sistema de ficheros local. Está bajo la licencia GNU, es un sistema tolerante a fallos, replica los datos para asegurar la disponibilidad y puede utilizar *hardware* de bajo costo. Sin embargo tiene escasa documentación para permitir que otras aplicaciones interactúen con el sistema.

El principal inconveniente de este sistema para el entorno universitario es que los nodos clientes, que serían principalmente las estaciones de trabajo, no almacenan los datos. Por otra parte, desplegar el módulo del servidor en todas las estaciones sería muy tedioso, ya que en el proceso de configuración del sistema hay que adicionar a todos los servidores a un grupo de confianza para que se comuniquen satisfactoriamente entre ellos.

Ceph

Uno de los más jóvenes es **Ceph** [28], un sistema de archivos distribuidos que es tolerante a fallos mediante la réplica de los datos almacenados. Se desarrolla como un proyecto de código abierto, bajo la licencia GNU/LGPL. El objetivo de Ceph [29] es proporcionar un sistema de archivos totalmente distribuido y de forma escalable, es a su vez fiable y se basa en un paradigma de almacenamiento de objetos y metadatos. Además se puede interactuar con él desde otras aplicaciones a través de los lenguajes Java, `C++` y Python. Una gran desventaja es la inmadurez de Ceph como plataforma ya que no ha sido ampliamente utilizado en un entorno de producción.

HDFS

Con el transcurso de los años el sistema que lleva por nombre *Hadoop Distributed File System* (**HDFS**) [30], desarrollado por la fundación de software Apache, se ha convertido en una de las principales herramientas de código abierto para el análisis y almacenamiento de grandes volúmenes de datos. Entre sus principales características se encuentran la detección de fallos y recuperación automática ante los mismos. Está diseñado para almacenar archivos de gran tamaño que se fragmentan

en una secuencia de bloques.

En la página oficial del proyecto *Hadoop*, que contiene al HDFS como un submódulo, se muestran los reconocimientos obtenidos por este proyecto, entre los que se destaca el primer premio de innovación de la *Media Guardian Innovation Awards*. También ha ganado el *Sort Benchmark Terabyte*, ya que ha sido utilizado para ordenar un Terabyte de datos en 209 segundos, que batió el récord anterior de 297 segundos. Este sistema ha sido usado para almacenar datos en el orden de los PB procedentes de la red social *Facebook* y de prestigiosas empresas como *Yahoo! Inc.* [21, 31]. Por su diseño, estrategia de replicación y popularidad, ha sido este sistema el seleccionado para realizar el estudio que se pretende en este trabajo, rezón por la cual en la próxima sección se muestra una descripción un poco más detallada de este sistema.

1.1.3. HDFS: Hadoop Distributed File System

HDFS [32] es un SAD para ejecutarse en cualquier variedad de *hardware*. Tiene muchas similitudes con otros SAD existentes ya mencionados, aunque algunas diferencias pueden ser significativas. La detección rápida de fallos y que se realice una recuperación automática, son los principales objetivos de la arquitectura de HDFS [30].

El sistema HDFS se desarrolló en el lenguaje de programación Java, y cualquier estación de trabajo que admita Java puede ejecutar los componentes del sistema. Un despliegue típico tiene un equipo de cómputo dedicado para que sea el servidor del sistema. Las restantes máquinas pueden ejecutar una instancia del módulo de *software* que se utiliza en los nodos que almacenan los datos. También presenta un API en Java que facilita a una aplicación, que desea almacenar y recuperar datos utilizando HDFS, interactuar de una manera elegante si está desarrollada con este lenguaje de programación. Además, brinda interfaces de programación para otros lenguajes como `C++`, Python, PHP y Ruby.

El sistema HDFS está compuesto por tres tipos de componentes: nodo cliente (*Client Node*), nodo servidor (*Name Node*) y nodos de datos (*Data Node*), tal y como se observa en la figura 1.1. A continuación se describe las principales funciones de cada uno de ellos.

Los nodos clientes son los encargados de iniciar el flujo de datos en el sistema, solicitan la

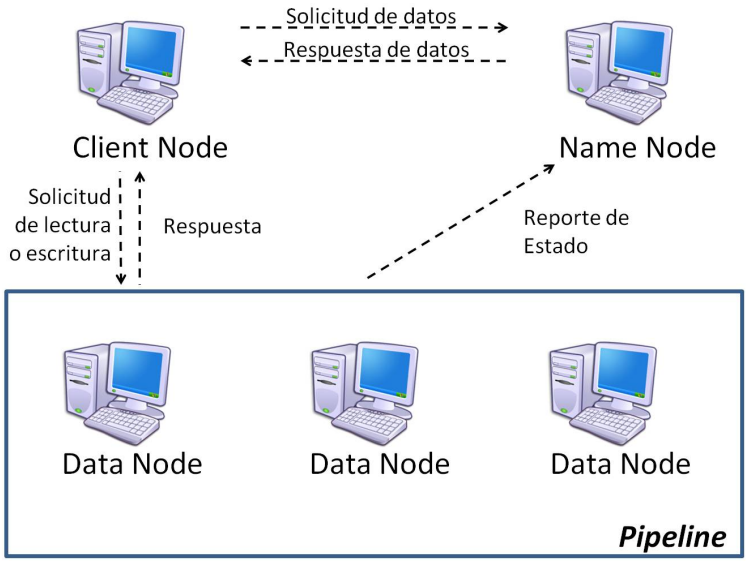


Figura 1.1: Componentes del sistema HDFS

escritura o lectura de un archivo y de esa manera comienzan a interactuar con los otros componentes. Una vez respondida la solicitud, ya sea exitosa o errónea llega al nodo cliente la respuesta. Una misma máquina, en momentos diferentes puede comportarse como nodo cliente o nodo de datos.

El nodo servidor es un registro de nombres que almacena los *metadatos* de los ficheros. Administra el sistema de archivos y gestiona la asignación de bloques de archivo a los nodos de datos para la escritura, así como la búsqueda y localización de las réplicas para la recuperación.

Los nodos de datos almacenan los bloques de archivos y tienen además la responsabilidad de reportarse al nodo servidor cada cierto tiempo para enviarle información sobre su estado. De esta manera se sabe si están disponibles y también se obtiene un reporte acerca de los bloques almacenados.

Proceso de Escritura

El proceso de escritura, a grandes rasgos, se puede describir con ocho pasos fundamentales que se muestran en la Figura 1.2. Primeramente, el nodo cliente manda a registrar el archivo a escribir, en el nodo servidor, una vez creado el registro, el archivo se divide en paquetes para los cuales se crean bloques en el nodo servidor, recibiendo para cada bloque los nodos de datos dónde serán

replicados (*pipeline*) y así comenzar la escritura de los datos. El proceso finaliza satisfactoriamente una vez almacenados todos los bloques y recibidas las respuestas de confirmación (*ACKs*).

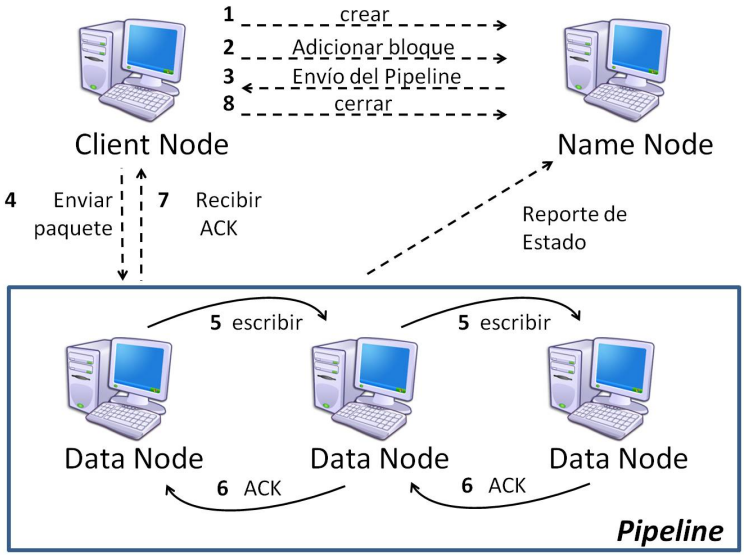


Figura 1.2: Principales acciones del proceso de escritura.

En caso de que la conexión con uno de los nodos de datos sea fallida, se elimina al nodo de datos que provocó el error del *pipeline* y se continúa el flujo de escritura con el resto de los nodos. La escritura falla finalmente si ningún nodo de datos del *pipeline* está disponible, pero si se copia el bloque en una cantidad de nodos que está por encima de determinado umbral, entonces se cierra el proceso satisfactoriamente y la estrategia de replicación implementada en el sistema se ocupa de detectar los bloques que están por debajo del factor de replicación para replicarlos nuevamente.

Proceso de Lectura

El proceso de lectura de igual manera se puede describir de forma simple como se muestra en la Figura 1.3. Para un archivo determinado que se desea leer, el nodo cliente le solicita al nodo servidor las ubicaciones de los bloques que lo conforman. Luego, el nodo servidor envía los nodos de datos donde se encuentra replicado cada bloque y el nodo cliente se conecta al primer nodo de datos para recuperar el bloque, en caso de que no se pueda conectar, lo intenta con el segundo y así sucesivamente hasta finalizar satisfactoriamente el proceso o generar un mensaje de error.

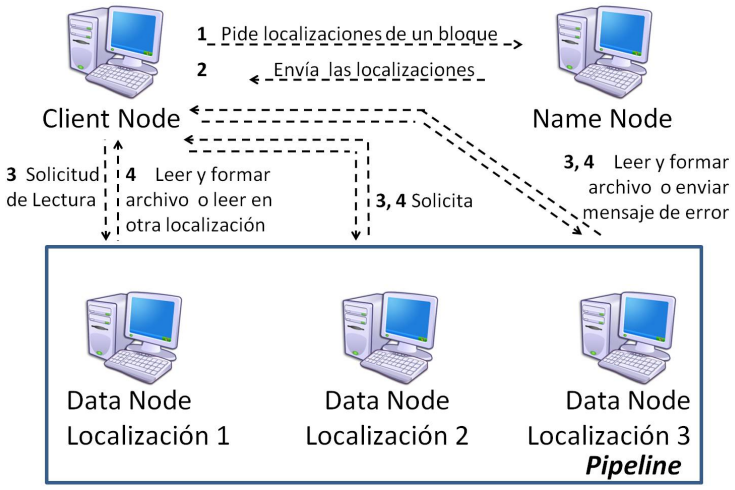


Figura 1.3: Principales acciones del proceso de lectura.

Estrategia de replicación

HDFS almacena cada archivo como una secuencia de bloques; todos los bloques son del mismo tamaño, excepto el último que puede ser de un tamaño diferente. Los bloques de un archivo se replican para asegurar la tolerancia a fallos y tanto el tamaño de los bloques como el factor de replicación son configurables en el sistema.

El nodo servidor toma todas las decisiones con respecto a la replicación de bloques teniendo en cuenta lo siguiente. Periódicamente recibe reportes (*Heart Beats*) de cada uno de los nodos de datos, como constancia de que el nodo de datos está funcionando correctamente. Estos reportes contienen una lista de todos los bloques existentes en los nodos de datos, lo que permite que se puedan detectar los bloques que están por debajo del factor de replicación.

Para distribuir las réplicas, la red se encuentra dividida en pequeñas subredes denominadas *Racks*. La colocación de las réplicas en los *Racks* es fundamental para la fiabilidad y el rendimiento, HDFS realiza esta distribución para mejorar la fiabilidad de los datos, disponibilidad y utilización del ancho de banda de la red.

La estrategia de replicación tiene en cuenta el *Rack* desde el que se solicita la escritura y selecciona a nodos del mismo *Rack* y a otros de diferentes subredes. Con esto se garantiza que en caso de que ocurra algún problema de pérdida de conexión en alguno de los *Racks*, existe una réplica

disponible en otro.

1.2. Simulación de Eventos Discretos

El proceso de simular consiste en imitar la evolución temporal de un fenómeno o de un sistema real [33]. Uno de los conceptos más utilizados es el de Robert Shannon cuando expresa que: "Simulación es el proceso de diseñar y desarrollar un modelo de un sistema o proceso real y conducir experimentos con el propósito de entender el comportamiento del sistema o evaluar varias estrategias para la operación del sistema" [34]. Este modelo se construye a partir de asunciones sobre el sistema expresadas como relaciones matemáticas, lógicas o simbólicas. Una vez desarrollado y validado, un modelo se puede usar para investigar una gran cantidad de cuestiones sobre qué ocurriría en un sistema real si se hiciese esto o aquello, o también se puede utilizar para predecir el rendimiento de sistemas no desarrollados físicamente durante las etapas de diseño.

En las últimas décadas, la rápida evolución de las tecnologías de la informática han propiciado la proliferación de nuevos sistemas dinámicos más complejos como son, entre otros, los sistemas distribuidos [35] como el que se pretende estudiar en este trabajo. Estos sistemas tienen un comportamiento que se caracteriza por una secuencia finita o infinita de estados delimitados por eventos que ocurren de manera asíncrona y se clasifican en Sistemas de Eventos Discretos (SED) [36]. Los SED pueden definirse también como un conjunto de procesos interconectados, dentro del cual la actividad fluye de un proceso a otro [37].

Un modelo para la simulación de eventos discretos [38] deberá respetar el orden en que ocurren los eventos en el sistema real, además del orden en que aparecen dentro del sistema. Los eventos y la interacción de entidades deben reflejarse en el modelo de la manera más genérica posible. La herramienta para este fin debe contar con un editor de modelos, soportar la ejecución del modelo construido y ser capaz de almacenar información sobre lo que sucede durante la simulación.

Con el empleo de la herramienta se hace necesario el modelado para hacer una aproximación al sistema real y poder observarlo, controlarlo, modificarlo y estimar medidas de rendimiento del sistema. El modelo permite además abstraernos de elementos que no aportan relevancia en el trabajo,

así como enfatizar y llevar a parámetros críticos los que si son de interés. De tal manera que se pueda, luego de varias simulaciones, llegar a conclusiones que se apliquen al sistema real.

Los elementos fundamentales en la simulación de eventos discretos son las variables y los eventos. En general, hay tres clases de variables que se utilizan con mayor frecuencia: La variable de tiempo se refiere al tiempo (simulado) que ha transcurrido, las variables de conteo son las que mantienen un conteo del número de veces que ciertos eventos han ocurrido hasta el instante y las variables de estado que caracterizan el estado del sistema en un instante determinado.

Estas variables tienen valores durante toda la simulación, pero inicialmente algunas de ellas deben tener un valor que pudiera ser generado aleatoriamente utilizando funciones estadísticas de distribución. La ventaja de inicializar valores en una simulación mediante la generación probabilística es la facilidad de emplear algoritmos matemáticos sencillos, para probar el efecto de diferentes valores de entrada. Para ellos se debe asegurar bien que la distribución probabilística utilizada sea representativa de las condiciones reales del sistema modelado. En la medida que esto ocurra los resultados serán más o menos fieles al comportamiento real del sistema.

1.2.1. Modelado del Tiempo

Para estudiar un sistema utilizando un modelo es de gran interés, en ocasiones, conocer cómo se comporta en el transcurso del tiempo. Para ello, se introduce un reloj global que es guiado por la sucesión de los eventos de la simulación. Los valores que representan el tiempo del reloj pueden ser de tipo enteros (discretos) o reales (continuos).

Cada variable puede llevar un valor de tiempo que indica el valor mínimo que debe tener el reloj del sistema para que pueda ser utilizada. Este valor de tiempo, una vez que la variable fue utilizada por un evento, puede mantenerse, aumentar, reiniciarse y volver a su valor inicial o desaparecer.

Si en algún momento el reloj global del sistema es menor que todos los valores de tiempo, es decir, si no hay ningún evento próximo a ocurrir y el sistema queda aparentemente detenido, entonces el reloj del sistema avanza de un salto hasta el valor del evento más próximo a ocurrir. No importa si sea casi inmediato o muy distante del instante actual, de esta manera continúa la corrida del modelo. El tiempo en el modelo se utiliza también para determinar cuánto se demoran

las solicitudes en ser respondidas y de esta manera evaluar el desempeño del sistema.

1.2.2. Herramientas para la simulación de sistemas distribuidos

Existen numerosas herramientas computacionales desarrolladas para el modelado y simulación de todo tipo de sistemas [39]. En particular, resultan de interés para este trabajo aquellas que facilitan modelar y simular sistemas distribuidos.

CloudSim

CloudSim [40] es un proyecto que propicia la simulación y experimentación de las nuevas infraestructuras de computación en la nube (*Cloud Computing* en Inglés). Mediante el uso de CloudSim, investigadores y desarrolladores pueden centrarse en aspectos específicos de diseño de sistemas para investigar sin preocuparse por los detalles de bajo nivel relacionados con la nube como son infraestructuras y servicios [41,42]. Está desarrollado en Java como lenguaje de programación y se necesita conocimiento de este lenguaje para interactuar con la herramienta.

MONARC2

MONARC2 [43] es una extensión del proyecto inicial MONARC, muy utilizada para el diseño y simulación de sistemas distribuidos a gran escala [44–46], con interacciones complejas que necesitan intercambiar y procesar grandes cantidades de datos. Además, está orientado al proceso de simulación de eventos discretos, es muy adecuado para describir los programas que se ejecutan concurrentemente. El objetivo principal del proyecto MONARC es proporcionar una simulación realista de los grandes sistemas de computación distribuida y ofrecer un entorno flexible y dinámico para evaluar el rendimiento de una amplia gama de procesamiento de datos [47–49]. Este proyecto de simulación se basa en la tecnología Java para el desarrollo de un proceso de simulación.

SimGrid

SimGrid es otra de las herramientas que no se puede dejar de mencionar. Es utilizada para la simulación de sistemas distribuidos teniendo en cuenta el control del ancho de banda y la redistri-

bución de archivos [50]. Además es utilizada para el estudio de simulaciones basadas en clúster y en la evaluación de una red y algoritmos *Point to Point* (P2P) [51]. Entre las características de la herramienta se encuentran: un motor de simulación escalable y extensible, permite simular redes de diferentes topologías, disponibilidad de los recursos de red, así como la tolerancia a fallos. Cuenta con un alto nivel de interfaces de usuario para la simulación con prototipos en C y en Java. [52–54].

GridSim

GridSim [55–57] es una herramienta de simulación basada en el simulador de eventos discretos SimJava, que como su nombre lo indica permite el modelado y la simulación de las entidades en la computación paralela y distribuida en el lenguaje de programación Java. Ofrece un servicio integral para la creación de diferentes clases de recursos heterogéneos que pueden ser agregados al modelo. Un recurso dentro del modelo puede ser, por ejemplo, un solo procesador o multiprocesador con memoria compartida o distribuida.

OptorSim

OptorSim es una herramienta útil para evaluar varios algoritmos de optimización en un entorno de computación Grid, donde cada sitio puede contener almacenamiento o varios elementos de cálculo. OptorSim permite la entrada de una configuración de red y luego ejecuta una serie de algoritmos utilizando la configuración dada, permitiendo al usuario visualizar el rendimiento de los algoritmos. [58, 59]. Al igual que los anteriores se basa en el lenguaje de programación Java.

Las herramientas de simulación mencionadas anteriormente están disponibles en la Web bajo licencia GNU/LGPL y pudieran ser utilizadas para el trabajo que se pretende desarrollar en esta investigación. Sin embargo, el API que se brinda, en algunos casos está diseñado para simular entornos Grid o de computación en la nube; pero no para simular redes locales con estaciones de trabajo dentro de una institución. Por otra parte los lenguaje Java y `C++`, que se utilizan para el modelado y especificación de los sistemas, tienen muchas potencialidades como lenguajes de programación de propósito general; pero son menos explícitos y apropiados que otros lenguajes de modelado como el que se presentará en la próxima sección.

1.2.3. Redes de Petri Coloreadas como lenguaje de modelado

Existen tantos lenguajes diferentes para el modelado de sistemas, que sería muy difícil y requiere mucho tiempo hacer una comparación explícita con todos ellos. En su lugar, esta sección dejará explícitamente algunas de las propiedades que hacen que las Redes de Petri Coloreadas (RdPC) sean un lenguaje de modelado valioso para el diseño, especificación y análisis de sistemas concurrentes y distribuidos [60].

Las RdPC fueron desarrolladas en la década de los 80 por Kurt Jensen, profesor y jefe del Departamento de Ciencias de la Computación en la Universidad de Aarhus, Dinamarca [61]. Consisten en un formalismo para el modelado analítico de sistemas que presentan concurrencia, sincronización y recursos compartidos [62, 63]. Permiten representar gráficamente comportamientos de sistemas simples o complejos que facilitan a los analistas tomar decisiones oportunas y eficientes sobre el sistema modelado [64]. Los elementos de modelado que se encuentran en una RdPC y cómo utilizarlos se encuentran en la bibliografía con manuales bien explicados que incluyen numerosos ejemplos [65].

Las Redes de Petri se han utilizado en trabajos [66–68] que muestran el ciclo completo por el que pasa una tarea en un sistema distribuido para aumentar cómputo, especificando parámetros como el número de computadoras, cantidad de tareas a distribuir, y cuidando otros parámetros como la latencia y el ancho de banda.

¿Por qué utilizar las Redes de Petri Coloreadas?

No es bueno generalizar ni sería lógico decir que las RdPC son superiores a todos los demás lenguajes de modelado. Sin embargo, para algunos propósitos las RdPC son muy útiles, y junto con otros lenguajes de programación son muy ventajosas para el análisis y diseño de sistemas. A continuación se mencionan algunas de sus ventajas.

1. Tienen una representación gráfica muy atractiva. Es muy fácil de entender y comprender incluso para las personas que no están muy familiarizados con los detalles de este lenguaje. Esto se debe a que los diagramas se parecen a muchos de los dibujos informales que los diseñadores e ingenieros hacen mientras construyen y analizan un sistema.

2. Tienen una semántica bien definida sin ambigüedad que define el comportamiento de las redes.

3. Son de propósito general y pueden ser usadas para describir una gran variedad de sistemas, entre los que se encuentran los sistemas distribuidos.

4. La concurrencia se puede definir de una manera muy natural y sencilla.

5. Ofrecen descripciones jerárquicas. Se puede construir una gran red relacionando otras redes entre sí, de una manera bien definida. Las construcciones jerárquicas hacen posible modelar sistemas muy grandes de una forma modular y manejable.

6. Se pueden ampliar con un concepto de tiempo mediante un reloj global y permitir que cada símbolo (*token*) lleve un sello de tiempo, además del valor de los datos que ya tiene. Intuitivamente, la marca de tiempo de un *token* especifica la hora a la que el *token* está listo para ser utilizado, es decir, consumido por una transición.

7. Ofrecen simulaciones interactivas donde los resultados se presentan directamente en el diagrama. De tal manera que los valores de los datos que tienen los *tokens* en movimiento pueden ser inspeccionados y esto facilita la depuración del modelo que se está construyendo.

8. Disponen de herramientas informáticas de apoyo para el modelado y simulación. La existencia de estas herramientas informáticas, es extremadamente importantes para el uso práctico de las RdPC.

1.2.4. Modelado y simulación con Redes de Petri Coloreadas

En esta sección se presenta una de las principales herramientas informáticas y un lenguaje de programación que pueden ser utilizados para el modelado y simulación de sistemas con las RdPC.

CPN Tools

En el Departamento de Computación de la Universidad de Aarhus, bajo la guía de Kurt Jensen, nace CPN Tools [69], una herramienta gráfica para la edición, simulación y análisis de sistemas modelados utilizando el lenguaje RdPC [70, 71]. Dispone de un simulador (tanto interactivo como

automático) para poder inspeccionar y depurar el sistema modelado. Se encuentra bien documentado y pueden consultarse manuales en la web [72, 73].

En 1998 la Universidad de Munich realizó un estudio comparativo de varios *softwares* existentes para el análisis y simulación de Redes de Petri. Ese estudio [74] incluyó 91 paquetes y determinó que entre los paquetes tanto comerciales como académicos, el que mejor puntaje obtuvo a través de todas las pruebas realizadas fue el llamado Design/CPN. Años después, en el 2007, en un trabajo similar [75] se recomienda la herramienta CPN Tools, sucesora de Design/CPN, debido a que una de las mayores ventajas que presenta esta herramienta es el lenguaje de modelado que utiliza, las RdPC, ya abordado en la sección anterior.

Elementos gráficos

En CPN Tools las descripciones de los modelos de las RdPC constituyen una combinación de un lenguaje gráfico y el lenguaje de programación CPN ML. El lenguaje gráfico es un grafo dirigido que consta de dos tipos de vértices: los nodos lugares dibujados como círculos u óvalos y transiciones dibujadas en forma de cuadrados o rectángulos. Las aristas del grafo son los arcos que conectan los nodos lugares (o sencillamente lugares) y transiciones. Además de estos elementos gráficos se considera importante mencionar los *tokens*, valores que se encuentran dentro de los lugares y que se mueven de un lugar a otro como resultado del disparo o ejecución de una transición.

En la herramienta CPN Tools se adicionan cada uno de los elementos de una RdPC utilizando las paletas que brinda la aplicación. Cada elemento tiene sus atributos descritos en el lenguaje CPN ML y se muestran en la figura 1.4

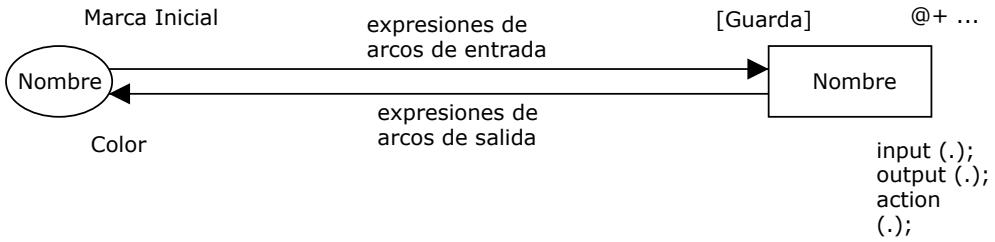


Figura 1.4: Nodo lugar y transición.

Lo primero que aparece en la figura (de izquierda a derecha) es un lugar, y puede tener tres

inscripciones asociadas: una obligatoria que es el color o tipo de dato y dos opcionales, el nombre y la marca inicial que es un valor cuyo tipo corresponde al color que tiene el lugar y está formada por un número inicial de *tokens*. Luego aparecen dos arcos representadas por saetas, de entrada y de salida con respecto a la transición, que cuentan con inscripciones formadas por expresiones en el lenguaje de programación CPN ML. El conjunto de colores que forma parte de la expresión de cada arco, tiene que coincidir con el color del lugar fijado al arco. De no ser así, aparecerá un mensaje de error, cerca del arco, durante el chequeo de la sintaxis que realiza la herramienta CPN Tools. Las transiciones pueden contar con cuatro inscripciones y todas opcionales: el nombre, la guarda, la expresión de tiempo y puede tener además un segmento de código asociado.

El nombre de una transición, por lo general, indica una acción a realizar con los *tokens* presentes en el lugar asociado a la transición por un arco de entrada. La guarda es una expresión booleana (en el lenguaje CPN ML) que se evalúa como verdadera o falsa antes de dispararse la transición. La transición se activa sólo en el caso de la guardia sea verdadera y esta guarda restringe la elección de *tokens* de entrada a la transición. Además, la guarda permite la comparación de los valores de los *tokens* de diferentes lugares mediante la combinación de expresiones.

Una expresión de tiempo asociada a una transición debe ser entera, positiva y tiene que estar precedida por @+, finalmente tiene la forma "@+ expresión de tiempo". Aunque este valor se plasma en la transición, indica el tiempo del modelo en el que pueden ser consumidos los *tokens* que están en el lugar asociado a la transición por un arco de entrada. Esta expresión modifica el valor de tiempo de los *tokens* consumidos y la ausencia de la expresión es equivalente a un tiempo cero.

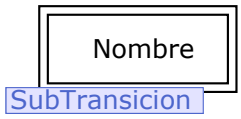


Figura 1.5: Subtransición.

Existe otro elemento (figura 1.5) que aparecerá frecuentemente a lo largo de las explicaciones del modelo. Se trata de las transiciones de sustitución o simplemente sub-transiciones. Éstas se representan con un cuadrado o rectángulo con doble contorno y se utilizan para realizar una jerarquía de redes. En la programación de un programa se enlazan módulos, procedimientos, funciones. De

manera similar un modelo jerárquico en la herramienta CPN Tools indica que existe una red en un nivel más bajo de la jerarquía y la sub-transición la representa en el nivel superior.

CPN ML

La herramienta CPN Tools permite enriquecer la lógica del modelado con el lenguaje CPN ML, que se obtiene mediante la ampliación del ML estándar. CPN Tools usa el lenguaje CPN ML para las declaraciones y las inscripciones que pueden ser: declaraciones de conjuntos de colores (tipos de datos), variables, funciones y valores (constantes).Para las personas que no están familiarizadas con el ML estándar esta ampliación facilita su uso ya que [76]:

1. Está adaptado para permitir que se defina el conjunto de colores a utilizar en el modelo.

2. Permite al usuario introducir variables de tipo, como parte de las declaraciones, ML estándar no tiene declaraciones de variables. Cada nodo lugar de la RdPC debe tener un tipo o color definido y puede contener elementos que pertenezcan al conjunto color especificado.

3. Las variables y las funciones se utilizan en las inscripciones de las transiciones y arcos. Existen declaraciones predeterminadas de tipos de datos básicos y el usuario puede añadir lo nuevo que necesita mediante el menú circular que brinda la herramienta CPN Tools.

Gnuplot

Para presentar los resultados de las simulaciones, la herramienta CPN Tools genera los datos para ser visualizadas con *Gnuplot*, un *software* que se distribuye bajo una licencia de software libre y permite hacer gráficas en 2D y en 3D. *Gnuplot* tiene además una amplia variedad de opciones de salidas para que el usuario pueda generar las gráficas resultantes como lo desee, ya sea para visualizarlas o para incluirlas en sus propios documentos. En el sitio oficial de este software existe amplia documentación y ejemplos para su uso [77–79].

Empleo de monitores en la simulación

Al simular con las RdPC es útil examinar las marcas que se producen en los nodos lugares del modelo, para extraer periódicamente la información generada y luego utilizarla para diferentes propósitos, tales como:

- Detener la simulación cuando un determinado lugar está vacío o tiene un determinado valor.
- Contar el número de veces que se produce una transición.
- Actualizar un archivo de texto cuando se produce una transición para registrar valores específicos.
- Realizar análisis estadísticos con los datos recopilados en los archivos de texto.

A pesar de que la información extraída puede usarse para fines diferentes, la forma en que se extrae la información es a menudo muy similar. Esta información se puede recoger en la herramienta CPN Tools gracias a la existencia de los monitores. Un monitor es un mecanismo que se utiliza para observar, inspeccionar, controlar o modificar una simulación. Muchos monitores diferentes pueden definirse para un modelo, estos pueden inspeccionar tanto las marcas de los nodos lugares como las transiciones durante una simulación. En CPN Tools existen cuatro tipos de monitores: los de punto de interrupción, los recopiladores de datos, los que escriben en archivos y los definidos por el usuario.

Los monitores de puntos de interrupción se utilizan para detener la simulación cuando se cumplen determinadas condiciones. Los monitores recopiladores de datos se utilizan para extraer los datos numéricos durante una simulación, que luego servirán para realizar cálculos estadísticos, y que pueden ser guardados en archivos de registro. Los archivos de registro pueden entonces ser post-procesado, por ejemplo, mediante la importación en programas de hojas de cálculo o ser mostrados en gráficos. Por otra parte los monitores definidos por el usuario pueden ser implementados en el lenguaje CPN ML para que sean utilizados con un propósito que no esté cubierto por los otros tipos de monitores. Todos estos monitores pueden ser desactivados o activados a conveniencia del usuario en cada simulación para recopilar los datos de interés.

1.3. Conclusiones parciales

Los SAD existentes y revisados en la literatura han sido ampliamente utilizados en entornos dedicados, no siendo así en aquellos donde no se tiene el control de la disponibilidad de las estaciones de trabajo que formarán parte del sistema. Por este motivo sería muy arriesgado desplegar un sistema de esta envergadura sin realizar un estudio previo para analizar su futuro comportamiento. De hacerlo se podría, entre otras consecuencias, perder definitivamente los datos almacenados y sobrecargar innecesariamente la red.

Después del análisis de diferentes soluciones existentes para desplegar un SAD y adaptarlo al entorno UCI, se seleccionó *Hadoop Distributed File System*. Entre las características principales que se tuvieron en cuenta para la selección de este sistema se encuentran: es un producto de código abierto, desarrollado en Java, brinda un API que permite a las aplicaciones interactuar con el sistema, es tolerante a fallas, presenta una amplia documentación y tiene gran prestigio internacional.

Se hace necesario un modelado para hacer una aproximación al sistema real y poder observarlo, controlarlo, modificarlo, abstraerse de los elementos que no aportan relevancia en el trabajo, así como enfatizar y llevar a parámetros críticos los que si son de interés. De tal manera que se pueda, luego de varias simulaciones, realizar corridas con el propósito de entender el comportamiento del sistema y evaluar varias estrategias para llegar a conclusiones que se apliquen al sistema real.

Las RdPC como lenguaje de modelado son de gran utilidad a la hora de realizar un modelo. Brindan facilidades en cuanto a: representación gráfica, semántica, documentación, simulaciones interactivas y la existencia de herramientas informáticas para desarrollar modelos.

La herramienta gráfica CPN Tools presenta, entre otras ventajas, un potente editor gráfico con facilidades para la edición y depuración del modelo, así como de la simulación. Además el lenguaje CPN ML es muy útil para enriquecer la lógica del modelo, adicionando declaraciones de tipos de datos, variables y funciones que son invocadas desde las inscripciones de los arcos y transiciones.

Capítulo 2

Modelado del Sistema de Almacenamiento Distribuido

En este capítulo se presenta el modelo construido para desarrollar la presente investigación. Primeramente se describe la estructura jerárquica que tiene dicho modelo, luego los componentes que lo forman y las interacciones entre ellos. También se muestra un estudio realizado al entorno donde se quiere desplegar el sistema y a partir de ahí se determinan parámetros que condicionan su funcionamiento durante la simulación.

2.1. Estructura jerárquica del modelo

El modelo cuenta con 67 páginas estructuradas jerárquicamente en las que están recogidas tanto la lógica del SAD, como las características del entorno no dedicado donde se quiere desplegar el sistema. La estructura jerárquica del modelo ha sido diseñada analizando la arquitectura y el código fuente de HDFS. La figura 2.1 muestra la página base de la jerarquía con los tres componentes principales del modelo: *Client Node*, *Name Node* y *Data Node*. Estos componentes son modelados como sub-transiciones y por cada una se puede navegar internamente para inspeccionar el modelo.

Estos componentes comparten un lugar común dentro del modelo, al que se llamó *Network* ya que representa la red que los conecta físicamente en un entorno real. Siguiendo esta representación de

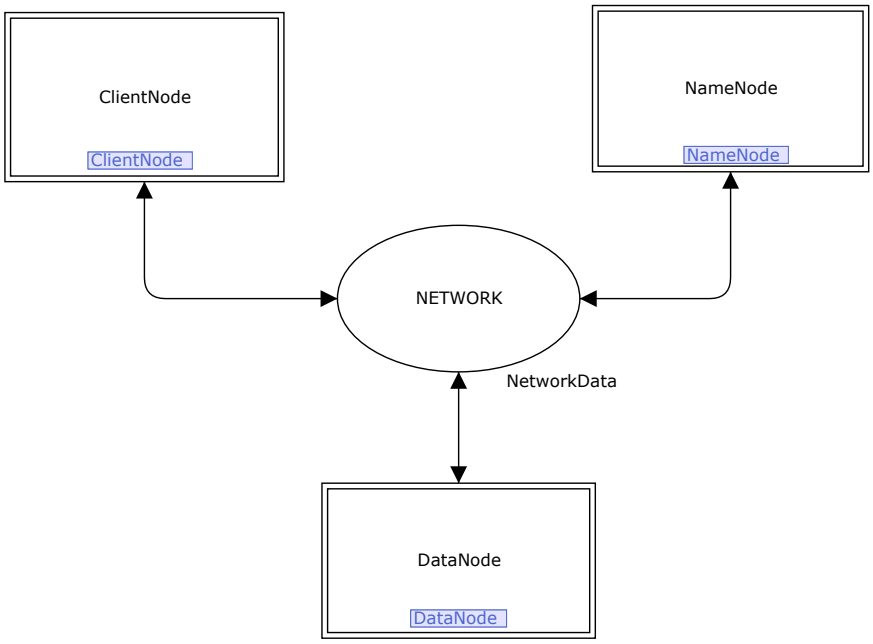


Figura 2.1: Componentes principales del SAD.

la red, todo lo que sale de cualquiera de los componentes llega a este lugar y luego cada componente identifica qué le corresponde de lo que ahí se encuentra. Las páginas principales que desglosan el funcionamiento de cada uno de los componentes, es decir el segundo nivel de esta jerarquía, se explicarán en las secciones siguientes.

El color *NetworkData* correspondiente al lugar *Network* representa los mensajes intercambiados a través de la red y se modela como:

```
colset DataCenter = index center with 0..pNumOfDataCenter;
colset Rack = index rack with 0..pNumOfRack;
colset Host = index host with 0.. pNumOfHostsPerRack;
colset MachineID = product DataCenter * Rack * Host;
colset NodeType = with CNode | DNode | NNode;
colset Source = product MachineID * NodeType;
colset Destiny = product MachineID * NodeType;
colset OpType = with create| addBlock| connect| abandonBlock
                    | complete| write| read| register
                    | setHeartBeat| reportBlocks| getBlocks;
colset Operation = product OpType * INT;
colset Args = list STRING;
colset Answer = list STRING;
```

```
colset NetworkData = product Source * Destiny * Operation
                             * Args * Answer timed;
```

Nótese que *pNumOfDataCenter*, *pNumOfRack* y *pNumOfHostsPerRack* son parámetros para la especificación del número de centros de datos, cantidad de *Racks* y cantidad de estaciones de trabajos por *Racks*, respectivamente. La identificación de cada equipo (*MachineID*) es un producto cartesiano ternario que modela las ubicaciones físicas de cada estación de trabajo en el clúster. El conjunto de colores *OpType* denota las posibles operaciones que se envían desde un nodo a otro. La definición de estas operaciones se basa en las interfaces de Java *ClientProtocol*, *DataTransferProtocol* y *DatanodeProtocol* que regulan la comunicación entre los tres componentes principales del HDFS. Por último, el color *NetworkData* contiene las operaciones y los argumentos que procesa el nodo destino y la respuesta devuelta al nodo origen.

2.2. Sub-transición *Client Node*

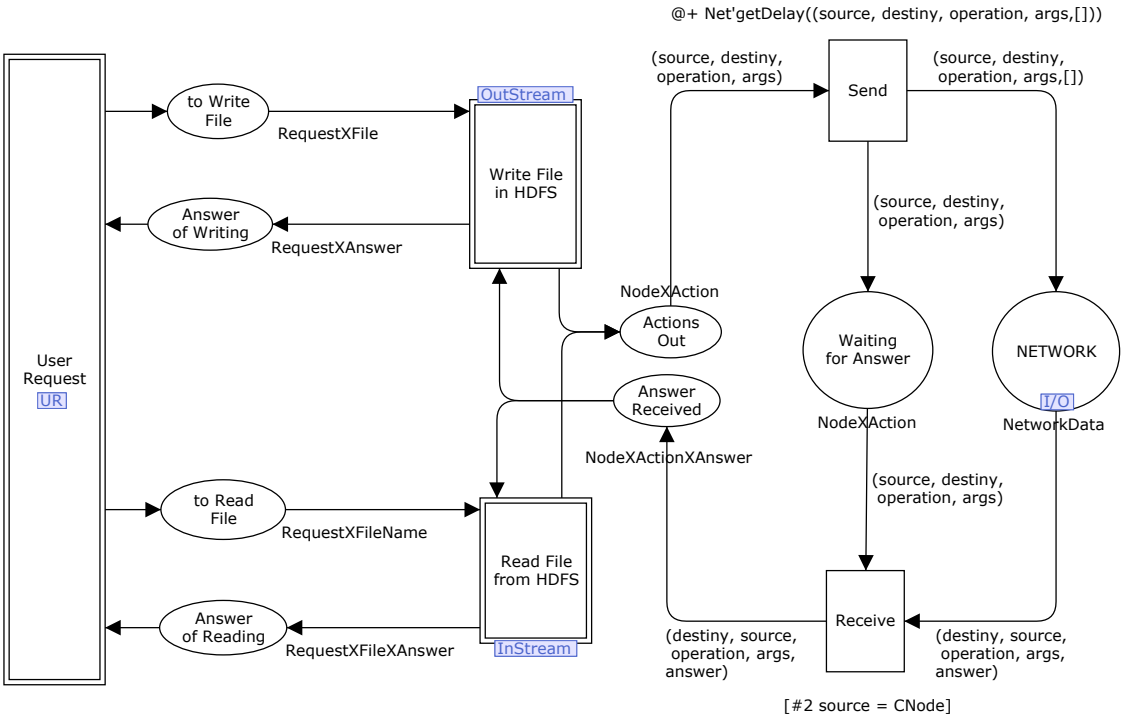


Figura 2.2: Modelado de la sub-transición *Client Node*.

La sub-transición *Client Node* (figura 2.1) tiene implementada la lógica de funcionamiento del nodo cliente que realiza solicitudes de escritura y lectura.

Como se puede apreciar en la figura 2.2, para el modelado de este nodo se tuvieron en cuenta tres sub-transiciones, que modelan los procesos fundamentales, y en las sub-secciones que aparecen a continuación se presenta una descripción más detalla de cada una de ellas.

2.2.1. Sub-transición *User Request*

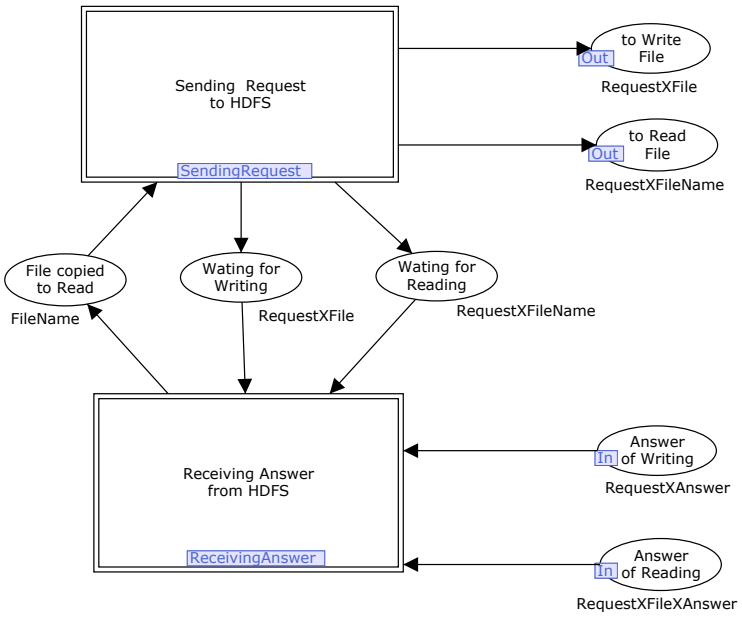


Figura 2.3: Modelado de la sub-transición *User Request*.

En la sub-transición *User Request* (figura 2.2) se generan las solicitudes de escritura o lectura de archivos como muestra la figura 2.3. La lógica implementada se modela teniendo en cuenta a su vez dos sub-transiciones: una de ellas genera y envía las solicitudes (*Sending Request to HDFS*) y la otra recibe las respuestas y las procesa (*Receiving Answer from HDFS*). A continuación se describen los colores utilizados.

```
colset FileName = union fileID:INT;
colset FileSizeMB = int with pMinFileSizeMB .. pMaxFileSizeMB;
colset File = product FileName *  FileSizeMB;
colset RequestType = with Write| Read;
```

```
colset Request = product MachineID * RequestType * INT;
colset RequestXFile = product Request * File timed;
colset RequestXFileName = product Request * FileName timed;
colset Answer = list STRING;
colset RequestXFileXAnswer = product Request * FileName * INT * Answer;
colset RequestXAnswer = product Request * FileName * Answer;
```

Un archivo es modelado mediante el color *File* como una tupla compuesta por un nombre y el tamaño en *Megabyte*, valor que se genera aleatoriamente entre dos números especificados como parámetros. En el caso de una solicitud (*Request*), se representa como una terna compuesta por el identificador de la máquina que hace la solicitud, el tipo de solicitud y un entero que representa la hora (tiempo del modelo) en que se realizó dicha solicitud. Para las solicitudes de escritura se utiliza el color *RequestXFile* especificando el *Request* y el *File*. Mientras que para representar las solicitudes de lectura el color más idóneo es el *RequestXFileName*, ya que solo se necesita el nombre para leer el archivo.

En la sub-transición *Sending Request to HDFS* (figura 2.3) se modela el comportamiento de la llegada de las solicitudes al sistema como se muestra en el anexo A.1. Para generar estas solicitudes se utilizó la función de distribución aleatoria *exponential(r:real): real*, que marca los tiempos en los que estas llegan al sistema. Las solicitudes enviadas son guardadas en los nodos *Wating for Writing* y *Wating for Reading* y así se pueden conocer las solicitudes pendientes por recibir respuesta de escritura y lectura respectivamente.

Por otra parte, la sub-transición *Receiving Answer from HDFS* (figura 2.3) modela la llegada de las respuestas a las solicitudes como se muestra en el anexo A.2. Una vez que los archivos son almacenados satisfactoriamente en el sistema distribuido, se guardan en el nodo lugar *File copied to Read* y solo entonces estos archivos pueden formar parte de la solicitud de lectura.

2.2.2. Sub-transición *Write File in HDFS*

En la sub-transición *Write File in HDFS* (figura 2.2) se manejan las solicitudes de escritura como muestra la figura 2.4.

Lo primero que ocurre en el proceso de escritura es el envío de los datos necesarios para crear

una entrada en el sistema de fichero del nodo servidor, esto ocurre dentro de la sub-transición *Create File in NameNode*. Si llega una respuesta satisfactoria entonces ocurre una inicialización de datos necesarios, sub-transición *Initialize idata*, y comienza la división del archivo en paquetes en la sub-transición *Split File in to packets*. Esta fragmentación del archivo es con el fin de aprovechar mejor el ancho de banda y el espacio disponible en los nodos de datos.

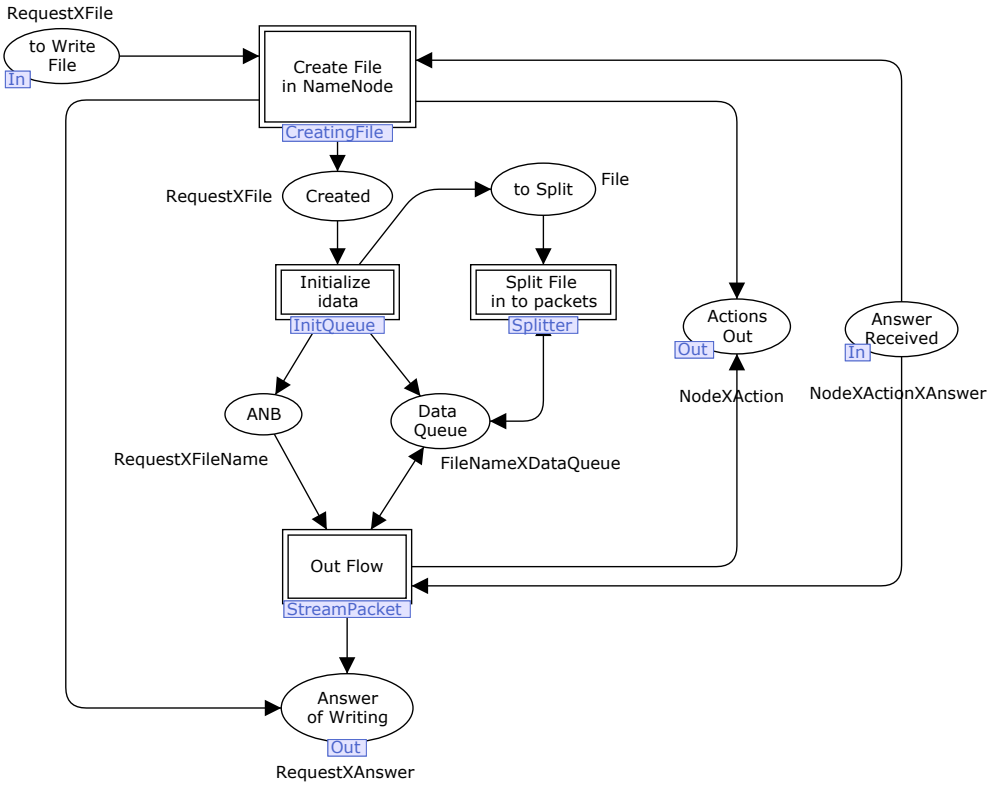


Figura 2.4: Modelado de la sub-transición *Write File in HDFS*.

Los paquetes creados producto de la división del archivo se van colocando en una cola interna llamada *Data Queue*. Esta cola es consumida por un flujo implementado en la sub-transición *Out Flow* de la figura 2.4, que se encarga de realizar el almacenamiento y replicación de cada uno de los paquetes del archivo y se muestra en la figura 2.5.

La primera sub-transición que aparece en el flujo *Out Flow* es *Allocate New Block* (figura 2.5), que tiene implementada la lógica para solicitar al nodo servidor la creación de un bloque en el sistema de fichero para cada paquete de la cola. El bloque retornado para cada paquete viene acompañado

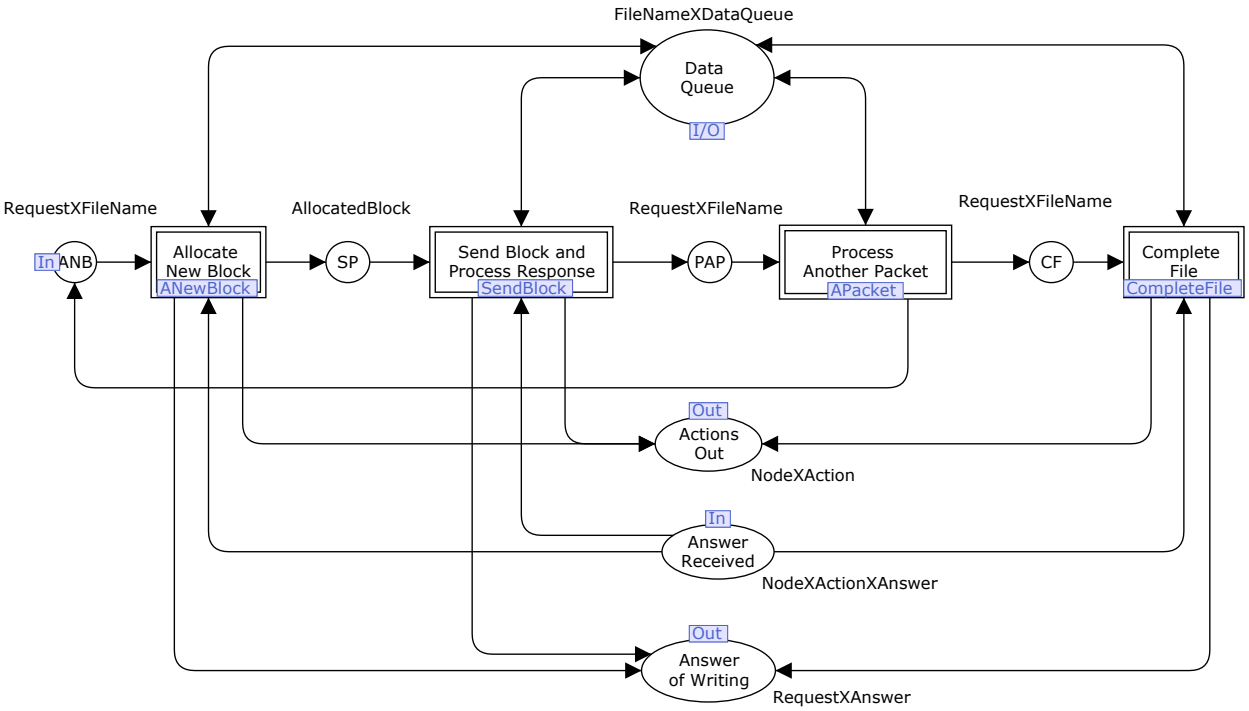


Figura 2.5: Modelado de la sub-transición *Out Flow*

de la lista de nodos de datos que forman una tubería (*pipeline*) para que se copie una réplica de los datos del bloque en cada uno de ellos. El *pipeline* retornado se encuentra ordenado de tal manera que se minimiza la distancia del camino a recorrer a partir del nodo cliente pasando por todos los nodos de datos de la tubería.

Un aspecto importante que se valida en este paso del flujo, consiste en verificar que exista una conexión entre el nodo cliente y todo el *pipeline*. En caso de que el *pipeline* este desconectado por algún nodo, se abandona el bloque recibido y se le solicita al servidor un nuevo bloque que vuelve a pasar por el mismo proceso de validación. Esto se realiza para dar una garantía de éxito total en el proceso de escritura. No obstante, para evitar caer en ciclos infinitos esta validación se realiza un número finito de veces y si no se logra obtener un *pipeline* completamente conectado entonces se reporta un error.

Para comprender el flujo por el que pasan los paquetes en la operación de replicar el bloque, es necesario volver a observar la figura 2.5. La segunda sub-transición *Send Block and Process Response* comienza el envío de datos para el primer paquete de la cola *Data Queue* y, al procesar

la respuesta, se eliminan del *pipeline* el o los nodos de datos que fallaron en el momento de la escritura e interrumpieron el almacenamiento para intentarlo tantas veces como sea necesario. Si el proceso de escritura falla para todos los nodos de datos del *pipeline* entonces se envía un mensaje de error informando que el archivo no puede ser almacenado. En el caso de que el bloque se almacene satisfactoriamente en al menos un nodo de datos entonces se procede a escribir el próximo paquete, para ello la sub-transición *Process Another Packet* elimina al paquete de la cola y habilita la sub-transición *Allocate New Block*.

Cuanto todos los paquetes que forman un archivo han sido almacenados y replicados, se invoca a la sub-transición *Complete File* que envía un mensaje al nodo servidor indicando que se completó de escribir el archivo.

Al concluir el flujo principal por el que pasan los paquetes en el nodo cliente durante el proceso de almacenamiento, en el lugar *Answer of Writing* (figura 2.4) se reciben las respuestas del proceso de escritura, ya sean satisfactorias o no.

2.2.3. Sub-transición *Read File from HDFS*

En la sub-transición *Read file from HDFS* (figura 2.2) se implementa el proceso de lectura en el nodo cliente, como muestra la figura 2.6, teniendo en cuenta tres sub-transiciones fundamentales: *Locate Block*, *Retrieve and Process Response* y *Process Another Block or Finish*. Las cuales se describen a continuación.

El proceso de lectura en el nodo cliente comienza en la sub-transición *Locate Block*, donde se solicita al nodo servidor la recuperación de los bloques de un archivo que fue almacenado con anterioridad. Una vez que se comprueba que el archivo existe en el sistema de fichero del nodo servidor, el nodo cliente recibe el identificador de los bloques y la lista de los nodos de datos que contienen cada una de las réplicas de los mismos. Esta lista se retorna de una forma ordenada teniendo en cuenta la cercanía al nodo cliente que hace la solicitud para un mejor rendimiento del sistema.

La operación de recuperar un bloque se realiza en la sub-transición *Retrieve and Process Response*, donde se intenta leer los datos del bloque estableciendo una conexión con el primer nodo de

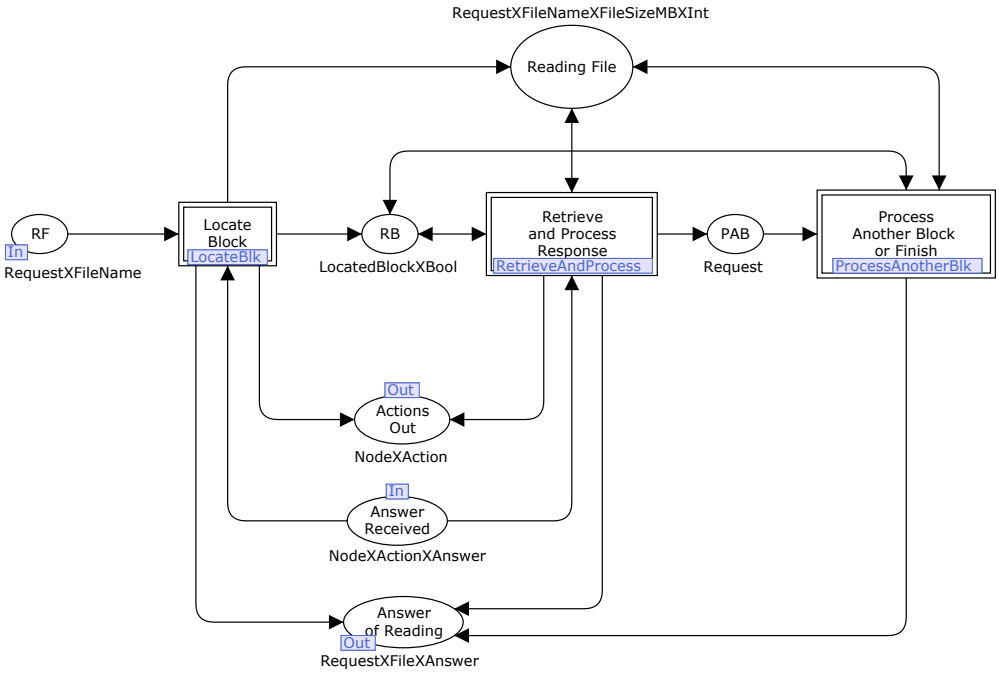


Figura 2.6: Modelado de la sub-transición *Read file from HDFS*.

datos de la lista, si no se lograr recuperar el bloque entonces con el segundo y así sucesivamente. Si el bloque no se puede recuperar de ninguna de las localizaciones, entonces se envía un mensaje de error indicando que el archivo no puede ser recuperado en ese momento.

En el caso de que el bloque sea leído satisfactoriamente entonces se procede a leer el próximo bloque, para ello la sub-transición *Process Another Block or Finish* elimina al bloque de la lista pendiente a procesar y habilita la sub-transición *Retrieve and Process Response*. Cuando todos los bloques que forman un archivo han sido recuperados entonces el proceso de lectura termina satisfactoriamente. La respuesta sea positiva o negativa, llega al lugar *Answer of Reading*.

2.3. Sub-transición *Name Node*

La sub-transición *Name Node* (figura 2.1) tiene implementada la lógica del funcionamiento del nodo servidor del sistema. Como se puede apreciar en la figura 2.7, para el modelado de este nodo se tuvieron en cuenta cuatro sub-transiciones que son fundamentales.

La sub-transición *Client Node Communication*, procesa las acciones que envía el nodo cliente

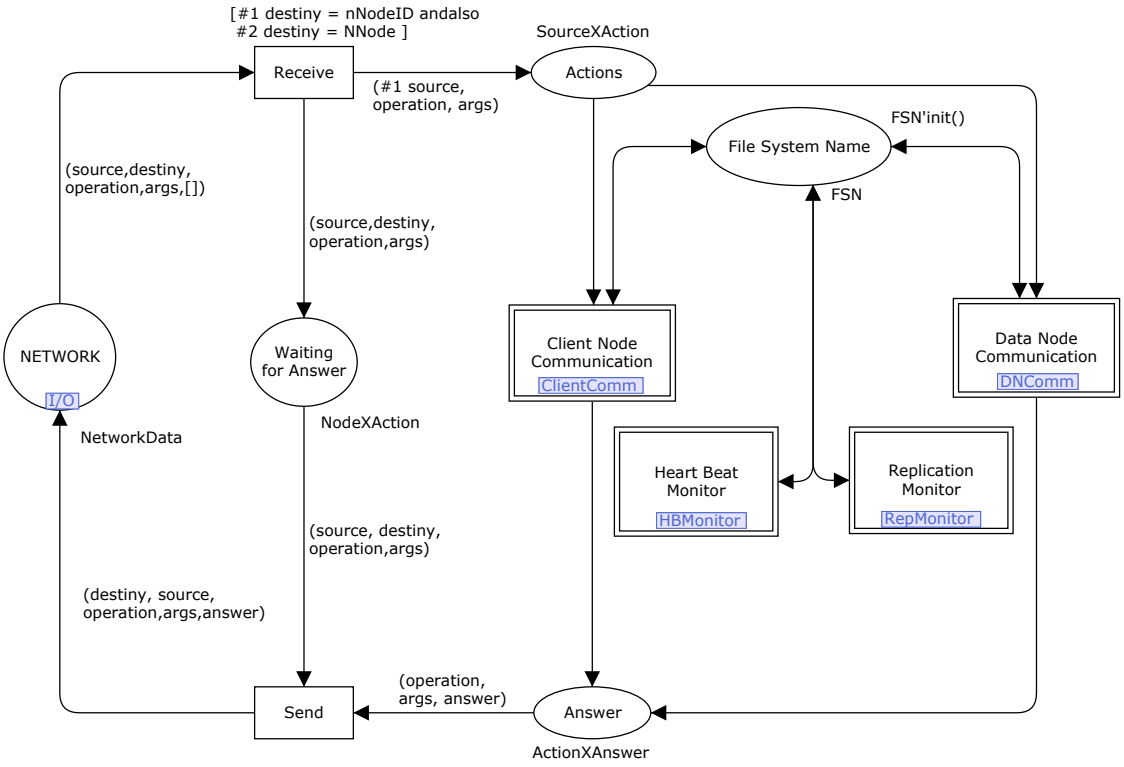


Figura 2.7: Modelado del Nodo Servidor.

mientras que *Data Node Communication* se encarga de las acciones enviadas por los nodos de datos. Las otras dos sub-transiciones *Heart Beat Monitor* y *Replication Monitor* se encargan de monitorear el estado de los nodos de datos y del factor de replicación de los bloques respectivamente. En las sub-secciones siguientes se presenta una descripción más detalla de cada una de ellas, no sin antes explicar el modelado del sistema de fichero mediante el lugar *File System Name* y el color *FSN*.

2.3.1. Modelado del sistema de fichero

El sistema de fichero ha sido modelado como una estructura de datos, que radica en el lugar *File System Name* (figura 2.7), definida mediante el color *FSN* de la siguiente forma:

```
colset FileInfo = record name:FileName * fSize:INT
                        * isOpen:BOOL * blocks:BlockList;
colset FileInfoList = list FileInfo;
colset BlockID = union blk:INT;
colset BlockSizeMB = int with 0..pBlockSizeMB;
```

```
colset Block = product BlockID * BlockSizeMB;
colset BlockList = list Block;
colset BlockInfo = record block:Block * fileName:FileName
                          * sourceDNs:DataNodeIDList
                          * scheduledDNs:DataNodeIDList ;
colset BlockInfoList = list BlockInfo;
colset BlockXTargets = product Block * DataNodeIDList;
colset BlocksToReplicate = list BlockXTargets ;
colset HeartBeat = product INT * INT * INT
                          (*freeSpace, usedSpace, threadLoad*);
colset DataNodeInfo = record dnID:DataNodeID * heartBeat:HeartBeat
                             * lastUpdate:INT * nBlkScheduled:INT
                             * storedBlk:BlockList
                             * blksToReplicate:BlocksToReplicate
                             * blksToInvalidate:BlockList;
colset DataNodeInfoList = list DataNodeInfo;
colset DataNodeMap = list DataNodeInfoList;
colset UnderReplicated = product Block * INT;
colset UnderReplicatedList = list UnderReplicated;
colset PendingReplication = product Block * INT * INT;
colset PendingReplicationList = list PendingReplication;
colset FSN = record fileMap:FileInfoList * blkMap:BlockInfoList
                    * dnMap:DataNodeMap
                    * neededReplications:UnderReplicatedList
                    * pendingReplications:PendingReplicationList
                    * dfsMetrics:DFSMetrics;
```

El color *FSN* está compuesto por una lista con información de los archivos (*fileMap*), una lista con información de los bloques (blkMap), una lista bidimensional (lista de lista) que representa a los nodos de datos del sistema (*dnMap*), la cola con prioridad que almacena a los bloques que necesitan replicación (*neededReplications*), la lista de bloques que están siendo replicados (*pendingReplications*) y unas métricas (*dfsMetrics*) que miden del sistema el total de espacio libre, espacio usado y la carga de concurrencia. A continuación se describen los colores definidos como parte del *FSN*.

El color *FileInfo* representa la información de cada archivo. Contiene el nombre, tamaño, una variable booleana que muestra si el archivo está abierto para escritura y una lista con los identifi-

cadores de los bloques que lo forman. A la lista de *FileInfo* se denotó como *FileInfoList*.

El color *BlockInfo* representa la información de cada bloque. Contiene el identificador y tamaño, nombre del archivo al que pertenece, una lista con los identificadores de los nodos de datos donde debe estar almacenado y otra lista con los identificadores de los nodos de datos donde ha sido planificado una replicación. A la lista de *BlockInfo* se denotó como *BlockInfoList*.

El color *DataNodeInfo* representa la información de los nodos de datos. Contiene el identificador del nodo, el último *heart beat* enviado, que no es más que una terna con los valores del espacio libre, espacio usado y la carga de concurrencia, la hora en que se envío el *heart beat*, el número de nuevos bloques que tiene planificado para que se copien en él, una lista con los bloques que se tienen almacenados, una lista con los bloques que deben ser replicados en otros nodos debido a que están por debajo del factor de replicación y una lista con los bloques que deben ser eliminados debido a que están por encima del factor de replicación. A la lista de *DataNodeInfo* se denotó como *DataNodeInfoList* y a la lista de *DataNodeInfoList* como *DataNodeMap*.

El color *UnderReplicated* representa la información de un bloque que necesita ser replicado. Contiene el bloque a replicar y un número que indica la cantidad de réplicas que le faltan al bloque para llegar al factor de replicación. A la lista de *UnderReplicated* se denotó como *UnderReplicatedList*. Para adicionar y extraer un elemento a la lista de tipo *UnderReplicatedList* se tiene en cuenta la cantidad de réplicas faltantes como la prioridad.

Por último, el color *PendingReplication* representa la información de un bloque que ha sido enviado a replicar. Contiene el bloque, la cantidad de réplicas y la hora en que se planificaron estas replicaciones. A la lista de *PendingReplication* se denotó como *PendingReplicationList*.

Es importante destacar que para el acceso y modificación de todos estos datos se han definido e implementado en el lenguaje CPN ML una serie de funciones que facilitan el diseño y construcción del modelo. Usándose estas funciones principalmente en las descripciones de arcos y segmentos de código.

2.3.2. Sub-transición *Client Communication*

La sub-transición *Client Communication* (figura 2.7) cuenta a su vez con cinco sub-transiciones que se encargan de recibir y responder todas las peticiones del nodo cliente correspondiente a los procesos de escritura y lectura, como se muestra en la figura 2.8. A continuación se describen cada una de ellas.

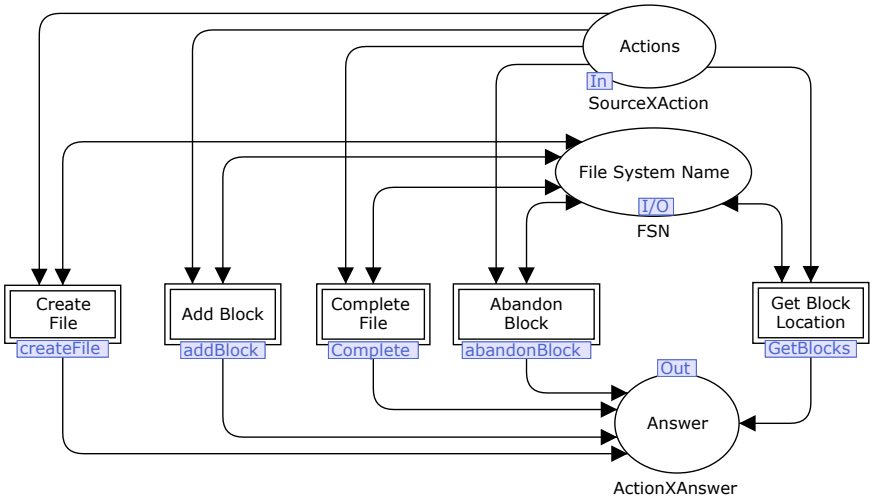


Figura 2.8: Modelado de la sub-transición *Client Communication*

La sub-transición *Create File* crea una nueva entrada en el sistema de fichero. La principal acción que realiza es crear un dato de tipo *FileInfo* y adicionarlo al *fileMap* del *FSN*. Retorna verdadero o falso en dependencia de si el archivo fue creado satisfactoriamente o no.

La sub-transición *Add Block* asigna un nuevo bloque a un fichero. La principal acción que realiza es crear un dato de tipo *BlockInfo*, adicionarlo al *blkMap* del *FSN* y además actualizar la lista de bloques del *FileInfo* correspondiente al archivo. Retorna el identificador del bloque creado con una lista que contiene a los identificadores de los nodos de datos más idóneos para replicar el dato del bloque, según la función *Rep1'chooseTarget*. Los nodos de datos devueltos por dicha función están ordenados de tal manera que se minimice la distancia a recorrer por el dato a copiar.

La sub-transición *Complete File* cierra el flujo de datos del proceso de escritura. La principal acción que realiza es obtener los datos de tipo *UnderReplicatedList* y adicionarlos al *neededReplications* del *FSN*. Retorna verdadero o falso en dependencia de si el archivo fue completado satisfactoria-

mente o no.

La sub-transición *Abandon Block* elimina un bloque asignado a un archivo. La principal acción que realiza es eliminar el dato de tipo *BlockInfo* del *blkMap* del *FSN*. Además de actualizar la lista de bloques del *FileInfo* correspondiente al archivo. Retorna verdadero o falso en dependencia de si el bloque fue abandonado satisfactoriamente o no.

Por último, la sub-transición *Get Block Location* obtiene, dado los bloques que forman un archivo, la ubicación de cada uno, es decir, para cada bloque de un archivo los nodos de datos en los que está replicado. La principal acción que realiza es iterar sobre el *blkMap* del *FSN* y para todos los *BlockInfo* que pertenecen al archivo consulta la lista *sourceDNs* para conformar la respuesta. Retorna una lista de duplas formadas por un bloque y una lista de identificadores con los nodos de datos que contienen al bloque.

2.3.3. Sub-transición *Data Node Communication*

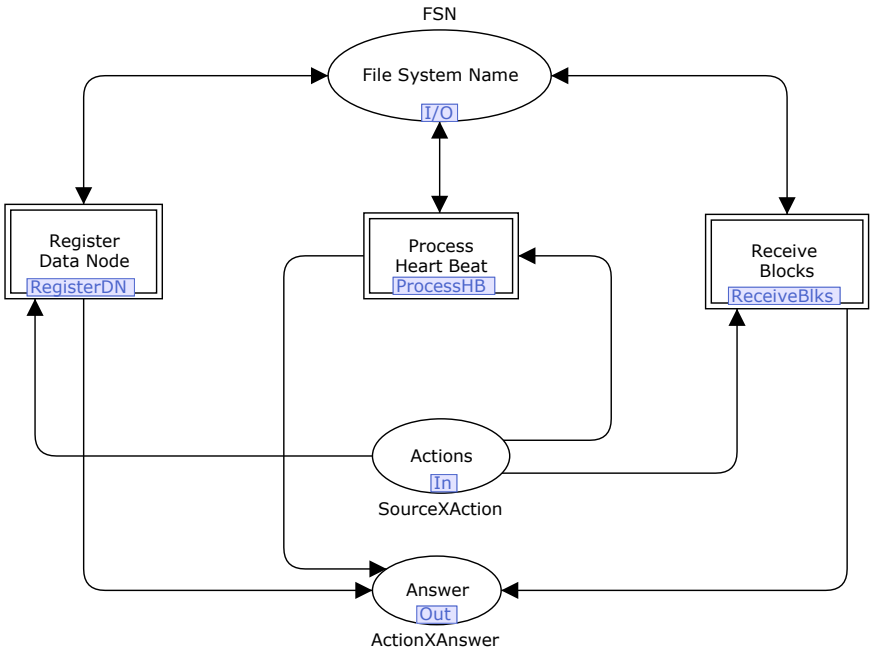


Figura 2.9: Comunicación del Nodo Servidor con los Nodos de Datos.

La sub-transición *Data Node Communication* (figura 2.7) cuenta a su vez con tres sub-transiciones que se encargan de recibir y responder todas las acciones enviadas por los nodos de datos, como se

muestra en la figura 2.9. A continuación se describen cada una de ellas.

La sub-transición *Register Data Node*, procesa las solicitudes de registro de los nodos de datos en el servidor. La principal acción que realiza es crear un dato de tipo *DataNodeInfo* que lo adiciona al *dnMap* del *FSN*. Retorna verdadero o falso en dependencia de si el nodo de datos fue registrado satisfactoriamente o no.

La sub-transición *Process Heart Beat*, procesa los mensajes con el dato de tipo *HeartBeat* que frecuentemente envían los nodos de datos para notificar que están activos. La principal acción que se realiza es obtener el *DataNodeInfo* almacenado en el *dnMap* del *FSN* para actualizar el *heart beat* y la hora en que se recibió el mismo. Retorna un grupo de acciones que debe realizar el nodo de datos relacionadas con la replicación de los bloques que están en la lista *blksToReplicate* y la eliminación de los que están en la lista *blksToInvalidate* del *DataNodeInfo* correspondiente.

La sub-transición *Receive Blocks*, recibe y procesa los bloques reportados por los nodos de datos. La principal acción que realiza es determinar los bloques nuevos y los que se han perdido, calculando la diferencia entre los bloques que están en la lista *storedBlk* del *DataNodeInfo* y los reportados, para actualizar el estado del *FSN*. Cada vez que se reportan nuevas réplicas de los bloques se verifica que los mismos no estén sobre-replicados, en cuyo caso se eliminen algunas réplicas al colocar los bloques en la lista *blksToInvalidate* de determinados nodos de datos que han sido seleccionados. La función *FSN'processOverRepBlks* implementa esta lógica en el lenguaje CPN ML. Por otra parte, para los bloques que han sido detectados como perdidos, se actualiza si es necesario la cola de bloques necesitados de replicación (*neededReplications*) del *FSN*. Esto es implementado en la función *FSNupUnderRepBlks* utilizando el lenguaje CPN ML.

2.3.4. Sub-transición *Heart Beat Monitor*

La sub-transición *Heart Beat Monitor* (figura 2.7) revisa periódicamente el momento (tiempo del modelo) en que los nodos de datos han enviado el último *heart beat* con el propósito de detectar los nodos que están fuera de servicio. Como se puede ver en la figura 2.10, la transición *Get Dead Data Nodes* auxiliándose de la función *DnMap'getDeadDataNodes* obtiene la lista de nodos de datos que no han dado señales de vida durante un cierto intervalo de tiempo y luego la transición *Remove*

Stored Block elimina los bloques (*BlockInfo*) pertenecientes a estos "nodos muertos" de la lista *blkMap* del *FSN*. Por último, se actualiza la cola con prioridad *neededReplications* del *FSN* con los bloques eliminados, por si necesitan replicación, utilizando la función *FSNúpUnderRepBlks* en la transición *Update Needed Replications*.

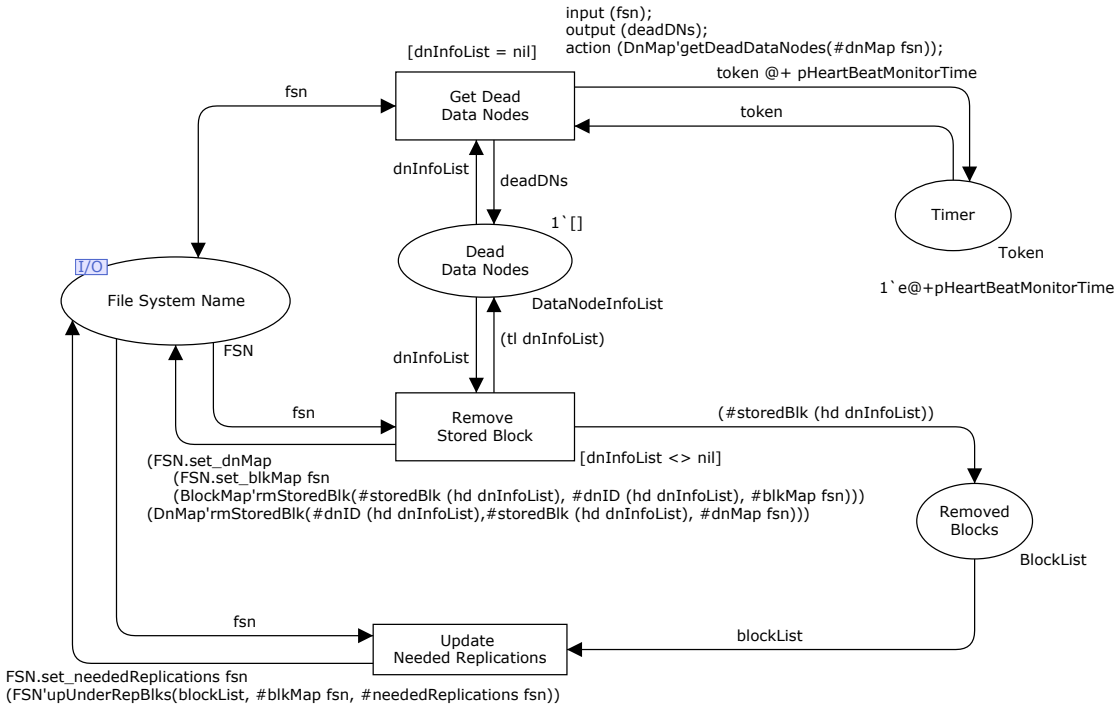


Figura 2.10: Monitorear los *Heart Beat*

En la figura 2.10 que el *token* en el lugar *Timer* tiene una marca de tiempo y esto hace que se habilite la transición *Get Dead Data Nodes* cada cierto intervalo de tiempo especificado en el parámetro *pHeartBeatMonitorTime*.

2.3.5. Sub-transición *Replication Monitor*

La sub-transición *Replication Monitor* (figura 2.7) se encarga de computar los trabajos de replicación que deben realizar los nodos de datos. Como se puede ver en la figura 2.11, la transición *Compute Replication Work* utiliza en la expresión del arco de salida la función *FSN'computeRepWork* implementada en CPN ML para extraer los bloques (*UnderReplicated*) necesitados de replicación que se encuentran en la cola con prioridad *neededReplications* del *FSN*, y asignarlos a la lista *blksToRe-*

plicate de los nodos seleccionados para que realicen la replicación según la estrategia implementada en HDFS. Una vez que esto sucede, se habilita la transición *Process Pending Replications*, que igualmente se auxilia de la función *FSN'processPendingRep* para comprobar si los trabajos de replicación que se encuentran en la lista *pendingReplications* del *FSN* han sido completados o necesitan ser re-planificados.

En la figura 2.11 que el *token* en el lugar *Timer* tiene una marca de tiempo y esto hace que se habilite la transición *Compute Replication Work* cada cierto intervalo de tiempo especificado en el parámetro *pReplicationMonitorTime*.

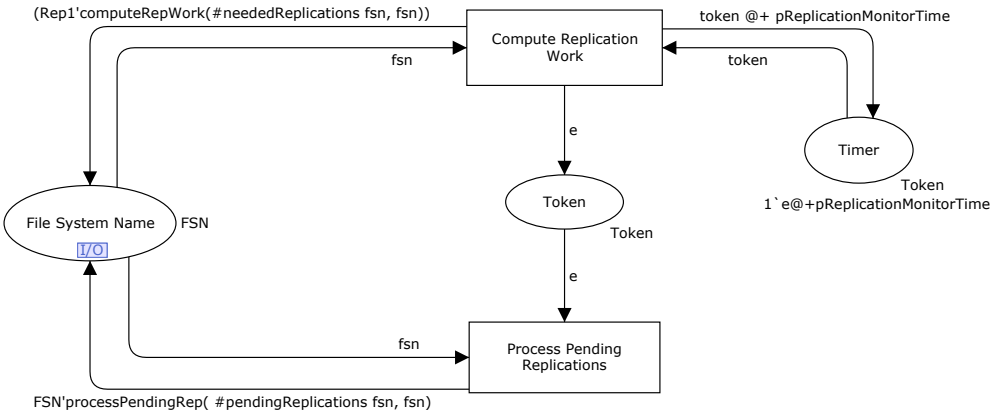


Figura 2.11: Monitorear la replicación

2.4. Sub-transición *Data Node*

La sub-transición *Data Node* (figura 2.1) tiene implementada la lógica de los nodos de datos que almacenan los bloques de los archivos, como se muestra en la figura 2.12 .

Algo que distingue a los nodos de datos es que ellos no solo reciben mensajes de los componentes clientes, también envían mensajes al servidor y más interesante aún es que envían y reciben mensajes hacia y desde otros nodos de datos. Para el modelado de este nodo la sub-transición *Data Node Core* recibe acciones las cuales procesa y envía la respuesta; pero también genera y envía acciones para las cuales recibe y procesa la respuesta.

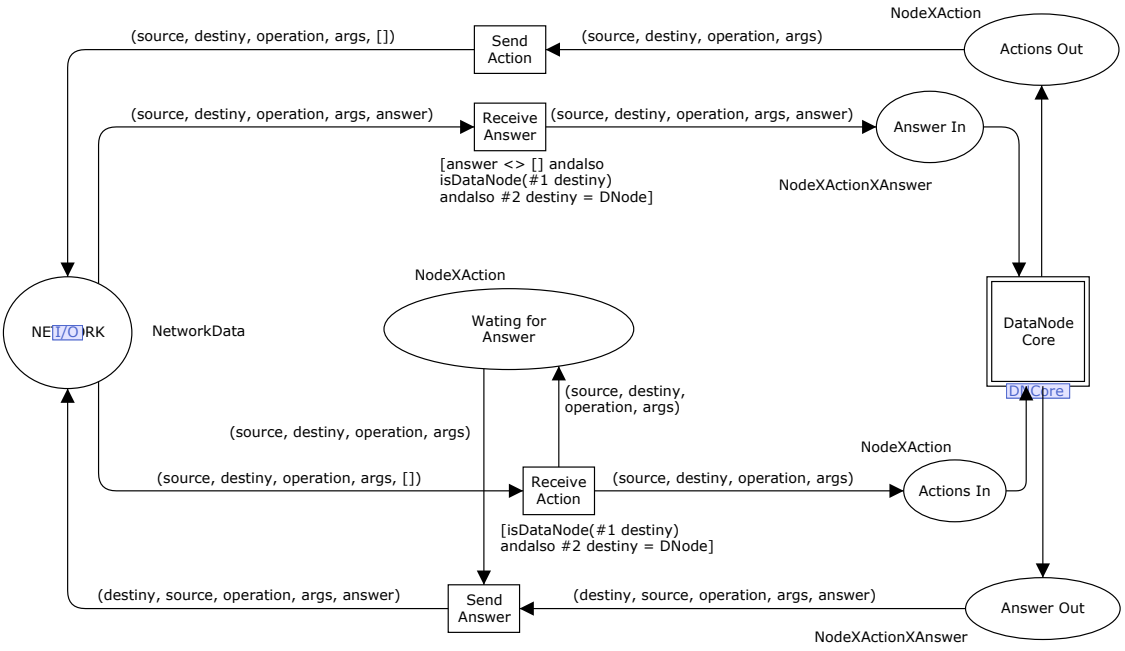


Figura 2.12: Modelado de la sub-transición *Data Node*.

2.4.1. Sub-transición *Data Node Core*

La sub-transición *Data Node Core* (figura 2.12) se detalla en la figura 2.13, donde al igual que en los componentes explicados anteriormente existen sub-transiciones que engloban los procesos fundamentales descritos en esta sección.

Como se puede apreciar en la figura 2.13, es en el lugar *Data Nodes* donde se encuentran ubicados los *tokens* que representan a los nodos de datos. Estos elementos son generados aleatoriamente a partir de ciertos parámetros en la función *genDataNodes()*, que aparece en la marca inicial del lugar *Data Nodes*. El color de este lugar es *DataNode* y es utilizado para modelar a una estación de trabajo que almacena bloques de archivos. A continuación se muestra la definición de este color y luego se describen las sub-transiciones implementadas en este componente.

```
colset OnOff = with On | Off;
colset DataNode = record dnID:DataNodeID * registered: BOOL
                         * freeMB:INT * usedMB:INT * status:OnOff
                         * blocks:BlockList * threadLoad:INT
                         * lastHeartBeat: INT * lastReport: INT;
```

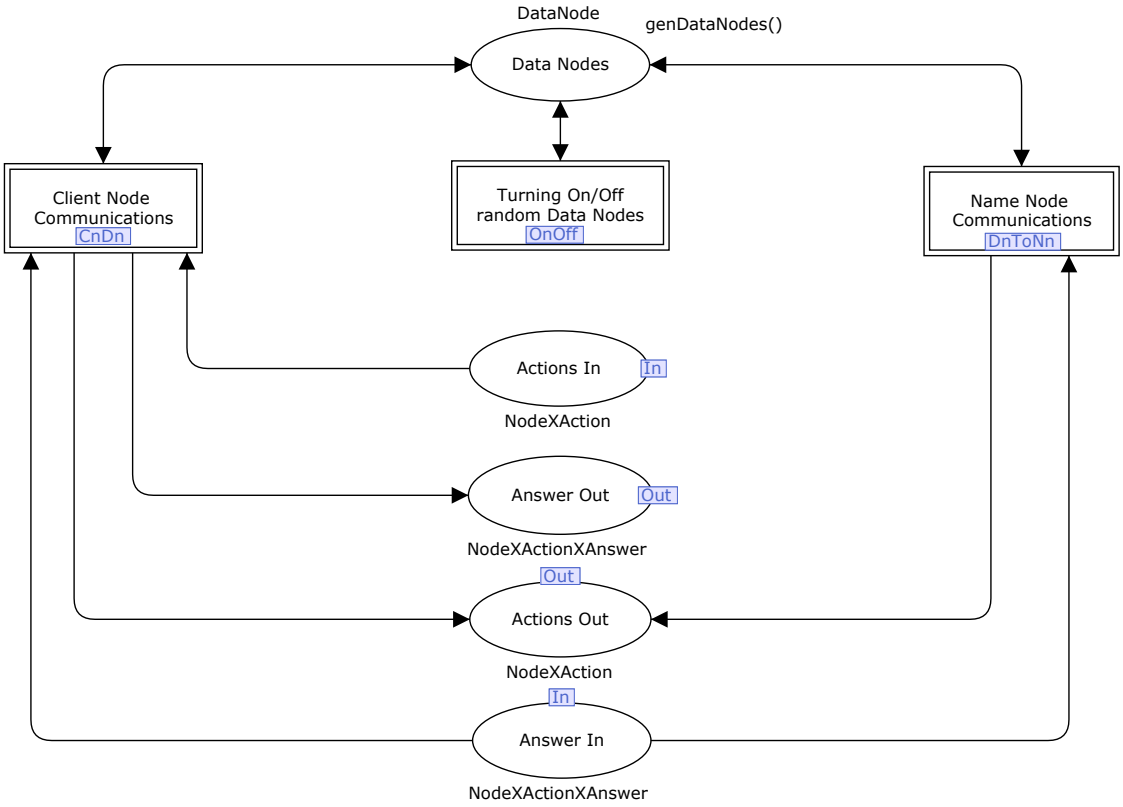


Figura 2.13: Modelado de la sub-transición *Data Node Core*.

El conjunto de colores *DataNode* es una estructura que consta de un identificador (*dnID*), un valor lógico (*registered*) que especifica si el nodo de datos se ha registrado en el nodo servidor o no, el espacio libre (*freeMB*), el espacio utilizado (*usedMB*), el estado de *On* u *Off* que indica la disponibilidad, la lista de bloques almacenados (*blocks*), un entero que indica la carga de concurrencia (*threadLoad*), un valor (*lastHeartBeat*) que representa el momento (tiempo del modelo) en que se envió el último *hear beat* y otro valor (*lastReport*), que representa el tiempo en que se realizó el último informe de los bloques que tiene almacenado un nodo de datos. Las instancias del conjunto de colores *DataNode* se generan de forma aleatoria de acuerdo a las características del entorno donde se desea desplegar el sistema.

2.4.2. Sub-transición *Client Node Communications*

La primera sub-transición *Client Node Communications* de la figura 2.13, atiende y procesa las peticiones de escritura y lectura de bloques de archivos que realizan los nodos cliente como se muestra en el modelado de la figura 2.14 . También da respuesta a una conexión que se quiere establecer con un nodo de datos. La respuesta será positiva o negativa en dependencia del estado *On* o *Off* que tenga el nodo en ese momento.

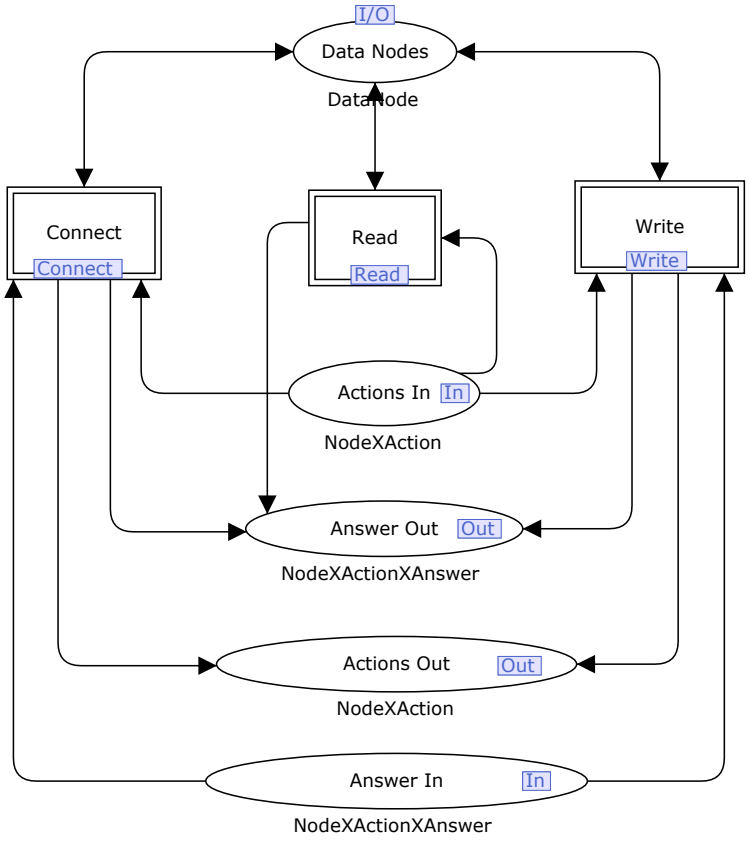


Figura 2.14: Modelado de la sub-transición que atiende y procesa las peticiones de escritura y lectura de bloques de archivos que realizan los nodos cliente

2.4.3. Sub-transición *Turning On/Off Random Data Nodes*

La segunda sub-transición *Turning On/Off Random Data Nodes* de la figura 2.13, regula la cantidad de estaciones de trabajo disponibles para el almacenamiento de datos tal y como se describe

en la sección 2.5.

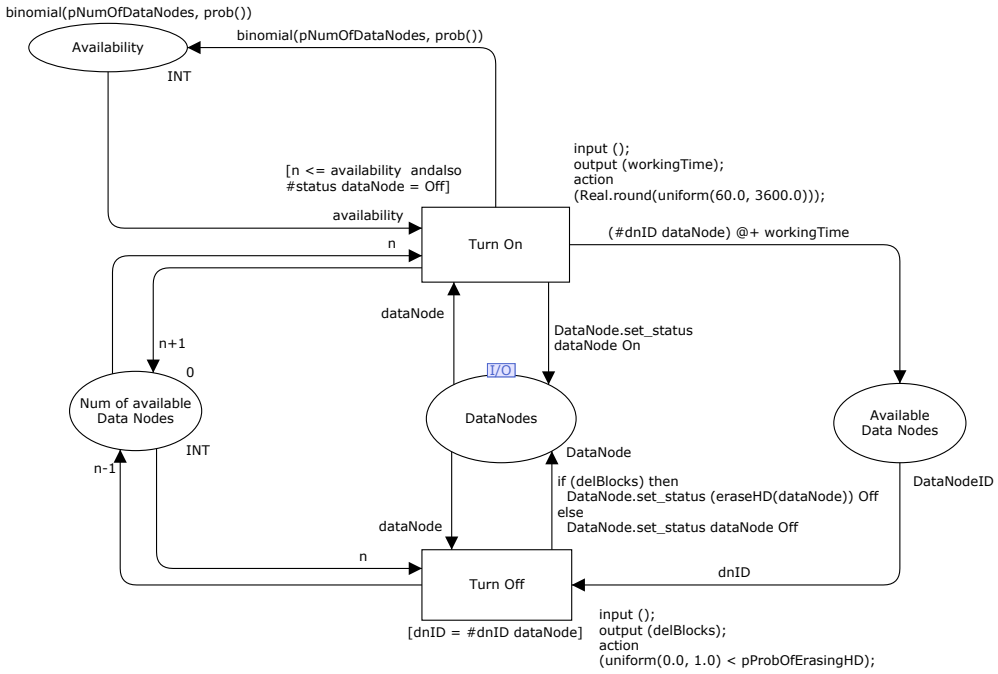


Figura 2.15: Modelado de la sub-transición que regula la cantidad de estaciones de trabajo disponibles para el almacenamiento de datos

En la figura 2.15 se puede ver la implementación de esta sub-transición, donde dos transiciones *Turn On* y *Turn Off* encienden y apagan respectivamente a los nodos de datos que se encuentran en el lugar *Data Nodes*. Cuando un nodo de datos es apagado se genera un número aleatorio, entre 0 y 1, y si ese número es menor que cierta probabilidad, especificada como parámetro, entonces se eliminan los bloques almacenados por ese nodo simulando una pérdida de los mismos.

2.4.4. Sub-transición *Name Node Communications*

Por último, la tercera sub-transición de la figura 2.13, *Name Node Communications*, se encarga de enviar mensajes con cierta frecuencia al nodo servidor para garantizar el buen funcionamiento del sistema. En la figura 2.16 se puede observar que estos mensajes son para registrar los nodos de datos una vez que son inicializados, reportar los bloques que se tienen almacenados e informar los nodos que se encuentran activos mediante los *heart beats*. En el caso de los reportes de los bloques y de los envíos de *heart beat*, la comunicación se realiza cada cierto intervalo de tiempo configurable

por parámetro.

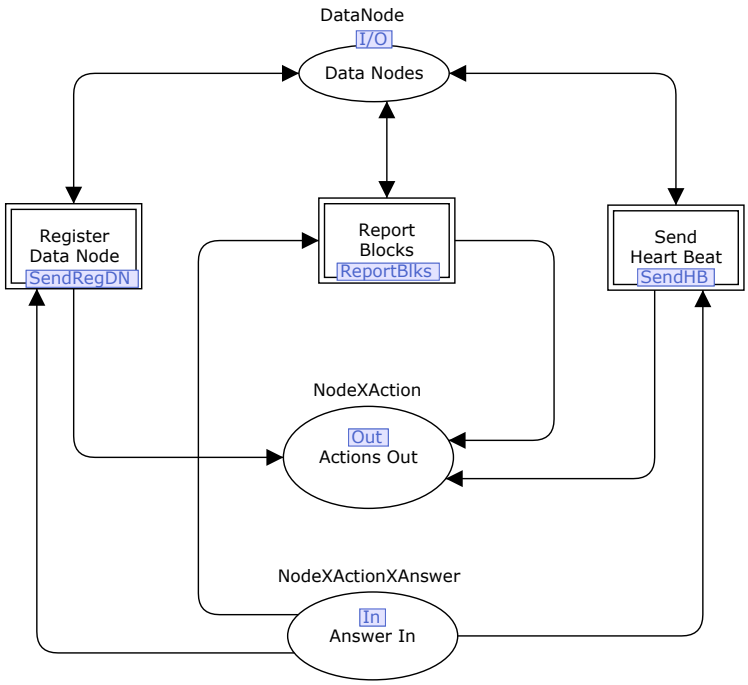


Figura 2.16: Modelado de la sub-transición que envía mensajes con cierta frecuencia al nodo servidor para garantizar el buen funcionamiento del sistema

2.5. Modelado de la disponibilidad de las estaciones de trabajo

Con el fin de modelar la disponibilidad de las estaciones de trabajo se realizó un estudio en el que se recogieron métricas de 127 ordenadores de distintos laboratorios docentes de la UCI. Las métricas fueron obtenidas utilizando una aplicación cliente/servidor en funcionamiento durante las 24 horas del día, por 40 días (del 1 de noviembre al 10 de diciembre de 2012).

Las figura 2.17 muestra el porcentaje de estaciones de trabajo encendidas en los laboratorios durante los 40 días. Una PC se contabilizó como encendida en el día si al menos se tiene un reporte para ese día. Como puede observarse, los porcentajes más bajos corresponden a los fines de semana, y más del 80 por ciento de las estaciones de trabajo están encendidas en la mayoría de los días.

La figura 2.18, por otra parte muestra el porciento calculado para cada una de las 24 horas del día. Ese por ciento es del promedio de máquinas encendidas por hora durante los 40 días sobre

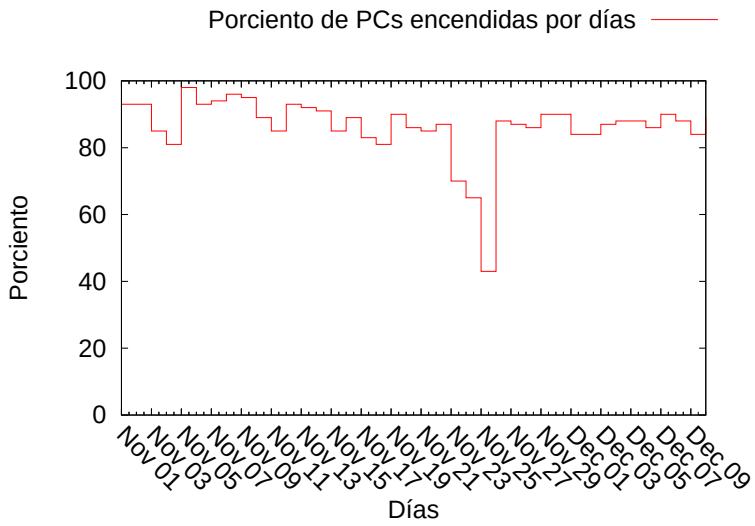


Figura 2.17: Porciento de PC encendidas por días.

el total de máquinas encendidas para cada día. Al observar la gráfica se puede apreciar que la disponibilidad de máquinas depende en gran medida de la hora del día. Los bajos por ciento de máquinas encendidas a determinados horarios responde a que los usuarios, con frecuencia, apagan los equipos antes de abandonar el laboratorio para dedicarse a otras actividades, cumpliendo así con las políticas de ahorro de energía eléctrica. Por lo tanto, se modelaron las máquinas encendidas por hora del día de la siguiente manera.

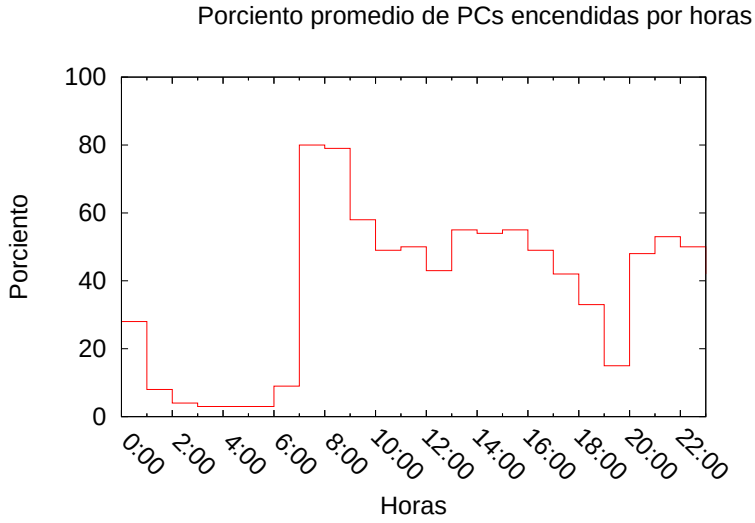


Figura 2.18: Porciento del promedio de máquinas encendidas por horas del día.

Sea la variable aleatoria de Bernoulli X_{ij} que toma los siguientes valores:

$$X_{ij} = \begin{cases} 1 & \text{si la estación de trabajo } i \text{ está encendida en la hora } j \\ 0 & \text{en otro caso} \end{cases} \tag{2.1}$$

para $i = 1, 2, \ldots, N$ y $j = 0, 1, 2 \ldots, 23$; donde N es el total de estaciones de trabajo. Si se denota a p_j como la probabilidad de que una computadora esté encendida en la hora j, entonces la probabilidad de un éxito es $Prob(X_{ij} = 1) = p_j$ y la probabilidad de un fracaso es $Prob(X_{ij} = 0) = 1 - p_j$.

Ahora, suponiendo que la variable aleatoria $\hat{X}_j$, que representa el total de máquinas encendida en la hora j, es definida como:

$$\hat{X}_j = \sum_{i=1}^{N} X_{ij} \tag{2.2}$$

La variable aleatoria $\hat{X}_j$ [1] tiene una distribución Binomial con parámetro N y p_j, denotado por $B(N, p_j)$. Siendo N un parámetro del modelo, mientras que las probabilidades fueron estimadas en base a los datos recopilados durante los 40 días en las 127 estaciones de trabajo: $p_j = \frac{n_j}{127}$, donde n_j es el número promedio de máquinas encendidas en la hora j. Por esta razón se utilizará la función de distribución aleatoria Binomial *binomial(n:int, p:real): int*, implementada en CPN ML, para generar el número de estaciones de trabajo encendidas a una hora determinada.

Un aspecto importante relacionado con la disponibilidad es la generación aleatoria del tiempo que las máquinas van a permanecer encendidas. Como es de interés predecir el comportamiento del sistema en un entorno caótico, la función de distribución aleatoria uniforme *uniform(a:real, b:real): real*,implementada en CPN ML, fue seleccionada para generar en el intervalo $(0, 3600]$, el período de tiempo en segundos que van a estar disponibles las estaciones de trabajo encendidas a una hora determinada.

Por último, cada vez que se apaga una máquina, de manera aleatoria se determina si se eliminan o no los bloques que almacena el nodo de datos apagado. Para ello, se utiliza la función *uniform(a:real, b:real): real* para generar un número aleatorio entre 0 y 1, se compara ese número con el parámetro

[1] Una variable aleatoria Binomial puede ser considerada como la suma de n variables aleatorias de Bernoulli.

pProbOfErasingHD, que representa la probabilidad de borrar todos los datos de un nodo al ser apagado, y si resuelta que el número aleatorio es menor pues entonces se eliminan los datos.

2.6. Validación del modelo

Como el modelo se encuentra en una fase temprana de su desarrollo, la validación realizada se realizó con el fin de verificar su funcionamiento: se compararon los resultados de las solicitudes de escritura y lectura que arroja el modelo, con los que se obtienen en el sistema real, desplegado en un entorno de prueba. Para realizar los experimentos de validación, se creó un escenario con 22 máquinas a las cuales se le instaló el sistema HDFS con los valores de configuración por defecto. Luego se ejecutó una aplicación cliente que se conecta con el sistema para enviar 100 solicitudes de escritura y 100 de lectura, a intervalos aleatorios, durante un periodo de tiempo de cuatro horas en la sesión de la mañana. El estudio de caso para la validación se replicó tres veces al igual que las simulaciones para comparar los resultados.

En la figura 2.19 se representan los eventos de apagado y encendido que experimentaron cada una de las PCs en una de las réplicas. La gráfica muestra en el eje y las 22 PCs que formaron parte del experimento de validación, mientras que en el eje x se representa el tiempo. Los segmentos que se grafican representan las operaciones de encendido y apagado de la máquina y y los tiempos que estuvieron trabajando cada una de las PCs. En el caso de las simulaciones, el encendido y apagado se comportó aleatoriamente tal y como fue descrito en la sección anterior.

Los resultados de la figura 2.20, muestran una comparación gráfica de las cantidades de solicitudes que fueron respondidas satisfactoriamente en cada uno de los tres estudios de caso de validación. Estos valores se obtienen a partir de las simulaciones y las corridas en el entorno real para ambos procesos: escritura y lectura. Para obtener los resultados del sistema real se realizaron los eventos de encendido y apagado de manera similar a como ocurrieron en las simulaciones. Bajo estas condiciones, los resultados presentados en la gráfica 2.20 tienen valores similares y en el entorno real son mejores que en las simulaciones. De esta manera se asume que si el comportamiento en las simulaciones es bueno, también lo será en el sistema real.

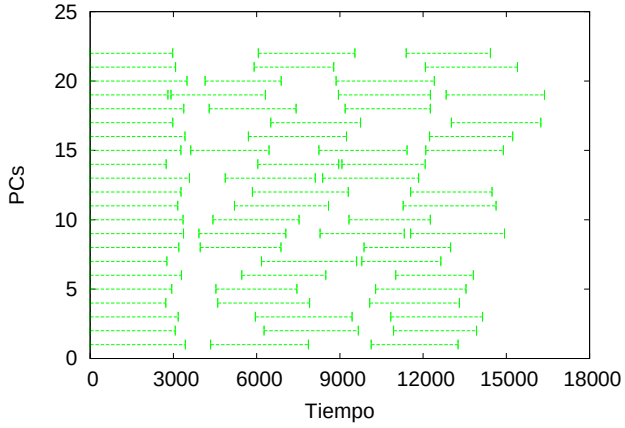
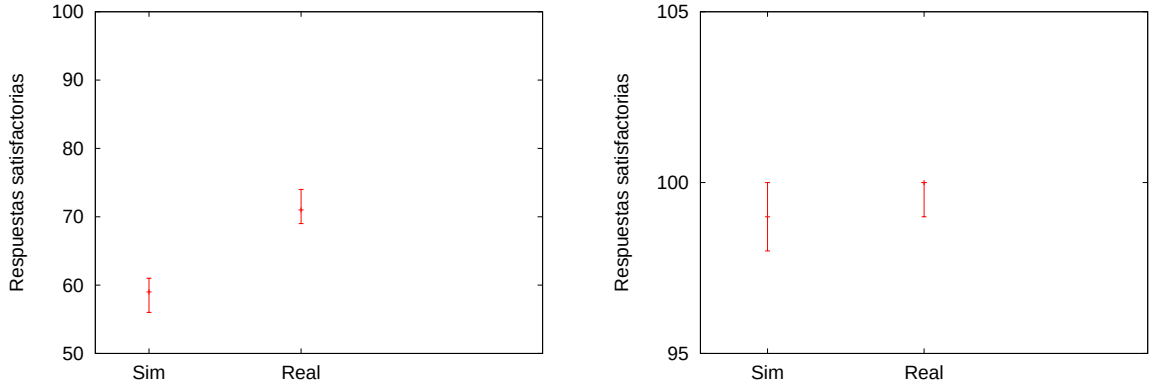


Figura 2.19: Eventos de encendidos y apagados.



(a) Proceso de Escritura

(b) Proceso de Lectura

Figura 2.20: Comparación entre los valores de las simulaciones y el entorno real teniendo en cuenta la cantidad de respuestas satisfactorias en el proceso de escritura (a) y el de lectura (b).

2.7. Conclusiones parciales

La estructura jerárquica del modelo permite una organización por módulos y mejor comprensión de los procesos fundamentales que intervienen en la escritura y lectura de archivos.

El modelo implementado contempla las características de un entorno real y los parámetros utilizados pueden ser modificados para explorar diferentes escenarios y alternativas de configuración.

El estudio realizado durante 40 días, en algunos laboratorios docentes de la UCI, permitió modelar la disponibilidad de las estaciones de trabajo mediante el empleo de funciones estadísticas de distribución implementadas en el lenguaje CPN ML.

Los fallos aleatorios incorporados al modelo permitieron analizar la tolerancia a fallos del sistema ante situaciones como: el apagado y encendido de las máquinas y la pérdida total de la información del disco duro.

La validación del modelo mediante la realización de un estudio de caso permitió comprobar que las cantidades de solicitudes respondidas satisfactoriamente en la simulación se asemejan a las obtenidas en el entorno real.

Capítulo 3

Simulaciones y análisis de los resultados

En este capítulo se analiza el sistema modelado mediante la ejecución de varias simulaciones que han sido diseñadas teniendo en cuenta las características del entorno de la UCI. Primeramente se realizan experimentos de diagnóstico con el propósito de detectar problemas con antelación, determinar las causas e implementar acciones correctivas. En una segunda etapa se realizan experimentos para comprobar la efectividad de los ajustes implementados. Finalmente, se experimenta con el modelo ajustado para evaluar el desempeño e inferir el comportamiento que tendría el sistema en el entorno real antes de ser desplegado.

3.1. Marco experimental

En esta sección se caracterizan las circunstancias bajo las cuales se experimenta en el sistema modelado. También se describen las variables de salida que serán observadas por los monitores implementados en el modelo.

3.1.1. Parámetros de entrada

Parámetro	Valores	Descripción
pHourInit	8	Hora a la que se comienza la simulación
pHourEnd	12	Hora a la que termina la simulación
pNumOfReadingRequest	100	Cantidad de solicitudes de lectura
pNumOfWritingRequest	100	Cantidad de solicitudes de escritura
pNumOfDataCenter	1	Cantidad de *Data Center*
pNumOfRack	6	Cantidad de *Racks*
pNumOfHostsPerRack	100	Cantidad de computadoras por *Racks*
pMinDiskFreeMB	1024	Cantidad mínima de espacio libre en los discos duros, en *Megabyte*
pMaxDiskFreeMB	10240	Cantidad máxima de espacio libre en los discos duros en *Megabyte*
pMinFileSizeMB	500	Tamaño mínimo de los archivos en *Megabyte*
pMaxFileSizeMB	1000	Tamaño máximo de los archivos en *Megabyte*
pReplicationFactor	3,5,7,9,11	Cantidad de réplicas de cada bloque
pBlockSizeMB	64	Tamaño de los bloques en *Megabyte*
pNumOfTriesInAllocatingBlock	3	Número de intentos para asignar un *pipeline*
pHeartBeatIntervalTime	300	Intervalo de tiempo entre entre los reportes de estado *(Heart Beats)* que realizan los nodos de datos al servidor.
pBlockReportIntervalTime	600	Intervalo de tiempo entre los reportes que realizan los nodos de datos de los bloques que tienen almacenados
pHeartBeatExpirationTime	3600	Tiempo límite que puede estar un nodo de datos sin reportarse y no ser considerado fuera de servicio
pHeartBeatMonitorTime	900	Intervalo de tiempo entre los chequeos que se realizan para determinar si hay nodos de datos fuera de servicio
pReplicationMonitorTime	300	Define cada qué tiempo se realizan los procesos necesarios para que los bloques alcancen el número de réplicas definido
pPendingReplicationsTimedout	900	Define el tiempo límite que un bloque puede estar en la lista de los bloques pendientes de replicación
pProbOfErasingHD	0.2	Valor de probabilidad que se utiliza para determinar si se procede a simular el borrado completo de los bloques de un disco duro.

Cuadro 3.1: Parámetros utilizados en las simulaciones realizadas.

En la tabla 3.1 se muestran los principales parámetros del modelo, una descripción de cada uno de ellos y los valores utilizados en las simulaciones. Cambiando estos valores se altera el comportamiento del sistema. Por lo tanto, la mayoría de los parámetros se mantuvieron constantes durante las simulaciones, excepto el factor de replicación que se incrementó gradualmente comenzando en 3 (valor por defecto en HDFS) hasta 11.

El horario de 8:00AM a 12:00M fue el seleccionado para realizar las primeras simulaciones de experimentación ya que en este horario se encontraron activas un mayor número de computadoras (ver figura 2.18). También se especificó la cantidad de 100 solicitudes de escritura y lectura respectivamente para ser generadas aleatoriamente así como el intervalo de arribo en el tiempo de estas solicitudes, que en este último caso se calcula de forma automática a partir de los horarios establecidos y las cantidades de solicitudes especificadas.

En cuanto al entorno de la institución, se consideró a la UCI como un gran centro de datos (*Data Center*) y a cada uno de los 6 docentes como un *Rack* provisto de 100 computadoras. Lo cual representa un escenario con un total de 600 computadoras. Para la identificación de cada máquina se utilizaron tres coordenadas $[x, y, z]$, donde x es el número del *Data Center*, y es el número del *Rack* dentro del *Data Center* y z es el número de la PC en el *Rack*. Siendo la PC con la coordenada $[0, 0, 0]$ la seleccionada como *Name Node*.

En el proceso de generación aleatoria de datos también se utilizaron algunos parámetros para especificar los rangos en el que estarán las cantidades de espacio libre que tendrán las estaciones de trabajo (entre 1 GB y 10 GB), y los tamaños de los ficheros que serán almacenados (entre 500 MB y 1000 MB). De esta manera, se experimentó utilizando a lo sumo 10 GB de espacio libre de cada PC y asumiendo que los datos a almacenar son archivos de multimedia cuyos tamaños se encuentran en el rango especificado.

Otros parámetros, además del factor de replicación, que existen en el producto HDFS fueron incorporados al modelo. En el caso de *pBlockSizeMB*, *pNumOfTriesInAllocatingBlock* y *pTimeToSleepInAllocatingBlock* se especificaron con los valores por defecto que tienen en HDFS. Sin embargo, los parámetros *pHeartBeatIntervalTime*, *pBlockReportIntervalTime*, *pHeartBeatMonitorTime*, *pReplicationMonitorTime*, *pHeartBeatExpirationTime*, *pPendingReplicationsTimedout* fueron configu-

rados con valores menores a los que tiene por defecto HDFS, debido a que se quiere experimentar en un entorno donde la entrada y salida de las computadoras tendrán un comportamiento caótico.

Por último, con el propósito de simular la perdida de datos por fallas en el disco duro se introdujo el parámetro *pProbOfErasingHD* con valor igual a 0,2, que se tiene en cuenta al encender una PC para determinar si se eliminan o no todos los datos almacenados. Mientras que el parámetro *pNetworkTransferMBps* es utilizado para introducir un retraso (*delay*) en el envío de datos, su valor es de 12,5 debido a que la tasa de transferencia de la red universitaria es de 100 Mb/s.

3.1.2. Variables que serán observadas

Variable	Descripción
τ	Representa el tiempo en el modelo.
$W(\tau)$	Respuesta positiva o negativa que se obtiene en el tiempo τ para una solicitud de escritura.
$Wb(\tau)$	Respuesta positiva o negativa que es recibida en el tiempo τ para la escritura de un determinado bloque de archivo.
$R(\tau)$	Respuesta positiva o negativa que se obtiene en el tiempo τ para una solicitud de lectura.
$Rb(\tau)$	Respuesta positiva o negativa que es recibida en el tiempo τ para la lectura de un determinado bloque de archivo.
$Dw(\tau)$	Tiempo que demoró la solicitud de escritura cuya respuesta es obtenida en el tiempo τ.
$Dr(\tau)$	Tiempo que demoró la solicitud de lectura cuya respuesta es obtenida en el tiempo τ.
$F(\tau)$	Espacio total (MB) del sistema que es reportado como libre en el tiempo τ.
$U(\tau)$	Espacio total (MB) que es usado por el sistema en el tiempo τ.
$B(m,n)$	Cantidad de bloques almacenados en el nodo de datos número n del *Rack* m, donde $m = 1...pNumOfRack$ y $n = 1...pNumOfHostsPerRack$.
$I(\tau)$	Número del intento realizado en el tiempo τ para asignar una lista de nodos de datos (*pipeline*) donde se debe copiar un determinado bloque.
$C(\tau)$	Cantidad de máquinas activas en el tiempo τ.
$N(\tau)$	Cantidad de *tokens* que se encuentran en la red (estado "Network" de la figura 2.1) en el tiempo τ.
$E_1(\tau)$	Identificador de la máquina que por un fallo provocado aleatoriamente tuvo que salir del sistema en el tiempo τ.
$E_2(\tau)$	Identificador de la máquina que por un fallo provocado aleatoriamente perdió todos los datos (bloques de archivos) que tenía almacenado en el tiempo τ.

Cuadro 3.2: Variables que serán observadas durante las simulaciones.

En la tabla 3.2 se muestran las variables que serán observadas durante las simulaciones para producir las salidas que permitan un análisis del comportamiento del sistema, siendo $W(\tau)$ y $R(\tau)$ las más importantes. La observación de estas variables se realizó con el empleo de los monitores que proporciona la herramienta CPN Tools. De manera general, se recopila información acerca de la

calidad del proceso de escritura, lectura y algunas métricas que denotan el rendimiento del sistema modelado.

3.2. Resultados de las corridas pilotos

En una primera fase se realizaron corridas pilotos en el horario de 8:00AM a 12:00M para la detección temprana de problemas que puedan ser solucionados aplicando los correctivos necesarios. La figura 3.1 muestra los resultados de estas simulaciones teniendo en cuenta el porcentaje de solicitudes de escritura y lectura que fueron respondidas con éxito, a medida que aumenta el factor de replicación. Claramente se puede observar que las respuestas satisfactorias de lectura aumentan al incrementar el número de réplicas y que, por el contrario, las respuestas exitosas de escritura disminuyen.

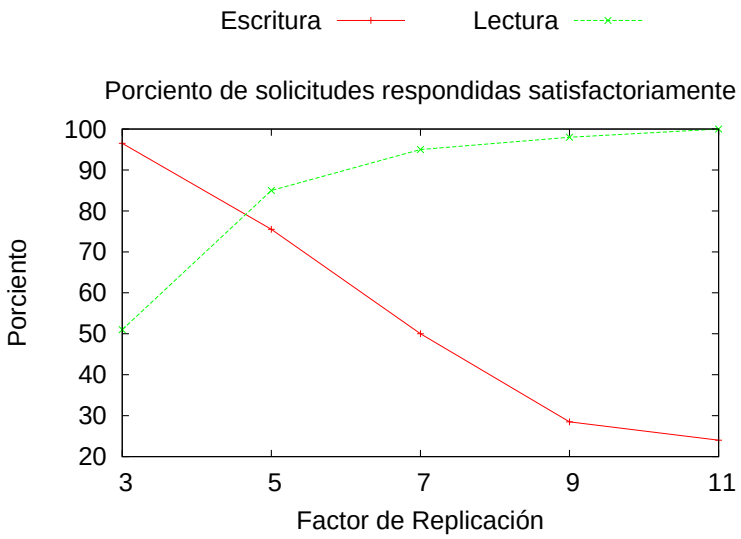


Figura 3.1: Porciento de respuestas satisfactorias de las solicitudes de escritura y lectura, obtenidas en las corridas pilotos.

Al indagar en el por qué de los resultados no deseables en el proceso de escritura, se detecta que para todas las solicitudes que fallaron se encontró que el número de intentos para asignar un *pipeline* con todos los nodos de datos activos excede lo permitido en el parámetro del modelo *pNumOfTriesInAllocatingBlock*, causando errores en la escritura. Matemáticamente, es-

te hallazgo se puede describir de la siguiente manera: $\forall\tau : W(\tau) = -1$ se cumple que $I(\tau) > pNumOfTriesInAllocatingBlock$, donde las variables $\tau, W(\tau) e I(\tau)$ se denotan en la sección 3.1.2.

3.2.1. Implementación de acciones correctivas

Es importante recordar, según lo descrito en la sección 2.2.2, que en el código fuente de HDFS se encuentra implementada la restricción de que el *Client Node* rechace el *pipeline* devuelto por el *Name Node* si al menos un nodo inactivo hace que el *pipeline* esté desconectado. Cuando esto sucede, el *Client Node* solicita una nueva lista de nodos para copiar el bloque si el número del intento no es mayor que un valor especificado en el fichero de configuración.

Esta restricción en un entorno dedicado garantiza una mayor certeza en la réplica de datos, pero en un entorno no dedicado donde las máquinas entran y salen aleatoriamente sucede que hay mayor probabilidad de encontrar un *pipeline* desconectado que termine siendo rechazado. Además, dicha probabilidad aumenta a medida que se incrementa el factor de replicación que está asociado directamente con la longitud del *pipeline*, de ahí los malos resultados de la figura 3.1 durante el proceso de escritura.

Teniendo en cuenta lo expresado en el párrafo anterior, una "relajación" de la mencionada restricción se hace necesario implementar en el modelo para luego comprobar si mejoran los resultados. Por este motivo, se implementó otra variante a seguir: una vez que el *Name Node* brinde los nodos de datos para realizar la escritura y réplica del bloque, se comprueba la conectividad de cada uno de ellos y se eliminan del *pipeline* a aquellos nodos que no estén disponibles en ese momento, aunque si lo hayan estado en el momento en que fueron seleccionados. De esta manera, solamente se solicita un nuevo *pipeline* si ningún nodo está activo.

Para una representación gráfica del mal causante de los errores de escritura, las gráficas de la figura 3.2(a) muestran el comportamiento de los intentos para encontrar un *pipeline* de tamaño 11 completamente conectado. Los impulsos representan los intentos, que pueden ser desde uno hasta tres según lo especificado en el parámetro *pNumOfTriesInAllocatingBlock*. Si al tercer intento el bloque no puede ser copiado por no haberse encontrado un *pipeline* adecuado, entonces se utiliza en la gráfica un valor negativo para representarlo.

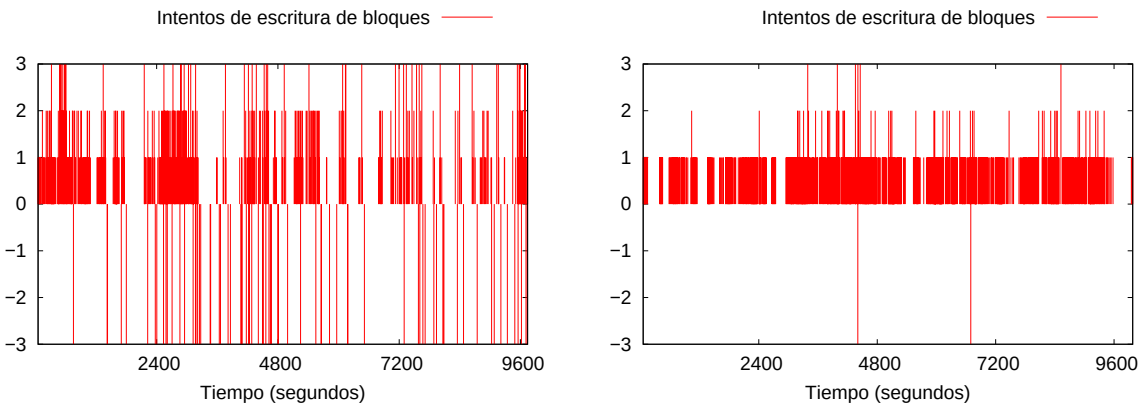


Figura 3.2: Comportamiento del número de intentos para asignar un *pipeline* a un bloque durante el proceso de escritura, antes (a) y después (b) de la implementación de las acciones correctivas.

Por otro lado la figura 3.2(b) muestra que: el numero de intentos de asignación de un *Pipeline* al bloque que falla, es significativamente menor una vez implementadas las correcciones en el modelo. Con esta nueva variante se favorece el proceso de escritura, pero el proceso de lectura podría verse afectado ya que no hay garantía de que los bloques se hayan copiado con todas sus réplicas.

Sin embargo, gracias a la estrategia de replicación implementada en HDFS, el sistema tiene sus mecanismos ya explicados anteriormente (sección 2.2.2) para detectar los bloques que están por debajo del factor de replicación y completar el número de réplicas necesarias. Los resultados de las simulaciones para la escritura y lectura con esta nueva variante muestran una notable mejoría en el proceso de escritura sin grandes afectaciones en las solicitudes de lectura (ver figura 3.3).

3.3. Resultados de las corridas finales

Una vez implementadas las acciones correctivas en el modelo, se fijo el parámetro *pReplication-Factor* en 11 réplicas por cada bloque, siendo al parecer una configuración de las más adecuadas para que funcione correctamente el sistema en un entorno con las características modeladas (figura 3.3). Además se realizaron corridas que simulan el sistema funcionando 24 horas. En las siguientes sub-secciones se muestran los resultados finales obtenidos a partir de la ejecución de varias simulaciones con la configuración seleccionada.

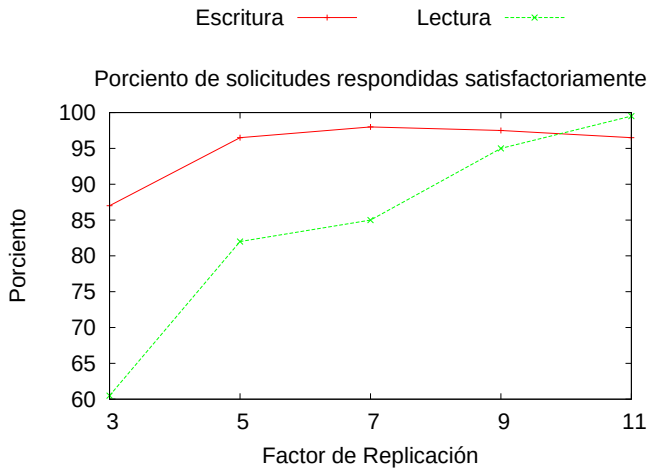
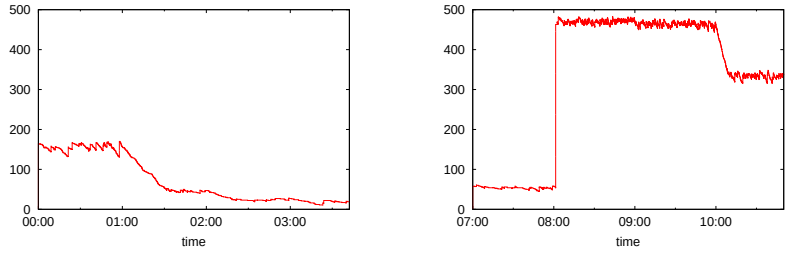


Figura 3.3: Porciento de respuestas satisfactorias de las solicitudes de escritura y lectura luego de implementadas las acciones correctivas.

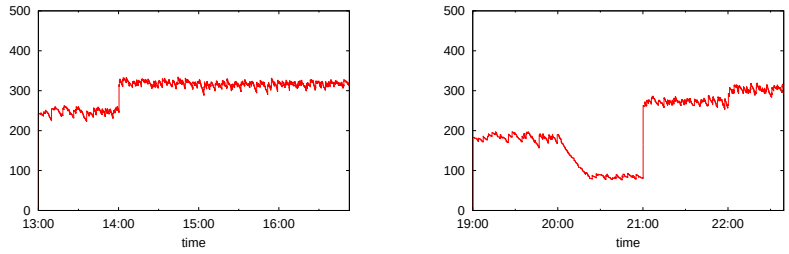
3.3.1. Disponibilidad y fallos aleatorios de los recursos computacionales

Primeramente es importante analizar el número de recursos computacionales que estuvieron disponibles a lo largo de las simulaciones y los fallos aleatorios que provocaron perdida de datos en algunos nodos. La figura 3.4 muestra el comportamiento de las máquinas activas durante los horarios de la madrugada, mañana, tarde y noche según lo explicado en la sección 2.5. Se aprecia que en los horarios de la mañana y la tarde, es donde mejor se comporta la disponibilidad de los recursos.

Por otra parte, los fallos ocurridos que provocaron el apagado de las máquinas y la perdida de datos se representan como cruces en las figuras 3.5(a) y 3.5(b) respectivamente. En la abscisa de ambas figuras se encuentra el tiempo en que ocurrió el suceso y en las ordenadas aparece el identificador de la PC que fue víctima del fallo. Estos fallos se realizan para que el modelo simule el apagado, encendido, formateo o rotura de las PCs que forman parte del sistema.

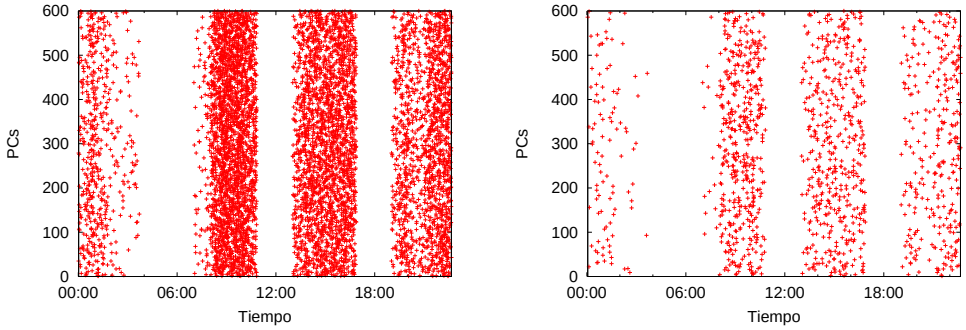


(a) Madrugada (b) Mañana



(c) Tarde (d) Noche

Figura 3.4: Disponibilidad de las PCs en los horarios de: (a) la madrugada, (b) la mañana, (c) la tarde y (d) la noche



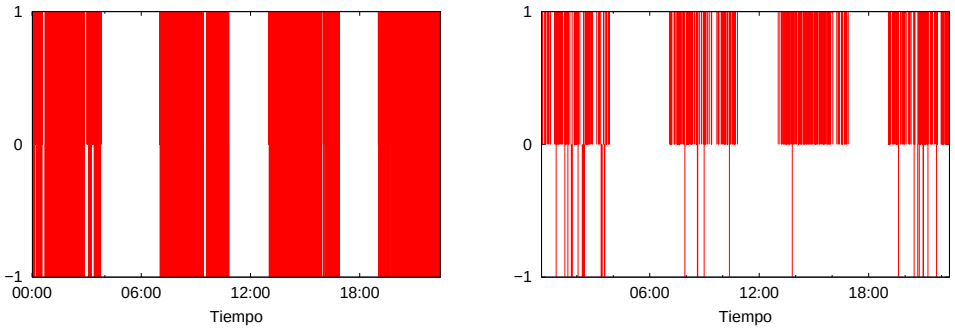
(a) Fallos de las PCs (b) Borrado de bloques

Figura 3.5: Eventos que ocurren en el modelo para representar los fallos de las PCs (a) y el borrado de los bloques de las PCs (b).

3.3.2. Respuestas del proceso de escritura y lectura

Una vez expuesta la disponibilidad de los recursos computacionales y los fallos aleatorios ocurridos, en la figura 3.6 se presentan dos nuevas gráficas: la de la izquierda (a), muestra las respuestas

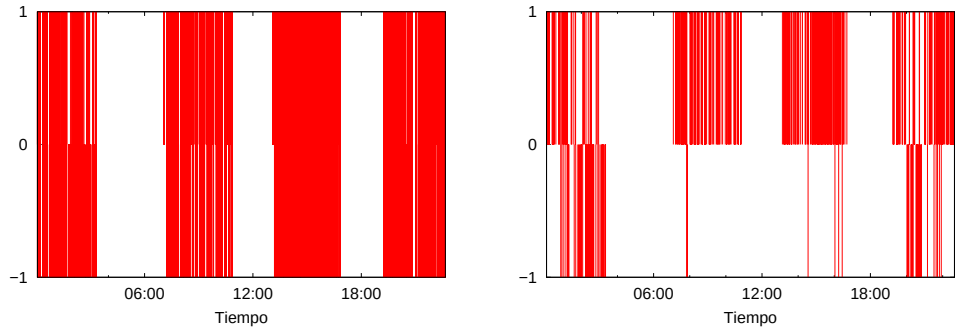
positivas ($Wb(\tau) = 1$) y negativas ($Wb(\tau) = -1$) correspondientes a la escritura de los bloques de archivo; mientras que la de la derecha (b), muestra las respuestas positivas ($W(\tau) = 1$) y negativas ($W(\tau) = -1$) de las solicitudes de escritura. Nótese que a pesar de todas las fallas que ocurren en la copia de determinadas réplicas de bloques (impulsos negativos en la figura de la izquierda), son pocos los fallos que ocurren en la copia de archivos (impulsos negativos en la figura de la derecha) ya que con copiar al menos una réplica por cada bloque, es suficiente para almacenar satisfactoriamente el archivo.



(a) Escritura de bloques (b) Escritura de archivos

Figura 3.6: Respuestas positivas y negativas que se obtienen para la escritura de los bloques (a) y archivos (b).

De manera similar se presentan las gráficas para el proceso de lectura de bloques y archivos, ver figura 3.7. Los resultados evidencian la alta tolerancia a fallos implementada en el sistema ya que, a pesar de existen fallos en la lectura de determinados bloques (impulsos negativos en la figura de la izquierda), se obtienen pocas respuestas insatisfactorias en el proceso de lectura de archivos (impulsos negativos en la figura de la derecha). En caso de que no se pueda recuperar un bloque a partir de un nodo de datos (lectura fallida), entonces se intenta recuperar el mismo bloque a partir de otra réplica almacenada en otro nodo del sistema. De esta manera, cuando todos los bloques son recuperados el archivo se reporta como que ha sido leído satisfactoriamente. En caso contrario, si existe al menos un bloque que no puede ser recuperado entonces se reporta una falla en la lectura del archivo.

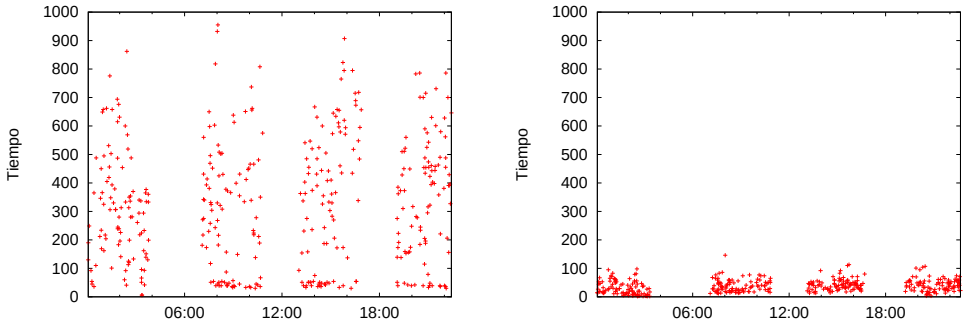


(a) Lectura de bloques (b) Lectura de archivos

Figura 3.7: Respuestas positivas y negativas que se obtienen para la lectura de los bloques (a) y archivos (b).

3.3.3. Tiempo que demoraron las solicitudes en responderse

En la sección anterior se analizó la calidad de las respuestas que brinda el sistema, pero también resulta interesante estudiar el tiempo que demora el sistema en dar una respuesta. Las figuras 3.8(a) y 3.8(b) muestran los tiempos de respuestas para las solicitudes de escritura y lectura respectivamente. Como se puede apreciar, escribir un archivo en el sistema demora más que leerlo.



(a) Proceso de escritura (b) Proceso de lectura

Figura 3.8: Tiempo de demora del proceso de escritura (a) y de lectura (b).

Esta característica es conveniente para el sistema estudiado, una de sus particularidades es que se escriben los archivos una sola vez y se leen varias veces, por lo que el proceso de lectura debe ser más rápido. La causa principal que retarda el proceso de escritura se debe a que el archivo es dividido en bloques y cada uno es replicado de 1 a 11 veces.

3.3.4. Transferencia de datos por la red

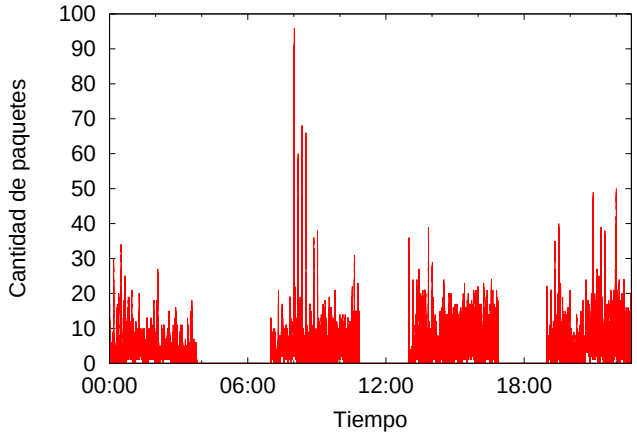
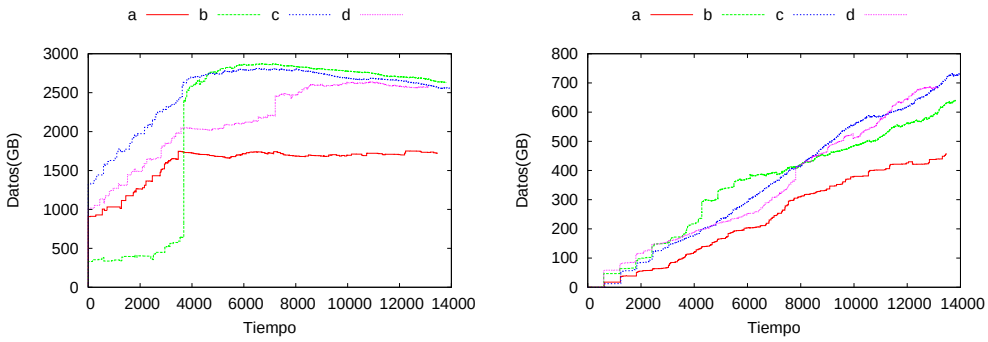


Figura 3.9: Transferencia de datos por la red.

Uno de los recursos estudiado fue la red para conocer el tráfico que puede generar el sistema modelado. En la figura 3.9 se representa la variabilidad en función del tiempo de la cantidad de *tokens* en el estado que modela la red (*Network*). Por lo general, se puede apreciar que el tráfico de datos por la red se mantiene estable la mayoría del tiempo. Sin embargo aparecen picos, que representan los instantes en que más datos transitan por la red. Esto ocurre cuando la mayoría de los nodos de datos se reportan al nodo servidor para el intercambio de información.

3.3.5. Espacio libre y espacio usado por el sistema

Como prueba de un buen aprovechamiento de los recursos disponibles, resulta interesante analizar el comportamiento del espacio libre y usado por el sistema en los horarios de la madrugada, mañana, tarde y noche. Como se muestra en la figura 3.10(a), el espacio disponible en un inicio va aumentando, porque las máquinas van entrando paulatinamente al sistema; pero luego llega el momento a partir del cual el espacio disponible comienza a disminuir, ya que se van consumiendo los espacios disponibles en las estaciones de trabajo por el almacenamiento de archivos.



(a) Espacio Libre

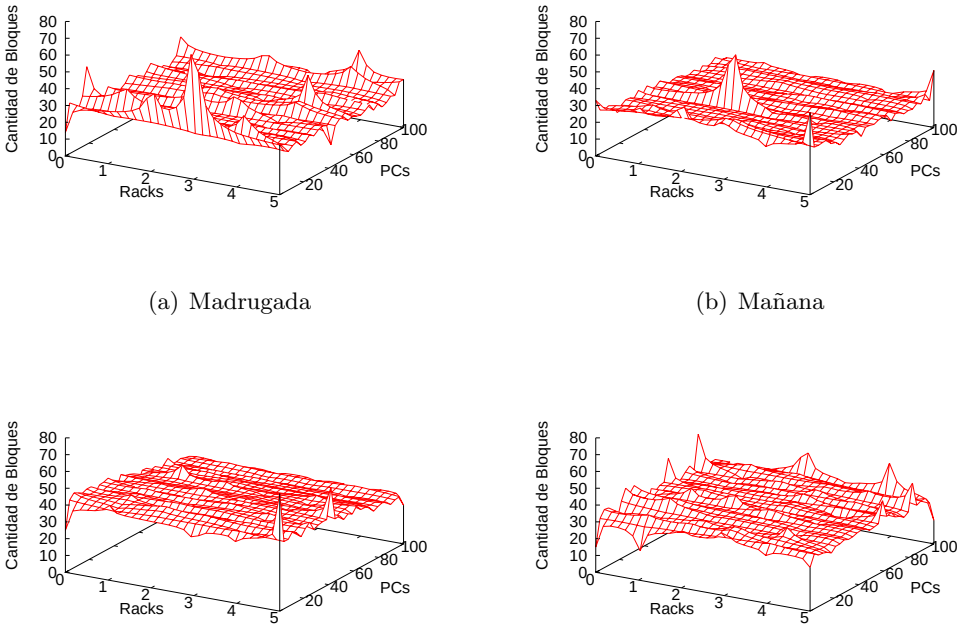
(b) Espacio Utilizado

Figura 3.10: Espacio libre(a) y utilizado(b) del SAD modelado durante las sesiones del día; madrugada (a en las leyendas), mañana (b), tarde (c) y noche (d).

Tal como se puede observar en la figura 3.10(b), donde el espacio utilizado va aumentando hasta llegar a ocupar un total de 736 GB aproximadamente.

3.3.6. Distribución de los bloques de archivos

Para el análisis de cómo fue el proceso de distribución de los bloques de archivos entre los recursos computacionales, la figura 3.11 muestra la cantidad de bloques que se encuentran en cada PC modelada. Como se observa, la estrategia de replicación implementada garantiza un balance adecuando y un buen aprovechamiento de los recursos disponibles, alcanzándose menos picos en el horario de la tarde principalmente ya que es donde hay más estabilidad de recursos computacionales.



(a) Madrugada

(b) Mañana

(c) Tarde

(d) Noche

Figura 3.11: Distribución de los bloques en las PCs durante los horarios de: (a) la madrugada, (b) la mañana, (c) la tarde y (d) la noche

3.4. Conclusiones parciales

En el horario de 8:00 AM a 12:00 M es donde se encuentran disponibles un mayor número de estaciones de trabajo, siendo este horario el seleccionado para realizar las corridas piloto del modelo.

A partir de las corridas piloto se detectó una restricción crítica relacionada con la aceptación o el rechazo del *pipeline* devuelto por el *NameNode* al *ClientNode*, durante el proceso de escritura de datos. Esta restricción, que se encuentra implementada en HDFS, funciona correctamente en un grupo de máquinas dedicadas, pero al ser relajada (nueva estrategia) se logra mejorar el número de respuestas satisfactorias para las peticiones de escritura y lectura de archivos.

Las simulaciones con la nueva estrategia implementada en el momento de aceptar el *pipeline* para el proceso de escritura, haciendo más flexible el criterio de aceptación, logra que los procesos

de escritura y lectura se comporten de una manera aceptable, mostrando un alto porcentaje de solicitudes respondidas satisfactoriamente excepto en los horarios de altas horas de la noche y en la madrugada.

Otros resultados de las simulaciones evidencian que la red se mantiene estable la mayor parte del tiempo, excepto en determinado momentos debido a las frecuentes comunicaciones que tienen los nodos de datos con el servidor. Por otra parte, se alcanza un buen aprovechamiento de la cantidad de espacio total disponible en el sistema.

Conclusiones de la investigación

En el transcurso de la investigación se llegaron a las siguientes conclusiones:

- Al terminar la elaboración del marco teórico conceptual se evidenció, que no han sido explotados los lenguajes gráficos de modelado existentes para el modelado y simulación de un SAD basado en HDFS.
- Se implementó un modelo en el lenguaje de modelado RdPC que permite incorporar las características y disponibilidad de las estaciones de trabajo, que inciden de manera crítica en el comportamiento del SAD, mediante la utilización de parámetros y algunas funciones estadísticas de distribución.
- La validación del modelo mediante la realización de un estudio de caso, permitió comprobar que las cantidades de solicitudes respondidas satisfactoriamente en la simulación se asemejan a las obtenidas en el entorno real.
- A partir de las simulaciones se pudo evaluar el comportamiento del sistema y constatar que se obtiene un alto porcentaje de respuestas satisfactorias para las peticiones de escritura y lectura de archivos, excepto en los horarios de altas horas de la noche y en la madrugada donde los resultados no fueron buenos por la poca disponibilidad de recursos. Otros indicadores reflejan el buen aprovechamiento del espacio disponible en las estaciones de trabajo que forman parte del sistema.
- Al identificar la existencia de una restricción crítica que incide negativamente en el desempeño del sistema y proponer las acciones correctivas, se evidencia la utilidad del modelo desarrollado

en la toma de decisiones para garantizar un correcto funcionamiento.

Recomendaciones

- Incorporar en el código fuente del sistema HDFS las acciones correctivas que relajan las restricciones en el momento de seleccionar el *pipeline* para replicar los bloques, con el fin de garantizar un correcto comportamiento del SAD en un entorno donde no se tiene el control de la disponibilidad de las estaciones de trabajo.
- Desplegar el sistema HDFS en los laboratorios docentes de la UCI luego de que se implementen las acciones correctivas, siendo cada docente un *rack*, para de esta manera aprovechar el espacio disponible de las PCs que se encuentra subutilizado.
- Explorar nuevas configuraciones del sistema en el modelo para comprobar si es posible mejorar su desempeño.

Bibliografía

[1] Dean Klein VPoSMD. History of Digital Storage. December 15, 2008;.

[2] Morris RJT, Truskowski BJ. The evolution of storage systems. IBM Syst J. 2008;42(2):205–217.

[3] Bolosky WJ, Douceur JR, Ely D, Theimer M. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. SIGMETRICS Perform Eval Rev. 2000 Jun;28(1):34–43.

[4] Anderson DP, Fedak G. The Computational and Storage Potential of Volunteer Computing. CCGRID '06; 2006. p. 73–80.

[5] Thanh TD, Mohan S, Choi E, Kim S, Kim P. A Taxonomy and Survey on Distributed File Systems. Networked Computing and Advanced Information Management, International Conference on. 2008;1:144–149.

[6] Anderson DP, Fedak G. The Computational and Storage Potential of Volunteer Computing. In: CCGRID '06: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid. vol. 1. Washington, DC, USA: IEEE Computer Society; 2006. p. 73–80.

[7] Satyanarayanan M. A Survey of Distributed File Systems. Pittsburgh, Pennsylvania; 1989. CMU-CS-89-116.

[8] Levy E, Silberschatz A. Distributed file systems: concepts and examples. ACM Comput Surv. 1990 December;22:321–374.

[9] Storage@home: Petascale Distributed Storage; 2007.

[10] Ghemawat S, Gobioff H, Leung ST. The Google file system. SIGOPS Oper Syst Rev. 2003 December;37(5):29–43.

[11] Decandia G, Hastorun D, Jampani M, Kakulapati G, Lakshman A, Pilchin A, et al. Dynamo: amazon's highly available key-value store. SIGOPS Oper Syst Rev. 2007;41(6):205–220.

[12] Kubiatowicz J, Bindel D, Chen Y, Czerwinski S, Eaton P, Geels D, et al. OceanStore: an architecture for global-scale persistent storage. In: ASPLOS-IX: Proceedings of the ninth international conference on Architectural support for programming languages and operating systems. 5. New York, NY, USA: ACM; 2000. p. 190–201.

[13] Chang F, Dean J, Ghemawat S, Hsieh WC, Wallach DA, Burrows M, et al. Bigtable: A Distributed Storage System for Structured Data. ACM Trans Comput Syst. 2008 June;26(2):1–26.

[14] Bockelman B. Using Hadoop as a grid storage element. Journal of Physics: Conference Series. 2009;180.

[15] Raja FR. A Highly Scalable and Efficient Distributed File Storage System. International Journal of Computer Science and Network Security. 2009 Diciembre;9(12):101–107.

[16] Bilicki V, editor. LanStore: a highly distributed reliable file storage system. .NET Technologies; 2005.

[17] Patil SV. Unified Virtual Storage: Virtualization of Distributed Storage in a Network. International Journal of Computer Applications. 2010;1(22):28–31.

[18] Braam PJ, Others. The Lustre storage architecture. White Paper, Cluster File Systems, Inc, Oct. 2003;23.

[19] Tang H, editor. The Panasas ActiveScale Storage Cluster - Delivering Scalable High Bandwidth Storage. ACM/IEEE conference on Supercomputing; 2004.

[20] Palankar MR, Iamnitchi A, Ripeanu M, Garfinkel S. Amazon S3 for science grids: a viable solution? In: Proceedings of the 2008 international workshop on Data-aware distributed computing. DADC '08. New York, NY, USA: ACM; 2008. p. 55–64.

[21] Borthakur D, Gray J, Sarma JS, Muthukkaruppan K, Spiegelberg N, Kuang H, et al. Apache hadoop goes realtime at Facebook. In: Proceedings of the 2011 international conference on Management of data. SIGMOD '11. New York, NY, USA: ACM; 2011. p. 1071–1080.

[22] Tahoe Homepage (Consultado en Noviembre, 2012);. Available from: (https://tahoe-lafs.org/).

[23] Wilcox-O'Hearn Z, Warner B. Tahoe: the least-authority filesystem. StorageSS '08; 2008. p. 21–26.

[24] Allmydata Homepage (Consultado en Noviembre, 2012);. Available from: http://www.allmydata.com/.

[25] Greenan KM, Miller EL, Wylie JJ. Reliability of flat XOR-based erasure codes on heterogeneous devices; 2010.

[26] Red-Hat (Consultado en Noviembre, 2012);. Available from: https://access.redhat.com/knowledge/docs/en-US/Red_Hat_Storage_Software_Appliance/3.2/html/User_Guide/chap-Administration_Guide-GlusterFS_Client.html.

[27] Písačka J, Hron M, Janky F, Pánek R. Cluster storage for COMPASS tokamak. Fusion Engineering and Design. 2012 Oct;.

[28] Weil SA, Brandt SA, Miller EL, Long DDE, Maltzahn C. Ceph: A scalable, high-performance distributed file system; 2006. p. 307–320. Available from: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.110.4574.

[29] CEPH Homepage (Consultado en Noviembre, 2012);. Available from: http://ceph.com/.

[30] Borthakur D. The Hadoop Distributed File System: Architecture and Design; 2007.

[31] Shvachko K, Kuang H, Radia S, Chansler R. The Hadoop Distributed File System. In: Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST). MSST '10; 2010. p. 1–10.

[32] The Hadoop Distributed File System Homepage (Consultado en Noviembre, 2012);. Available from: http://hadoop.apache.org/hdfs/.

[33] Cassandras CG, Lafortune S. Introduction to Discrete Event Systems. Secaucus, NJ, USA: Springer-Verlag New York, Inc.; 2006.

[34] Review of "Systems Simulation: The Art and Science, by Robert E. Shannon", Prentice-Hall, 1975. SIGSIM Simulation Digest. 1976;7(3):3. Available from: http://dx.doi.org/10.1145/1102746.1102759.

[35] Schruben L. Analytical simulation modeling. In: Proceedings of the 40th Conference on Winter Simulation. WSC '08; 2008. p. 113–121.

[36] Ramadge PJG, Wonham WM. The control of discrete event systems. Proceedings of the IEEE. 1989 Jan;77(1):81–98.

[37] Ben-Naoum L, Boel R, Bongaerts L, De Schutter B, Peng Y, Valckenaers P, et al. Methodologies for discrete event dynamic systems: A survey. Journal A. 1995 Dec;36(4):3–14.

[38] Robinson S, Brooks R, Kotiadis K, Van Der Zee DJ. Conceptual Modeling for Discrete-Event Simulation. 1st ed. Boca Raton, FL, USA: CRC Press, Inc.; 2010.

[39] Sulistio A, Yeo CS, Buyya R. Simulation of Parallel and Distributed Systems: A Taxonomy and Survey of Tools. International Journal of Software Practice and Experience Wiley Press. 2002;p. 1–19.

[40] ClodSim Home Page. Consultado en noviembre 2012;. Available from: `http://cloudbus.org/cloudsim/`.

[41] Calheiros RN, Ranjan R, Beloglazov A, De Rose CAF, Buyya R. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw Pract Exper. 2011 Jan;41(1):23–50.

[42] Calheiros RN, Ranjan R, Rose CAFD, Buyya R. CloudSim: A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services. CoRR. 2009;abs/0903.2525.

[43] MONARC Homepage (Consultado en noviembre 2012);. Available from: `http://monarc.cacr.caltech.edu/www_monarc/monarc_e__Design.htm`.

[44] Dobre C, Pop F, Cristea V. A Simulation Framework for Dependable Distributed Systems. In: Proceedings of the 2008 International Conference on Parallel Processing - Workshops. ICPPW '08; 2008. p. 181–187.

[45] Dobre C, Cristea V, Legrand I. Simulation Framework for Modeling Large-Scale Distributed Systems. CoRR. 2011;abs/1106.6122.

[46] Dobre C, Stratan C. MONARC Simulation Framework. CoRR. 2011;abs/1106.5158.

[47] Dobre C, Cristea V. Simulation Analysis of CMS Data Replication and Production Activities. In: Proceedings of the 2011 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing. 3PGCIC '11. Washington, DC, USA: IEEE Computer Society; 2011. p. 372–377.

[48] Legrand IC, Newman HB. The MONARC toolset for simulating large network-distributed processing systems. In: Proceedings of the 32nd conference on Winter simulation. WSC '00; 2000. p. 1794–1801.

[49] Boteanu A, Dobre C, Pop F, Cristea V. Simulator for fault tolerance in large scale distributed systems. In: Proceedings of the Proceedings of the 2010 IEEE 6th International Conference on Intelligent Computer Communication and Processing. ICCP '10; 2010. p. 443–450.

[50] Beaumont O, Rejeb H. On the importance of bandwidth control mechanisms for scheduling on large scale heterogeneous platforms. In: 2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS). IEEE; 2010. p. 1–12.

[51] Casanova H, Legrand A, Quinson M. SimGrid: A Generic Framework for Large-Scale Distributed Experiments. 2008;p. 126–131.

[52] Merz S, Quinson M, Rosa C. SimGrid MC: verification support for a multi-API simulation platform. In: Proceedings of the joint 13th IFIP WG 6.1 and 30th IFIP WG 6.1 international conference on Formal techniques for distributed systems. FMOODS'11/FORTE'11; 2011. p. 274–288.

[53] Choi S, Lee J, Yu H, Lee H. 56. In: Kim Th, Adeli H, Cho Hs, Gervasi O, Yau SS, Kang BH, et al., editors. Replication and Checkpoint Schemes for Task-Fault Tolerance in Campus-Wide Mobile Grid. vol. 261 of Communications in Computer and Information Science. Berlin, Heidelberg: Springer Berlin Heidelberg; 2011. p. 455–467.

[54] Cornea BF, Bourgeois J. A framework for efficient performance prediction of distributed applications in heterogeneous systems. The Journal of Supercomputing. 2012 Sep;p. 1–26.

[55] Wangikar V, Jain K, Shah S. TST: a job scheduling algorithm in computational grid. In: Proceedings of the International Conference & Workshop on Emerging Trends in Technology. ICWET '11; 2011. p. 467–469.

[56] Borovska P, Aleksieva-Petrova A, Gancheva V, Beshkov S. Grid resource management using lottery scheduling. In: Proceedings of the 12th International Conference on Computer Systems and Technologies. CompSysTech '11; 2011. p. 253–258.

[57] Nandagopal M, Gokulnath K, Uthariaraj VR. Sender initiated decentralized dynamic load balancing for multi cluster computational grid environment. In: Proceedings of the 1st Amrita ACM-W Celebration on Women in Computing in India. A2CWiC '10; 2010. p. 63:1–63:4.

[58] Sashi K, Thanamani AS. A new dynamic replication algorithm for European data grid. In: Proceedings of the Third Annual ACM Bangalore Conference. COMPUTE '10; 2010. p. 17:1–17:4.

[59] Jain K, Vidhate AV, Wangikar V, Shah S. Design of file size and type of access based replication algorithm for data grid. In: Proceedings of the International Conference & Workshop on Emerging Trends in Technology. ICWET '11; 2011. p. 315–319.

[60] Niu J, Zou J, Ren A. OOPN: An Object-Oriented Petri Nets and its Integrated Development Environment. In: Proceedings of 2003 IASTED International Conference on Software Engineering and Applications (SEA'2003). Marina del Ray, USA: ACTA Press; 2003. .

[61] Jensen K. Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use (Volume 1). vol. 1 of EATCS Series. Springer Verlag; 2003.

[62] Kristensen LM, Jrgensen JB, Jensen K. Application of Coloured Petri Nets in System Development; 2003. .

[63] Roy N, Dabholkar A, Hamm N, Dowdy LW, Schmidt DC. Modeling Software Contention using Colored Petri Nets. In: Miller EL, Williamson CL, editors. MASCOTS. IEEE Computer Society; 2008. p. 317–324. Available from: `http://dblp.uni-trier.de/db/conf/mascots/mascots2008.html#RoyDHDS08`.

[64] Jensen K. An introduction to the practical use of coloured Petri Nets; 1998. p. 237–292.

[65] Kristensen LM, Christensen S, Jensen K. The practitioner's guide to coloured Petri nets. International Journal on Software Tools for Technology Transfer. 1998;2:98–132.

[66] Tjensvold JM. Petri net modeling and simulation of a distributed computing system; 2007.

[67] Trcka N, van der Aalst WMP, Bratosin C, Sidorova N. Evaluating a Data Removal Strategy for Grid Environments Using Colored Petri Nets; 2008. .

[68] Bratosin C, Aalst W, Sidorova N, Trčka N. A Reference Model for Grid Architectures and Its Analysis. In: Proceedings of the OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008. Part I on On the Move to Meaningful Internet Systems:. OTM '08; 2008. p. 898–913.

[69] Beaudouin-Lafon M, Mackay WE, Andersen P, Janecek P, Jensen M, Lassen HM, et al. CPN/-Tools: A Post-WIMP Interface for Editing and Simulating Coloured Petri Nets. In: Proceedings of the 22nd International Conference on Application and Theory of Petri Nets. ICATPN '01. London, UK, UK: Springer-Verlag; 2001. p. 71–80.

[70] Ratzer A, Wells L, Lassen H, Laursen M, Qvortrup J, Stissing M, et al. CPN Tools for Editing, Simulating, and Analysing Coloured Petri Nets; 2003. p. 450–462.

[71] Wells L. Performance analysis using CPN tools. In: valuetools '06: Proceedings of the 1st international conference on Performance evaluation methodolgies and tools. New York, NY, USA: ACM; 2006. .

[72] Jensen K, Kristensen LM, Wells L. Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems. Int J Softw Tools Technol Transf. 2007 May;9(3):213–254.

[73] CPN Tools Homepage. Consultado en noviembre 2012;. Available from: `http://cpntools.org/`.

[74] Störrle H. An Evaluation of High-end Tools for Petri-nets. Bericht // Institut für Informatik, Ludwig-Maximilians-Universität München. Univ., Inst. für Informatik; 1998.

[75] Hrz B, Zhou MC. Modeling and Control of Discrete-event Dynamic Systems: with Petri Nets and Other Tools. 1st ed. Springer Publishing Company, Incorporated; 2007.

[76] Christensen S, Haagh TB. Design/CPN Overview of CPN ML Syntax; 1996. University of Aarhus.

[77] gnu Homepage. Consultado en noviembre 2012;. Available from: `http://www.gnuplot.org`.

[78] Janert PK. Gnuplot in Action: Understanding Data with Graphs. Greenwich, CT, USA: Manning Publications Co.; 2009.

[79] Racine J. gnuplot 4.0: a portable interactive plotting utility. Journal of Applied Econometrics. 2006;21(1):133–141.

[80] Hey T, Tansley S, Tolle K, editors. The Fourth Paradigm: Data-Intensive Scientific Discovery. Redmond, Washington: Microsoft Research; 2009.

[81] Gantz JF, Reinsel D, Chute C, Schlichting W, Mcarthur J, Minton S, et al. IDC - The Expanding Digital Universe: A Forecast of Worldwide Information Growth Through 2010;.

[82] Gantz JF, Chute C, Msnfrediz A, Minton S, Reinsel D, Schlichting W, et al. IDC - El Universo Digital, diverso y en expansión acelerada : Un pronóstico actualizado del crecimiento de la información en el mundo hasta el 2011; 2008.

[83] The Economist. A special report on managing information: Data, data everywhere. The Economist. 2010 February;.

[84] The 15-petabyte network and the atom smasher. Zdnet Asia. 2008;(PRESSCUT-V-2009-762).

[85] Shvachko K, Kuang H, Radia S, Chansler R. The Hadoop Distributed File System. In: 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST). IEEE; 2010. p. 1–10.

[86] Porter G. Decoupling Storage and Computation in Hadoop with SuperDataNodes. Operating Systems Review. 2010 Abril;44(2):41–46.

[87] Bolosky WJ, Douceur JR, Ely D, Theimer M. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. SIGMETRICS Perform Eval Rev. 2000 Jun;28(1):34–43.

[88] Beisecker E, Balestrini S, Mavris D. Modeling a new connectors contribution to humanitarian relief response using discrete event simulation. In: Proceedings of the 2011 Grand Challenges on Modeling and Simulation Conference. GCMS '11; 2011. p. 213–221.

[89] Seok MG, Kim TG. Parallel discrete event simulation for DEVS cellular models using a GPU. In: Proceedings of the 2012 Symposium on High Performance Computing. HPC '12; 2012. p. 11:1–11:7.

[90] Raunak MS, Osterweil LJ, Wise A. Developing discrete event simulations from rigorous process definitions. In: Proceedings of the 2011 Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium. TMS-DEVS '11; 2011. p. 117–124.

[91] Dimakis AG, Ramchandran K, Wu Y, Suh C. A Survey on Network Codes for Distributed Storage. Proceedings of the IEEE. 2010;99(3):13.

[92] Lamehamedi H, Shentu Z, Szymanski B, Deelman E. Simulation of Dynamic Data Replication Strategies in Data Grids. In: Proceedings of the 17th International Symposium on Parallel and Distributed Processing. IPDPS '03; 2003. p. 100.2–.

[93] Paulson LC. ML for the working programmer (2nd ed.). New York, NY, USA: Cambridge University Press; 1996.

[94] Christensen S, Haagh TB. Design/CPN Overview of CPN ML Syntax; 1996. University of Aarhus.

Anexo A

A.1. Modelado de la sub-transición que genera y envía las solicitudes

A.2. Modelado de la sub-transición que recibe las respuestas de las solicitudes

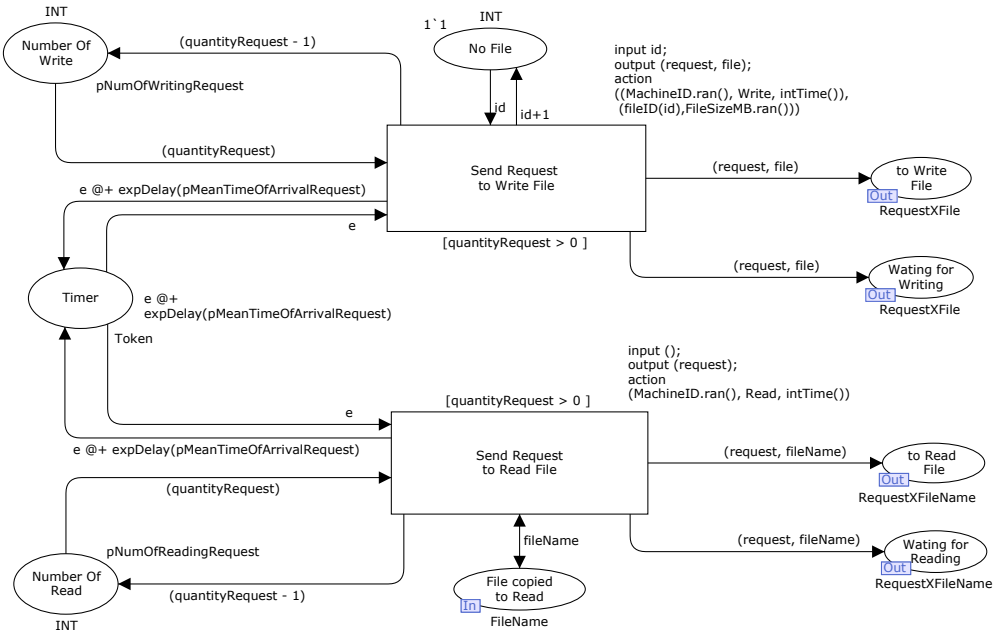


Figura A.1: Modelado de la sub-transición que genera y envía las solicitudes

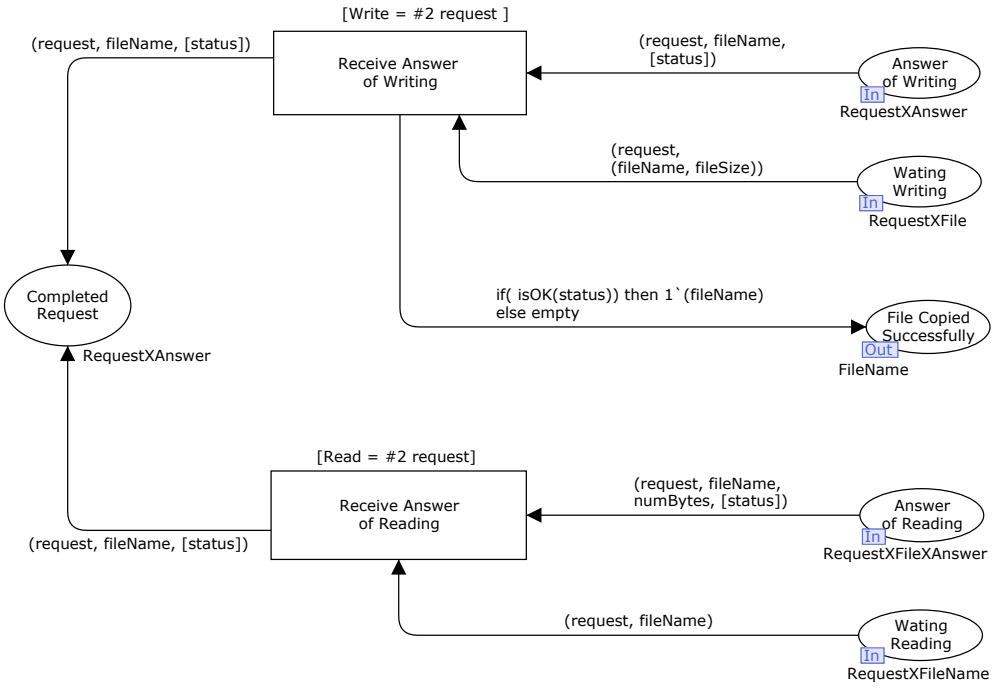


Figura A.2: Modelado de la sub-transición que recibe las respuestas de las solicitudes