Centro de Investigación y de Estudios
Avanzados del Instituto Politécnico Nacional

Unidad Zacatenco
Departamento de Computación

Framework para manejo distribuido de sistemas expertos

Tesis que presenta

Guillermo Alejandro Barrera Granados

para obtener el Grado de

Maestro en Ciencias en Computación

Director de la Tesis

Dr. Amilcar Meneses Viveros

México, D.F. Noviembre 2014

Resumen

Los sistemas expertos son herramientas utilizadas cuando se requiere tomar decisiones, ya que poseen inteligencia artificial y una capacidad de razonamiento parecida a la humana, necesitan conocimiento y experiencia de expertos humanos del área en donde serán utilizados. Los sistemas expertos sirven de apoyo a los expertos humanos encargados de controlar, monitorizar y administrar sistemas de diversa índole; cuando se utilizan como herramientas auxiliares para monitorizar sistemas grandes o con una gran cantidad de eventos, deben ser capaces de escalar a medida que crecen los sistemas que supervisan.

En esta tesis se presenta una arquitectura distribuida que permite añadir a un sistema experto las siguientes capacidades: escalabilidad, alta disponibilidad y tolerancia a fallas. La arquitectura distribuida está diseñada para distribuir los servicios del sistema entre múltiples instancias, asegurando que siempre exista una instancia que sea capaz sustituir a cualquier otra que ha fallado. De esta manera, un sistema experto puede tener algunas de las características de los sistemas distribuidos.

La arquitectura propuesta encapsula los mecanismos de comunicación del sistema distribuido y evita que el usuario tenga que lidiar con esta capa. En esta propuesta la arquitectura distribuida se compone de *TAO*, que es una implementación de la especificación de *CORBA* para sistemas distribuidos, también incorpora la biblioteca *Boost* para permitir el trabajo concurrente en cada uno de los módulos del sistema y utiliza una interfaz del tipo *REST* para establecer la comunicación con el usuario. El diseño de la arquitectura define el uso de una plataforma de distribución de archivos que tiene por objetivo permitir la distribución de información rápidamente desde las instancias de acceso a todas las instancias del tipo experto mediante una comunicación *Multicast*.

Abstract

Expert systems are tools used when it is required to take decisions, since they have artificial intelligence and human-like reasoning capacity, they need expertise and knowledge of human experts in the area where it will be used; they serve as support to the human experts to monitor and manage systems of various kinds. An expert system that is used as an auxiliary tool in large systems or with lots of events, needs to be able to scale as the supervised system grows.

This thesis presents a distributed architecture that allows to add to an expert system the following capabilities: scalability, high availability and fault tolerance. The distributed architecture is designed to distribute the services of the system among multiple instances, ensuring that there is always an instance that can replace any other failed. In this way, an expert system could have some features of the distributed systems.

The proposed architecture encapsulates the communication mechanisms of the distributed system and saves the user from having to deal with this layer. In this proposal the distributed architecture is composed of *TAO*, which is an implementation of the *CORBA* specification for distributed systems, also incorporates the Boost library to enable a concurrent work on each of the modules of the system and uses an interface type *REST* to establish communication with the user. The design of the architecture defines the use of a platform for distribution of files which aims to enable distribute information quickly from access instances to all the experts instances using a *Multicast* communication.

Dedicado a mi familia y futura familia

A quienes me han dado la vida en toda la extensión de la palabra.
A quienes me han brindado y brindarán su amor, alegría, cariño y todas esas emociones que nos hacen humanos.
A quienes que han estado y estarán en los momentos de felicidad, así como en los momentos difíciles y desafiantes.
Por quienes me esfuerzo y trabajo para tener una mejor vida.
A ustedes: Rocio mi madre, Guillermo mi padre, Jared mi hermano y Viri mi futura esposa, toda mi familia y los que vengan en el futuro... ***Gracias****.*

Agradecimientos

Doy gracias a Dios por haber permitido nuestra existencia en el universo y permitirnos descubrir las cosas tan magnificas que en el existen.
Agradezco el amor y apoyo incondicional de mi familia; gracias a ti madre y padre, quienes me ayudaron a tener una educación y con ella trabajar para ser un mejor ser humano. Te doy gracias a ti Jared, mi hermano, por compartir conmigo tantos momentos de diversión.
Viri, mi futura esposa; te agradezco inmensamente por todo el amor y alegría que le das a mi vida, por soportar conmigo la dificultad para obtener este logro en la vida.
Le doy gracias a cada miembro de mi familia, quienes me apoyaron en todo aspecto.
Quiero agradecer a mi director de tesis, el doctor Amilcar Meneses Viveros, por su guía, apoyo, y paciencia para el desarrollo del trabajo de esta tesis; por haber compartido su conocimiento conmigo y por su tiempo que me brindo.
Les agradezco a los sinodales de esta tesis; el doctor José Guadalupe Rodríguez García y el doctor Dominique Decouchant, quienes me brindaron su ayuda y conocimiento en la revisión de esta tesis.
Quiero dar las gracias al Centro de Investigación y Estudios Avanzados del I.P.N, por darme la oportunidad de ser parte de su gran programa de posgrado, por permitirme avanzar en mi vida académica y por todo el apoyo que nos brinda en nuestra vida estudiantil. Quiero reconocer a todo el personal de apoyo del departamento de computación, quienes nos ayudan y apoyan para hacer nuestra vida estudiantil un poco más asequible.
Le doy gracias al Consejo Nacional de Ciencia y Tecnología, por al apoyo económico que me brindo, él cual me permitió lograr esta meta académica.
A todos... Gracias.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

El propósito de esta tesis es determinar si es posible agregar a un sistema experto las siguientes características: Escalabilidad, tolerancia a fallas y alta disponibilidad; propias de un sistema distribuido mediante el uso de una arquitectura distribuida. Agregar estas características a un sistema experto, puede incrementar los beneficios que brinda a sus usuarios, pero esto depende de los objetivos o tareas para los que se utiliza dicho sistema experto.

El sistema de la arquitectura distribuida debe tener la capacidad de distribuir la carga de trabajo entre las instancias que lo conforman, esto se logra mediante el uso de varias políticas de asignación de carga; lo cual le proporciona al sistema distribuido la capacidad de cambiar en tiempo de ejecución la política de asignación en uso.

La arquitectura distribuida está diseñada para ser lo más independiente posible del sistema experto asociado; esto tiene como finalidad, que sea fácil cambiar el sistema experto utilizado por otro, sin requerir grandes cambios en la arquitectura distribuida. Es necesario que el sistema experto que se desee utilizar como reemplazo tenga algunas características específicas, permitiendo el correcto funcionamiento de la arquitectura distribuida.

1.1. Motivación

Los sistemas expertos son herramientas utilizadas cuando se requiere tomar decisiones, ya que poseen inteligencia artificial y una capacidad de razonamiento parecida a la humana, necesitan conocimiento y experiencia de expertos humanos del área en donde serán utilizados [1]. Los sistemas expertos sirven de apoyo a los expertos humanos encargados de controlar, monitorizar y administrar sistemas de diversa índole, algunos son de importancia para sus usuarios, lo que tiene como consecuencia que estos sistemas deban estar en funcionamiento permanente, como ejemplo tenemos a los sistemas para la administración de redes de datos del tipo WAN [2], sistemas de energía eléctrica [3, 4], sistemas industriales [5], sistemas de telecomunicaciones [6], sistemas de transporte [7], entre otros; en ocasiones llegan a tener un gran tamaño, abarcar una gran extensión y contar con muchos componentes, presentar alguna falla en cualquier momento y en cualquiera de sus componentes, por lo que es necesario que sean monitoreados constantemente. Existen momentos en los cuales son de vital importancia los servicios que proveen estos sistemas, en donde no hay opción de interrupciones o fallas, por lo cual los sistemas deben ser vigilados de forma intensiva y constante, pues en caso de ocurrir algún incidente es necesario tomar medidas para restablecer dichos servicios y que los usuarios no noten los problemas de los proveedores de los servicios.

Para poder monitorizar sistemas de gran tamaño y extensión, es necesario tener una gran cantidad de personas con conocimiento en el sistema y en ocasiones no es suficiente, entonces se utilizan sistemas de monitoreo que tienen en su interior sistemas expertos para que de manera automática se pueda determinar la causa de la falla y posiblemente una solución a un problema, basándose en la información que el sistema de monitoreo provee al sistema experto en su interior. Los sistemas expertos tienen el conocimiento acerca de un sistema o campo y la inteligencia para poder resolver problemas en situaciones en donde es necesario una solución correcta y con un tiempo de respuesta que solo una computadora puede cumplir [1, 8].

1.2. Planteamiento del problema

Los sistemas expertos son utilizados en muchas áreas y para diversos fines, por ejemplo, en sistemas auxiliares para la toma de decisiones y en el interior de los sistemas de monitoreo; sin embargo pueden presentar problemas cuando son utilizados en situaciones de alta carga de trabajo con un corto tiempo de respuesta, también cuando el número de usuarios se incrementa. Existen sistemas expertos que se integran en algunos sistemas de monitoreo, cuyo objetivo es auxiliar a los encargados a determinar el origen de una falla o elegir la mejor acción a tomar en ciertas situaciones de inestabilidad o falla de los sistemas que se encargan de monitorizar, mediante la información recolectada por el sistema de monitoreo contenedor. Es posible que estos sistemas expertos presenten fallas o intermitencias de los servicios que proveen cuando son utilizados en el monitoreo de sistemas de gran tamaño, o con una gran cantidad de eventos; en estos escenarios los sistemas expertos deben tener la capacidad de escalarse conforme se van incrementando las peticiones que atender [1, 8].

Un sistema experto que se utiliza como auxiliar en el monitoreo de sistemas de gran tamaño o con gran cantidad de eventos necesita poder escalar conforme el sistema de monitoreo crece, también es necesario que sea tolerante a fallas y tenga una alta disponibilidad de los servicios que provee; las características anteriores son los beneficios que tienen los sistemas distribuidos. Estas características son las que se quiere agregar a un sistema experto en el presente trabajo de tesis, mediante el diseño e implementación de una arquitectura distribuida.

Implementar un sistema experto distribuido presenta grandes problemas, tales como, la distribución de la información, la sincronización entre los componentes, el control de temporizadores de eventos, entre otros. Existen sistemas expertos distribuidos que utilizan sistemas de memoria compartida [3] para permitir que los componentes se comuniquen, pero ésto representa un gran problema de escalabilidad [9]. La alternativa es el uso del mecanismo de paso de mensajes, pero ésto complica más los problemas de la implementación de un sistema experto distribuido, es en esta capa en donde se integrará arquitectura distribuida que se propone para proveer de las capacidades de un sistema distribuido a un sistema experto sin tener que hacer grandes modificaciones a éste.

1.3. Objetivos

El propósito de esta tesis es determinar si es posible agregar a un sistema experto las siguientes características: Escalabilidad, tolerancia a fallas y alta disponibilidad; mediante la utilización de una arquitectura distribuida. Por tal razón, es importante dividir dicho propósito en varios objetivos, mediante los cuales se podrá lograr. A continuación se detalla el objetivo general y los objetivos específicos de esta tesis.

1.3.1. General

Determinar si es posible agregar las características: Escalabilidad, tolerancia a fallas y alta disponibilidad; propias de un sistema sistema distribuido, a un sistema experto, mediante la utilización de una arquitectura distribuida. A continuación se detalla el objetivo general y los objetivos específicos de esta tesis.

1.3.2. Particulares

1. Determinar si el *Middleware* basado en la especificación CORBA [10–12], para la implementación de sistemas distribuidos, provee las herramientas necesarias para la creación de la arquitectura distribuida que se propone.

2. Determinar la mejor estrategia para la utilización de un sistema gestor de bases de datos en la arquitectura distribuida propuesta, con el objetivo de respaldar toda la información del funcionamiento del sistema; la estrategia debe considerar poder utilizar distintas implementaciones de sistemas gestores sin requerir modificaciones en la arquitectura distribuida.

3. Determinar si es posible utilizar el sistema experto CLIPS [13], en la arquitectura distribuida que se propone.

4. Determinar las características que necesita un sistema experto para poder ser utilizado con la arquitectura distribuida que se propone.

5. Determinar las instancias necesarias para que la arquitectura distribuida propuesta pueda funcionar correctamente, así como determinar las responsabilidades que debe tener cada una.

6. Determinar las estrategias necesarias para que el sistema de la arquitectura distribuida, sea fácil de instalar y administrar.

7. Determinar las reglas para el distribución de la carga en los nodos de inferencia; es importante tener la capacidad de cambiar la regla utilizada, en tiempo ejecución, sin tener que reiniciar el sistema de la arquitectura distribuida.

1.4. Justificación

Un sistema experto es una poderosa herramienta, pues es capaz de ayudar en la resolución de una amplia variedad de problemas, en donde solo es necesario entrenar o transferir el conocimiento a dicho sistema. Mediante la incorporación de una capa inferior, conformada por una arquitectura distribuida, se brindan algunas cualidades de un sistema distribuido a un sistema experto.

La arquitectura distribuida está diseñada para repartir las obligaciones del sistema entre varias instancias, asegurando que siempre exista una instancia que pueda reemplazar a alguna que haya fallado, de esta manera, se permite que un sistema experto tenga algunas de las características de los sistemas distribuidos. El arquitectura se compone por distintos tipos de nodos. Algunos nodos actúan como el acceso al sistema de la arquitectura mediante peticiones *REST* [14]. Algunos nodos son los encargados de la administración y el control del sistema distribuido, también tienen la capacidad de conectarse a un sistema gestor de bases de datos para el almacenamiento de la información de trabajo para su posterior recuperación. Otros nodos son los encargados del proceso de inferencia, este tipo de nodo puede tener asociado uno o más procesos con un sistema experto. La arquitectura encapsula los mecanismos del sistema distribuido y evita que el usuario tenga que lidiar con esta capa. Con este diseño se tiene la intención de asegurar que en caso de que alguna parte del sistema falle, la información se puede recuperar por cualquier otro nodo capaz realizar las funciones del nodo que falló.

La arquitectura está implementada utilizando *TAO* [15], el cual es una implementación de la especificación *CORBA* [10–12] para sistemas distribuidos que son implementados utilizando el paradigma de Programación Orientados a Objetos [16]. *CORBA* permite qué los nodos no necesiten una configuración previa con el listado

de nodos con los que trabajarán, esto es posible mediante la utilización del servicio de nombres de *CORBA*, el cual permite saber qué otros nodos se encuentran listos para trabajar en la arquitectura. *CORBA* también permite ejecutar funciones de objetos remotos de manera transparente, solo es necesario tener la referencia o identificador de los objetos, así cada nodo se puede comunicar con los demás nodos dejando a *CORBA* que realice todo el trabajo de resolución y conexión. También se incorpora la biblioteca *Boost* [17] para permitir un trabajo concurrente en cada uno de los módulos del sistema; se utiliza el patrón grupo de hilos de ejecución, el cual tiene como estrategia la creación de todos los hilos que se utilizarán en tiempo de ejecución y solo se asignan conforme se solicitan.

Los sistemas expertos son utilizados en diversos campos; como en la microbiología clínica [18], optimización numérica [19], sistemas ferroviarios [20], telecomunicaciones [21], entre otros; cuando el sistema experto se encuentra en un único dispositivo, y si llega a fallar puede perderse el acceso al servicio y el conocimiento del sistema experto. Para afrontar las situaciones de pérdida ante una falla en un sistema experto, se crearon los sistemas expertos distribuidos, en los cuales el procesamiento y conocimiento se encuentran distribuidos en varios dispositivos; existen varias formas en las que son implementados:

- Sistemas expertos distribuidos basados en sistemas expertos que se encuentran distribuidos en la Web y se comunican mediante el protocolo de Internet, los sistemas expertos cooperan entre sí durante el proceso de inferencia [22].
- Sistemas expertos distribuidos conformados por varios módulos que contienen un sistema experto en su interior, los cuales no interactúan durante todo el proceso de inferencia [6].
- Sistemas expertos distribuidos constituidos por varios nodos, los cuales consisten en computadoras con un sistema experto embebido, componentes adicionales con funciones especializadas y dispositivos de comunicación; bajo este esquema, cada nodo puede colaborar con los demás durante el proceso de inferencia. [4, 5, 23].
- Sistemas expertos distribuidos basados en una arquitectura con sistemas multiagentes, en donde cada nodo posee un sistema agente capaz de resolver pro-

blemas de manera local; pero cuando es necesario, son capaces de cooperar entre sí para resolver problemas de mayor complejidad que involucran a más de un nodo [2, 7, 24].

- Sistemas expertos distribuidos diseñados utilizando una arquitectura orientada a servicios (SOA, por sus siglas en inglés); en donde los sistemas expertos que los componen son servicios implementados mediante servicios Web (WS, por sus siglas en inglés), y publicados en la nube [25].

Aunque los sistemas expertos y los sistemas expertos distribuidos pueden proporcionar ventajas cuando son implementados de forma correcta [26], no se han popularizado debido a que existen limitantes de diversa índole que han evitado que tengan un mayor grado de utilización [27]. Aun así, existen varios trabajos de investigación, como los mencionados anteriormente, lo que genera cierto grado de confianza acerca de la dirección a la que pretende avanzar el presente trabajo de tesis.

1.5. Organización de tesis

La estructura del contenido del presente trabajo de tesis es el siguiente. En el capítulo 2 se habla del marco teórico relacionado con el campo de investigación de esta tesis, se abordan algunos temas relacionados. El capítulo 3 explica el análisis y diseño, de la arquitectura distribuida que se propone, detallando cada una de las instancias que la conforman; también se explica las funciones y responsabilidades que tienen, así como la forma en que interactúan entre sí, para lograr el correcto funcionamiento del sistema distribuido. En el capítulo 4 se explica cómo se mantiene el control de los nodos del sistema distribuido, se detalla la lógica en la que se basa los mecanismos de administración que utilizan los nodos de acceso para controlar el funcionamiento del sistema. En el capítulo 5 se aborda el funcionamiento del servidor REST, al cual recibe las peticiones de los usuario, también se definen los comandos que entiende y responde. El capítulo 6 tiene como objetivo explicar el diseño y funcionamiento de la plataforma de distribución de archivos utilizada en la arquitectura propuesta, cuyos servicios tienen como propósito la distribución de los

archivos de hechos y scripts, a todos los nodos expertos en el sistema distribuido. El capítulo 7 está enfocado en el mecanismo de comunicación con el proceso experto; en los nodos expertos existe uno o más procesos expertos, con los cuales se comunican el proceso del nodo experto mediante un *Socket* local. El capítulo 8 está dedicado a las pruebas de la arquitectura distribuida. El capítulo 9 contiene las conclusiones del trabajo de tesis y el trabajo futuro a realizar con respecto a lo desarrollado. En la figura 1.1 se puede observar un diagrama de la organización del presente trabajo de tesis.

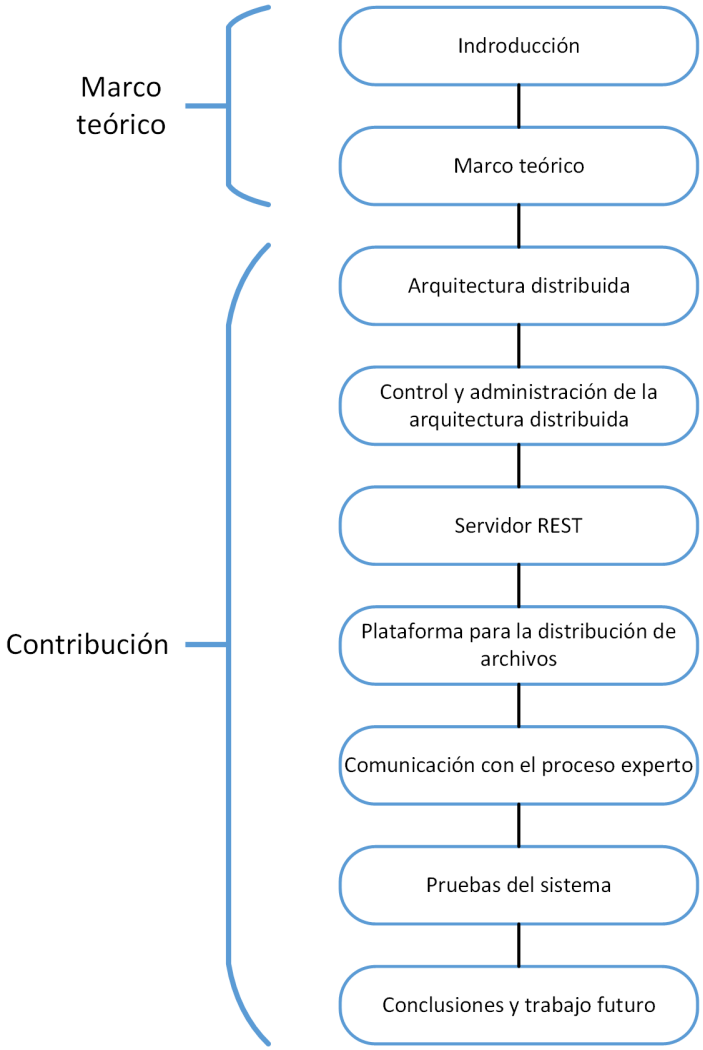

Figura 1.1: Organización del contenido de la tesis.

Capítulo 2

Marco teórico

En este capítulo se abordan los temas y conceptos necesarios para entender mejor el diseño de la arquitectura que se propone. En la sección 2.1, se habla acerca de los sistemas distribuidos y algunos conceptos relacionados. En la sección 2.2, se hace referencia de los sistemas expertos y sistemas expertos distribuidos. Y por último, en la sección 2.3, se hace un breve análisis de algunos trabajos relacionado en la literatura.

2.1. Sistemas distribuidos

El objetivo del presente trabajo de tesis es el análisis y diseño de una arquitectura distribuida, por esta razón es necesario el entendimiento de los sistemas distribuidos y algunos temas relacionados. Este sección está dividida en las siguientes subsecciones. En la sección 2.1.1 se habla acerca de los *sistemas distribuidos*; se mencionan algunas definiciones, características, clasificaciones, fortalezas y debilidades. La subsección 2.1.2 se refiere a las *capas de software* en un sistema distribuido y algunas definiciones relacionadas. En la subsección 2.1.3 se listan algunas *arquitecturas de sistemas* que se han desarrollado en el campo de los sistemas distribuidos. La subsección 2.1.4 explica brevemente el término *Concurrencia* y algunos tópicos necesarios para su entendimiento.

2.1.1. Sistemas distribuidos

Los sistemas computacionales se encuentran en constante cambio, en 1945 cuando inició la computación moderna, las computadoras eran grandes y caras. Sin embargo, hacía la mitad de la década de 1980, dos avances en la tecnología comenzaron a cambiar esta situación. El primero de estos avances fue el desarrollo de poderosos microprocesadores a menores precios, el segundo principalmente fue el desarrollo de la redes de computadoras de alta velocidad, las redes de área local (*LAN* por sus siglas en inglés, *local area networks*) y las redes de área amplia (*WAN* por sus siglas en inglés, *wide area eetworks*). El beneficio de estas tecnologías es que ahora es factible y relativamente fácil poner a trabajar sistemas de cómputo compuestos por grandes cantidades de computadoras interconectadas mediante una red de alta velocidad; a esto se le conoce como **sistemas distribuidos** [9].

Existen distintas definiciones de sistema distribuido, podemos destacar la de Tanenbaum y Steen: "Un sistema distribuido es una colección de computadoras independientes que dan al usuario la impresión de constituir un único sistema coherente" [9]. De esta definición se pueden notar diversos aspectos importantes los cuales se enlistan a continuación:

1. Un sistema distribuido está constituido de computadoras autónomas.

2. Los usuarios, los cuales pueden ser personas o programas, pueden inferir qué interactúan con un sistema único o tradicional.

3. El punto anterior implica la necesidad de que los componentes autónomos, de los que está constituido el sistema distribuido, trabajen en conjunto para poder satisfacer las necesidades de los usuarios.

Otra definición de un sistema distribuido bastante significativa es la de Coulouris, Dolimore y Kindberg, la cual dice: "Un sistema distribuido es aquel en el que los componentes de *hardware* o software, localizados en distintas computadoras unidas mediante red, comunican y coordinan sus acciones sólo mediante el paso de mensajes" [28]. Algunas características de los sistemas distribuidos son: concurrencia, inexistencia de reloj global, fallas independientes, entre otras [28].

Un sistema distribuido debe ser implementado cuando existe algún problema que puede ser resuelto de una manera eficaz y eficiente, sin embargo, no todos los problemas deben ser resueltos con el uso de un sistema distribuido. Las características de un sistema distribuido son [9, 28]:

1. **Accesibilidad:** la principal característica de un sistema distribuido es facilitar a los usuarios el acceso a los recursos remotos y compartirlos una manera controlada y eficiente. La conexión de usuarios y recursos facilita la colaboración e intercambio de información, como ejemplo de éxito se puede mencionar a Internet.

2. **Heterogeneidad:** en un sistema distribuido la heterogeneidad se aplica en las redes, *hardware*, sistemas operativos, lenguajes de programación, entre otros. Los tipos de datos pueden representarse de diferente manera en distintas clases de *hardware*; hay que tratar con estas diferencias de representación si se va a intercambiar mensajes entre programas que se ejecutan en diferente *hardware*.

3. **Transparencia:** en un sistema distribuido es importante ocultar el hecho de que sus procesos y recursos se encuentran físicamente distribuidos en múltiples computadoras. Un sistema distribuido es **transparente** si es capaz de presentarse ante los usuarios como si se tratara de un sistema en una sola computadora.

4. **Grado de apertura:** un **sistema distribuido abierto** es un sistema que ofrece servicios de acuerdo con las reglas estándar que describen la sintaxis y semántica de dichos servicios; dichas reglas se formalizan mediante protocolos. En los sistemas distribuidos, por lo general, los servicios son especificados a través de **interfaces**, las cuales frecuentemente se definen como un **lenguaje de definición de interfaz** (**IDL**, por sus siglas en inglés).

5. **Extensibilidad:** es una característica que determina si el sistema puede ser extendido y reimplementado en diversos aspectos. Es determinada por el grado en el cual se pueden añadir nuevos servicios, así como nuevos recursos para ponerlos a la disposición de una variedad de usuarios.

6. **Seguridad:** entre los recursos de información que se ofrecen y se mantienen en los sistemas distribuidos, muchos tienen un alto valor e importancia para los usuarios, por tal razón su seguridad es de considerable importancia. La seguridad de los recursos de información tiene tres componentes: confidencialidad (protección contra el descubrimiento por individuos no autorizados), integridad (protección contra la alteración o corrupción) y disponibilidad (protección contra interferencia con los procedimientos de acceso a los recursos).

7. **Escalabilidad:** es una de las características más importantes de un sistema distribuido. Un sistema es **escalable** si conserva su efectividad cuando ocurre un incremento significativo en el número de recursos o el número de usuarios. La escalabilidad de un sistema se puede medir de acuerdo con al menos tres dimensiones:

   a) un sistema puede ser escalable con respecto a su tamaño, lo cual significa que se le pueden agregar fácilmente usuarios y recursos;

   b) un sistema escalable geográficamente es aquel en que los usuarios y recursos se pueden encontrar a distancias muy grandes entre ellos;

   c) un sistema puede ser escalable administrativamente, lo qye se significa que el sistema puede ser fácil de administrar incluso si involucra muchas organizaciones administrativas diferentes.

   Desafortunadamente, con frecuencia, un sistema escalable en alguna o algunas de estas dimensiones muestra pérdida de rendimiento al escalarlo.

8. **Tratamiento de Fallas:** cuando aparecen fallas en el *hardware* o *software*, los sistemas pueden producir resultados incorrectos o pueden detenerse antes de concluir su tarea de procesamiento. Las fallas en un sistema distribuido son parciales; algunos componentes fallan mientras otros siguen funcionando. Algunas técnicas para tratar fallas son:

   a) **Detección de fallas:** algunos fallas son detectables.

   b) **Enmascaramiento de fallas:** algunas fallas que han sido detectadas pueden ocultarse o atenuarse.

c) **Tolerancia a fallas:** sistemas distribuidos en donde sus clientes pueden diseñarse para tolerar ciertas fallas, lo que implica que también los usuarios tendrán que tolerarlos generalmente.

d) **Recuperación frente a fallas:** la recuperación implica el diseño de un software en el que, tras la caída del servidor, el estado de los datos pueda reponerse o restaurarse (*roll back*) a una situación anterior.

e) **Redundancia:** puede lograr que los servicios toleren fallas mediante el uso redundante de componentes.

f) **Concurrencia:** los servicios y recursos provistos por los sistemas distribuidos pueden compartirse entre los usuarios. Existe la posibilidad de que varios usuarios intenten acceder al mismo recurso compartido al mismo tiempo, lo cual puede generar inconsistencia en el recurso. Para que un recurso sea seguro en un entorno concurrente, sus operaciones deben sincronizarse de forma que sus datos permanezcan consistentes, lo cual puede lograrse mediante el empleo de técnicas conocidas como los semáforos, consulte la sección 2.1.4.

Aunque parezca que los sistemas distribuidos pueden ser la solución a la mayoría de los problemas computacionales, existen muchas cuestiones a considerar durante la etapa del diseño, pues si son omitidas pueden ocasionar que la complejidad del sistema se incremente de una manera innecesaria, generando muchos errores en el sistema que, en algún momento, tienen que ser reparados. Peter Deutsch formuló estos errores como las siguientes falsas suposiciones que se tienen cuando se desarrolla por primera vez un sistema distribuido [9]: La red es confiable. La red es segura. La red es homogénea. La topología no cambia. La latencia es igual a cero. El ancho de banda es infinito. El costo del transporte es igual a cero. Existe un administrador.

Existen distintos tipos de sistemas distribuidos; los sistemas distribuidos de cómputo, sistemas distribuidos de información y sistemas distribuidos embebidos. Para este trabajo de tesis nos centraremos en los sistemas distribuidos de cómputo. Los sistemas distribuidos de cómputo son utilizados para tareas de cómputo de alto rendimiento. Dentro de este grupo existen dos subgrupos: los sistemas de cómputo en clúster y los sistemas de cómputo en malla o *grid* [9].

Los sistemas de cómputo en clúster han adquirido popularidad, debido a que mejoró la relación precio-rendimiento de las computadoras personales y las estaciones de trabajo. En cierto grado, se volvió financieramente y técnicamente atractiva la construcción de supercomputadoras, usando tecnología sencilla, simplemente conectando computadoras tradicionales de bajo costo mediante una red de alta velocidad. En la mayoría de los casos, la computación en clúster se utiliza para la programación en paralelo en donde un solo programa, utilizado para el cálculo intensivo, se ejecuta paralelamente en múltiples máquinas [9].

Los sistemas de cómputo en *grid* tienen como característica la heterogeneidad de los sistemas que los componen. No se pueden hacer suposiciones de ninguna índole con respecto al *hardware*, sistemas operativos, redes, dominios administrativos, políticas de seguridad, entre otros. El objetivo de un sistema de cómputo en *grid* es reunir los recursos de diferentes organizaciones para permitir la colaboración de un grupo de personas o instituciones. Tal colaboración se realiza en la forma de una organización virtual, de manera que la gente que pertenece a dicha organización virtual tiene derechos de acceso a los recursos que proporciona la organización [9].

Los sistemas distribuidos son realmente complejos, ya que sus componentes se encuentran dispersos en diversas máquinas. Para dominar esta complejidad, resulta crucial que los sistemas se encuentren organizados adecuadamente. Existen diferentes formas de visualizar la organización de un sistema distribuido, una de ellas es diferenciar la organización lógica de la colección de componentes del software de la organización física real [9].

La organización de los sistemas distribuidos trata básicamente sobre los componentes de software que constituyen el sistema, también conocidas como **arquitecturas de software**, las cuales indican cómo se organizarán los componentes de software, y cómo deben interactuar entre ellos [9]. La arquitectura de un sistema es su estructura en términos de componentes especificados por separado, cuyo objetivo global es asegurar que la estructura podrá satisfacer las demandas presentes y previsibles sobre él. Un modelo arquitectónico de un sistema distribuido simplifica y abstrae, en un principio, las funciones de los componentes individuales de dicho sistema y posteriormente considera [28]:

- la ubicación de los componentes en la red, buscando definir patrones utilizables

para la distribución de datos y carga de trabajo;

- las interrelaciones entre los componentes, sus papeles funcionales y los patrones de comunicación entre ellos.

Existen varios patrones utilizados ampliamente para la distribución del trabajo en un sistema distribuido, que tienen un impacto importante en las prestaciones y efectividad del sistema; para más información vea la sección 2.1.3.

2.1.2. Capas de software

El término **arquitecturas de software** se refería inicialmente a la estructuración del software como capas o módulos en una única computadora y más recientemente en términos de los servicios ofrecidos y solicitados entre procesos localizados en la misma o diferentes computadoras. Esta vista orientada a procesos y a servicios puede expresarse en términos de las siguientes capas de servicios [28]:

- **Plataforma:** el nivel de *hardware* y las capas más bajas de software se denominan plataforma para sistemas distribuidos y aplicaciones. Estas capas más bajas proporcionan servicios a las que están por encima de ellas y que son implementadas independientemente en cada computadora, proporcionando una interfaz de programación del sistema a un nivel que facilita la comunicación y coordinación entre procesos.

- ***Middleware*:** es una capa de *software* cuyo propósito es enmascarar la heterogeneidad y proporcionar un modelo de programación conveniente para los programadores de aplicaciones. Se representa mediante procesos u objetos en un conjunto de computadoras que interactúan entre sí para implementar mecanismos de comunicación y de recursos compartidos para aplicaciones distribuidas. El *middleware* se ocupa de proporcionar bloques útiles para la construcción de componentes de software que puedan trabajar con otros en un sistema distribuido. Mejora el nivel de las actividades de comunicación de los programas de aplicación soportando abstracciones como: procedimiento de invocación remota, comunicación entre grupos de procesos, notificación de eventos, replicación de datos compartidos y transmisión de datos multimedia en tiempo

real. Muchas aplicaciones distribuidas dependen enteramente de los servicios proporcionados por el *middleware* disponible para soportar sus necesidades de comunicación y compartir datos.

2.1.3. Arquitecturas de sistemas

La división de responsabilidades entre los componentes del sistema y la ubicación de los componentes en la computadoras en la red es quizá el aspecto más evidente del diseño de un sistema distribuido. Sus implicaciones fundamentales están en las prestaciones, fiabilidad y seguridad del sistema resultante [28].

En un sistema distribuido, los procesos con responsabilidades bien definidas interactúan con los otros para realizar una actividad útil. A continuación se describen los principales tipos de modelos arquitectónicos [28]:

- **Modelo cliente-servidor:** es la arquitectura más citada cuando se discute sobre sistemas distribuidos, ya que es la más ampliamente utilizada. Posee una estructura sencilla, en donde los procesos clientes interactúan con los procesos servidores, en distintas computadoras, con el fin de acceder a los recursos compartidos que estos últimos gestionan. Los servidores pueden, a su vez, ser clientes de otros servidores.

- **Servicios proporcionados por múltiples servidores:** los servidores pueden implementarse como distintos procesos de servidor en computadoras separadas, interactuando entre sí cuando sea necesario, para proporcionar un servicio a los procesos clientes. Los servidores pueden dividir el conjunto de objetos en los que está basado el servicio y distribuírlos entre ellos mismos o pueden mantener copias replicadas de ellos en varias máquinas. La replicación se utiliza para aumentar las prestaciones, disponibilidad y para mejorar la tolerancia a fallas, pues proporciona múltiples copias consistentes de datos en procesos que se ejecutan en diferentes computadoras.

- **Servidores *proxy* y cachés:** un caché es un almacén de objetos de datos utilizados recientemente y que se encuentra más próximo que los objetos en sí. Al recibir un objeto nuevo en una computadora se añade al almacén de la caché,

reemplazando si es necesario algunos objetos existentes. Cuando se necesita un objeto en un proceso cliente, el servicio caché comprueba inicialmente la caché y le proporciona el objeto de una copia actualizada. Sino, se buscará una copia actualizada. Las cachés pueden estar ubicadas en cada cliente o en un servidor *proxy* que puede ser compartido por varios clientes.

- **Procesos de par a par:** en esta arquitectura todos los procesos desempeñan tareas semejantes, interactuando cooperativamente como iguales para realizar una actividad distribuida o cómputo, sin distinción entre clientes y servidores. En este modelo, el código en los procesos iguales mantiene la consistencia de los recursos y sincroniza las acciones a nivel de la aplicación cuando es necesario. Eliminar el proceso servidor reduce los retardos de comunicación entre los procesos al acceder a los objetos locales.

2.1.4. Concurrencia

La concurrencia significa que dos o más programas secuenciales se pueden ejecutar al mismo tiempo como procesos paralelos; un sistema operativo multitarea tiene múltiples procesos concurrentes, en donde sólo uno puede acceder al procesador, alternando el acceso entre los procesos concurrentes [29, 30]. En la figura 2.1 se observan 3 hilos de ejecución, también denominados *threads* en inglés, los cuales se dividen el uso del procesador alternando entre ellos, pues sólo uno a la vez puede utilizarlo.

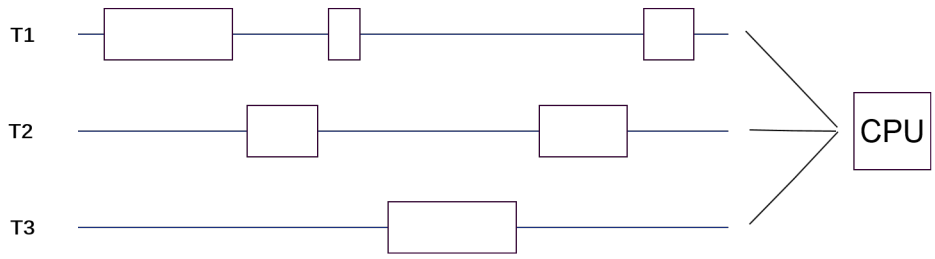

Figura 2.1: Tres hilos de ejecución trabajando concurrentemente en un procesador.

El paralelismo significa que dos o más hilos de ejecución se ejecutan al mismo

tiempo en distintos procesadores. En máquinas con múltiples procesadores, muchos hilos de ejecución diferentes pueden ejecutarse en paralelo, o incluso de forma concurrente en cada procesador [30]. En la figura 2.2 se observan 3 hilos de ejecución, los cuales se ejecutan en tres procesadores, así cada uno se ejecuta simultáneamente sin que tenga que compartir el tiempo de ejecución del procesador.

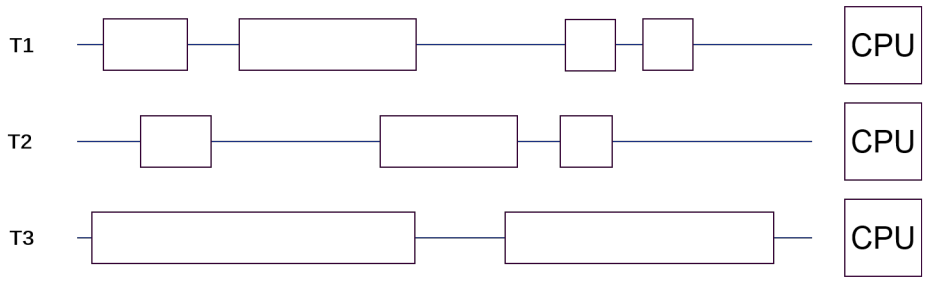

Figura 2.2: Tres hilos de ejecución ejecutando paralelamente en tres procesadores.

En el presente trabajo de tesis se desarrolla un sistema distribuido en donde no se presenta concurrencia a nivel distribuido, pues cada proceso que compone el sistema se ejecuta en maquinas distintas con sus propio procesador; mientras que en cada proceso existen múltiples tareas que se realizan al mismo tiempo, las cuales son ejecutadas de forma concurrente.

Llamadas al sistema

Una **llamada al sistema** es básicamente una función que contiene funciones del *kernel* del sistema operativo. Para los programas multihilo, existe una cuestión muy importante entorno a la cantidad de hilos de ejecución que pueden hacer **llamadas al sistema** de forma concurrente; en algunos sistemas operativos es "uno" y en otros "muchos" [30].

Calendarización

La calendarización es el acto de colocar los hilos de ejecución en los procesadores, de forma en que se puedan ejecutar y quitarlos de los procesadores para que otros hilos se puedan ejecutar en su lugar [30].

Sincronización

La sincronización es el método para asegurar que múltiples hilos de ejecución coordinan sus actividades de manera que un hilo no cambia datos que otro hilo está trabajando, esto se hace proporcionando llamadas a función que puedan limitar el número de hilos que pueden acceder a los datos de manera concurrente [30].

El **Mutex**, del inglés *mutual exclusion lock* o cerradura de exclusión mutua, presenta el caso más sencillo en el que sólo un hilo de ejecución a la vez puede ejecutar un pedazo de código, que puede modificar algunos datos globales, así como leer y escribir en un dispositivo. Por ejemplo, el hilo de T1 obtiene un bloqueo y empieza a trabajar en algunos datos globales. El hilo T2 debe esperar, por lo general cambia su estado a dormido, hasta que el hilo T1 haya terminado, antes de que el hilo T2 pueda ejecutar el mismo código. Mediante el uso de la misma cerradura alrededor del código que cambia los datos, se puede asegurar que los datos se mantienen consistentes [30].

Acciones atómicas e instrucciones atómicas

La implementación de la sincronización requiere de la existencia de la instrucción atómica *test y set* en el conjunto de instrucciones del *hardware*, en máquinas con monoprocesador y multiprocesador. La instrucción *test y set* prueba una variable de la memoria y le establecen un valor (normalmente 1), todo en una sola instrucción, sin la posibilidad de que ocurra algo entre las dos mitades (e.g., una interrupción o una escritura por un microprocesador diferente). Si el valor de la variable objetivo era 0, entonces se cambia a 1 y se considera que se tiene la propiedad de la cerradura. Si ya era 1, no se realiza algún cambio y no se tiene la propiedad de la cerradura. Toda sincronización se basa en la existencia de esta instrucción [30].

Secciones críticas

Una sección crítica es una sección de código a la que se le debe permitir completar su ejecución atómicamente sin interrupción que afecte a su finalización. Las secciones críticas se crean mediante el bloqueo de una cerradura, la manipulación de los datos y después la liberación de la cerradura; cosas tales como incrementar un contador o actualizar un registro deben ser secciones críticas. Otras cosas pueden pasar al

mismo tiempo, y el hilo que se está ejecutando en la sección crítica incluso puede perder su procesador, pero ningún otro flujo puede entrar en la sección crítica; en caso de que otro hilo desee ejecutar en la misma sección crítica, se verá obligado hasta que termine la ejecución del primer hilo. Las secciones críticas normalmente son lo más cortas como sea posible, optimizadas cuidadosamente, ya que pueden afectar significativamente a la concurrencia del programa [30].

Variables de sincronización

La sincronización se proporciona mediante un conjunto de funciones que manipulan estructuras especiales en el espacio de la memoria del usuario. Hay 2 cosas básicas para las que se utilizan las variables de sincronización: La primera es para proteger los datos compartidos; para esto se utilizan las cerraduras. La segunda es que los hilos de ejecución se ejecuten sin tener algo que hacer, desperdiciando tiempo de ejecución del microprocesador; para esto se utilizan los semáforos, las variables de condición, barreras, entre otros mecanismos [30].

Cerraduras de exclusión mutua

La cerradura de exclusión mutua es la variable de sincronización más simple y primitiva, también es conocida como mutex. Proporciona un único, dueño absoluto de la sección crítica que se encuentra entre las llamadas `pthread_mutex_lock()` y `pthread_mutex_unlock()`. El primer hilo que bloquea el mutex obtiene la propiedad, y todos los intentos posteriores para bloquear dicho mutex fallarán, causando que el hilo solicitante pase a modo dormido. Cuando el propietario del mutex lo desbloquea, uno de los hilos que duermen será despertado, cambia a modo ejecutable y se le da la oportunidad de obtener la propiedad del mutex. Es posible que algún otro hilo ejecute la instrucción `pthread_mutex_lock()` y obtener la propiedad del mutex, antes que el hilo recién despertado [30].

En la figura 2.3 se observan 3 hilos de ejecución que necesitan un mutex. Tienen diferentes prioridades ("P:"), que determinan el orden en el que van a la cola de dormidos. Los hilos han solicitado el bloqueo en el orden: T1, T2, T3. T1 es el primero en intentar, por tal razón T1 posee el bloqueo y T3 despertará tan pronto como T1 lo libere, a pesar de que T2 solicitó el bloqueo antes de T3. Tenga en cuenta

que el mutex no sabe quién es el dueño.

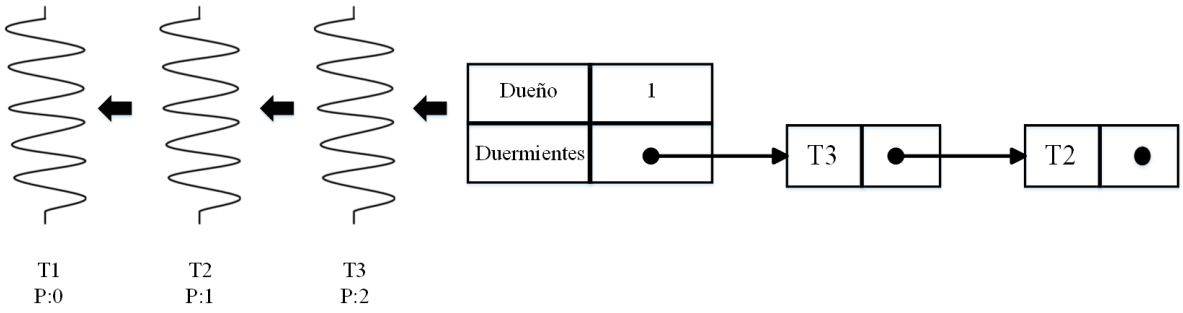

Figura 2.3: Un mutex con tres hilos de ejecución durmiendo.

Semáforos

En la década de 1960, E. W. Dijkstra, profesor del Departamento de Matemáticas de la Universidad Tecnológica, Eindhoven, Países Bajos, extendió el concepto de; semáforo ferroviario al campo de la ciencia computacional. Un semáforo contador es una variable que se puede incrementar arbitrariamente en gran cantidad, pero sólo disminuir a cero. La operación `sem_post()` incrementa el semáforo, mientras que `sem_wait()` lo intenta disminuir. Si el semáforo es mayor que cero, la operación tiene éxito, si no, el hilo que solicita tiene que ir a dormir hasta que un hilo diferente incrementa el semáforo [30].

Un semáforo es útil para trabajar con objetos, en los que importa si hay cero objetos, o más que cero; e. g., los buffers y las listas que se llenan y vacían. Los semáforos también son útiles cuando se desea que un hilo espere por algo [30].

En la figura 2.4, el semáforo inicia con el contador igual a 0. Los hilos ejecutan sus respectivas operaciones en el orden: T1, T2, T3, T4, T5. Después que T1 ejecuta la instrucción `sem_wait()` , tiene que esperar pues el valor del contador es 0. Cuando T2 ejecuta la instrucción `sem_post()`, T1 despierta, y disminuye el valor de nuevo a 0. T3 ejecuta la instrucción `sem_post()`, incrementando el valor a 1. Cuando T4 ejecuta la instrucción `sem_wait()`, puede continuar sin tener que esperar. Finalmente T5 ejecuta la instrucción `sem_wait()` y se queda en espera.

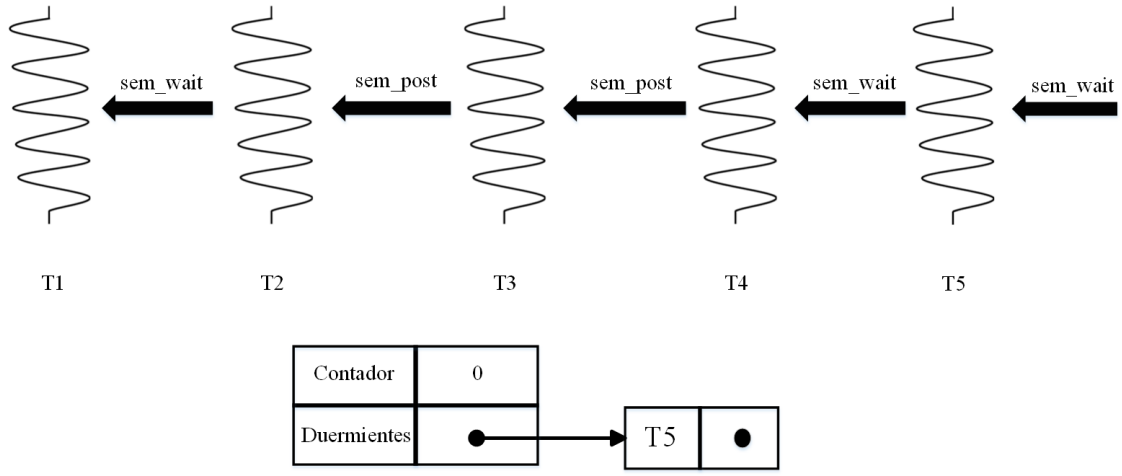

Figura 2.4: Como funciona un semáforo.

Variables condicionales

Una variable condicional crea un entorno seguro para realizar pruebas condicionales, el hilo se detiene al resultar falsa la condición, y se despertará cuando la condición se convierte en verdadera. Una variable condicional trabaja de la siguiente manera: Un hilo obtiene una cerradura de exclusión mutua (también conocida como mutex), las variables condicionales siempre tienen un mutex asociado, y pone a prueba la condición bajo la protección del mutex; ningún otro hilo debe alterar cualquier aspecto de la condición sin tener la propiedad del mutex. Si la condición es verdadera, el hilo completa su tarea, y libera el mutex cuando sea apropiado; si la condición no es verdadera, el mutex se libera, y el hilo se va a dormir en la variable condicional. Cuando algún otro modifica a la variable condicional, ejecuta un llamado a la función `pthread_cond_signal()`, despertando a un hilo dormido; el hilo vuelve a adquirir el mutex, y debe volver a evaluar la condición, si la evaluación es falsa vuelve a dormirse, sino continua con sus operaciones [30].

En la figura 2.5, los hilos T1, T2 y T3, evalúan la condición, determinan que es falsa, entonces se duermen en la variable condicional. El hilo T4 se ejecuta y modifica la condición a verdadera, después despierta a la hilos que están durmiendo. El hilo T3 despierta, vuelve a evaluar la condición, encuentra la condición como verdadera; realiza sus operaciones y libera el mutex cuando ha terminado. Suponiendo que el

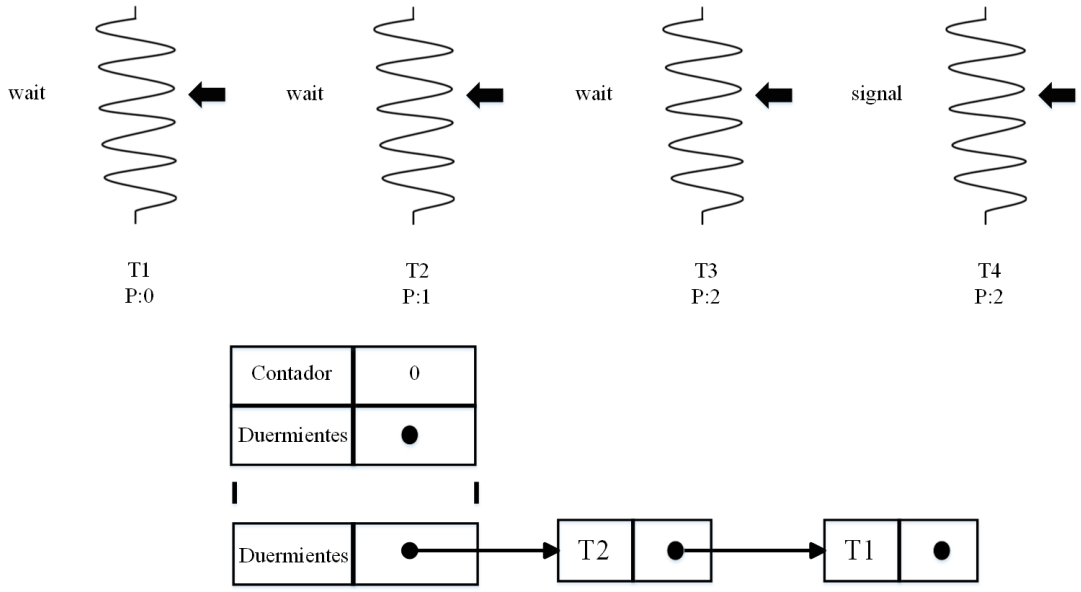

Figura 2.5: Uso de una variable condicional.

hito T3 también cambió la condición de nuevo a falsa, así que no hay razón para despertar a los otros hilos; si el hilo T3 no cambia la condición, entonces el siguiente hilo en la lista de dormido debe despertar.

2.1.5. Resumen

Un sistema distribuido es definido por Tanenbaum y Steen como: "Un sistema distribuido es una colección de computadoras independientes que dan al usuario la impresión de constituir un único sistema coherente" [9]. Las características de un sistema distribuido son [9, 28]: accesibilidad, heterogeneidad, transparencia, grado de apertura, extensibilidad, seguridad, escalabilidad y tratamiento de fallas. Existen distintos tipos de sistemas distribuidos; los sistemas distribuidos de cómputo, sistemas distribuidos de información y sistemas distribuidos embebidos. Para este trabajo de tesis nos centramos en los sistemas distribuidos de cómputo. Los sistemas distribuidos de cómputo son utilizados para tareas de cómputo de alto rendimiento, dentro de este grupo existen dos subgrupos; los sistemas de cómputo en clúster y los sistemas de cómputo en malla o grid [9].

La organización de los sistemas distribuidos trata básicamente sobre los com-

ponentes de software que constituyen el sistema, también conocidas como **arquitecturas de software**, las cuales indican cómo se organizarán los componentes de software y cómo deben interactuar entre ellos [9]. La arquitectura de un sistema es su estructura en términos de componentes especificados por separado, el objetivo global es asegurar que la estructura podrá satisfacer las demandas presentes y previsibles sobre él [28].

La concurrencia significa que dos o más programas secuenciales se pueden ejecutar al mismo tiempo como procesos paralelos; un sistema operativo multitarea tiene múltiples procesos concurrentes, en donde sólo uno puede acceder el procesador, alternando el acceso entre los procesos concurrentes [29, 30]. El paralelismo significa que dos o más hilos de ejecución se ejecutan al mismo tiempo en distintos procesadores. En maquinas con múltiples procesadores, muchos hilos de ejecución diferentes pueden ejecutarse en paralelo, también ejecutándose de forma concurrente en cada procesador [30].

La sincronización se proporciona mediante un conjunto de funciones que manipulan estructuras especiales en el espacio de la memoria del usuario. Hay 2 cosas básicas para las que se utilizan las variables de sincronización: La primera es para proteger los datos compartidos; para esto se utilizan las cerraduras. La segunda es que los hilos de ejecución se ejecuten sin tener algo que hacer, desperdiciando tiempo de ejecución del microprocesador; para esto se utilizan los semáforos, las variables de condición, barreras, entre otros mecanismos [30].

2.2. Sistemas expertos

Como se especificó en el capítulo 1, durante el presente trabajo de tesis se especifica y detalla el diseño una arquitectura distribuida para el manejo de un sistema experto. En esta sección se habla acerca de los sistemas expertos y sistemas expertos distribuidos, algunas definiciones, características y los elementos que los componen; también se menciona su clasificación y las metodologías que existen para su implementación.

2.2.1. Sistemas expertos

Feigenbaum define a un sistema experto como un programa computacional inteligente que utiliza el conocimiento y procedimientos de inferencia para resolver problemas que son lo suficientemente complicados para requerir una experiencia significativa [1]. La idea básica detrás de un sistema experto es que la experiencia, que es un gran cuerpo de conocimiento especifico de una tarea, se transfiera de un ser humano a una computadora; este conocimiento es utilizado por el sistema experto cuando los usuarios lo consultan para obtener consejos específicos sobre el tema de conocimiento que posee [8].

Los sistemas expertos son una rama de la inteligencia artificial aplicada; por lo general, un sistema experto debe resolver problemas muy complicados de forma correcta como un humano e incluso mejor. Utilizan lo que son esencialmente reglas generales, usualmente de una forma muy especifica del dominio; procesan información simbólica, funcionando con errores en los datos y con reglas de razonamiento imperfectas; también pueden mantener simultáneamente hipótesis contradictorias [1].

Otra característica de un sistema experto es que su razonamiento puede ser seguido por un ser humano, normalmente se puede exigir que un sistema experto explique como llegó a una conclusión; también es posible que, en el medio de una interacción cuando el sistema experto solicita alguna información, se le puede preguntar la razón por la cual solicita la información y debe proporcionar una respuesta razonable del motivo [1].

Un sistema experto está compuesto por un motor de inferencia; el cual tiene asociada una base de conocimiento, que contiene reglas de inferencia y la información relacionada con los hechos sobre un dominio en particular. Una base de conocimiento consiste en cientos de reglas, el motor de inferencia aplica las reglas y calcula los valores de los parámetros que dependen de las reglas; una de las mayores tareas del motor de inferencia es programar las tareas, decidiendo el orden en el que se disparan [1].

Existen varios criterios para determinar cuándo es apropiado utilizar los sistemas expertos [26]:

- El dominio del problema está bien delimitado.

- Los tareas a resolver requieren de habilidades cognitivas y no de sentido común.
- El problema está bien definido y entendido.
- El problema debe tener una estructura simbólica y debe haber heurísticas disponibles para su solución.

Si los criterios anteriores son cumplidos y el sistema experto es desarrollado de manera exitosa, puede ofrecer las siguientes ventajas [26]:

- **Coherencia y fiabilidad:** los expertos humanos frecuentemente son inconsistentes en su asesoramiento y toma de decisiones. Los sistemas expertos, las mismas entradas siempre producen los mismos resultados.
- **Flexibilidad:** la mayoría de los sistemas expertos tienen la capacidad de evolucionar con el tiempo, permitiendo cambiar las reglas fácilmente. Los sistemas expertos pueden ser modificados para integrarse con sistemas gestores de bases de datos y programas de hojas de cálculo.
- **Accesibilidad:** los sistemas expertos se pueden utilizar para aumentar la accesibilidad y disponibilidad de conocimientos técnicos, proporcionando un apoyo oportuno y sensible cuando no es rentable ni práctico depender de los expertos humanos.
- **Eficiencia:** los sistemas expertos proporcionan la velocidad y consistencia que los seres humanos pueden no ser capaz de igualar. Los sistemas expertos pueden considerar una mayor cantidad de datos.
- **Productividad:** los sistemas expertos pueden trabajar más rápido que los humanos, la interacción con una computadora procede más rápido que la consulta con los verdaderos expertos. La productividad se mejora mediante la liberación de los expertos humanos de las tareas de rutina que, sin embargo, requieren una gran experiencia. El conocimiento del experto se puede acceder en cualquier momento y distribuirlo ampliamente.
- **Tiempo de Respuesta:** los sistemas expertos en algunos casos responden mucho más rápido que un experto humano, sobre todo cuando hay que pasar por un gran volumen de datos.

- **Procesamiento de Información Incompleta o Incierta:** en contraste con los sistemas informáticos convencionales, los sistemas expertos pueden, al igual que los expertos humanos, trabajar con información incompleta.
- **Mejoramiento de la Resolución de Problemas:** los sistemas expertos mejoran la resolución de problemas al permitir la integración del juicio del usuario en el análisis
- **Beneficios para la educación:** la mayoría de los sistemas expertos pueden explicar el razonamiento detrás de sus conclusiones. A través de estas explicaciones, el experto humano puede obtener conocimientos adicionales y también permite entrenar a nuevos expertos.
- **Rentable:** el sistema experto se escribe una vez, se distribuye a todos los usuarios, y es utilizado en varias ocasiones.
- **Varios expertos:** el conocimiento de varios expertos puede ser integrado en un sistema de conocimiento.

2.2.2. Clases de sistemas expertos

Basado en el modo de funcionamiento de los sistemas expertos se han identificado dos clases de sistemas expertos [19]:

- **Sistemas expertos autónomos:** un sistema experto autónomo usa datos y restricciones pertinentes al problema y lo resuelve utilizando procedimientos bastante simples, no utiliza el enfoque de la investigación de operaciones, el cual implica modelar el problema dado y resolver el modelo usando un óptimo o un "buen" algoritmo heurístico.
- **Sistemas expertos tándem:** un sistema experto tándem combina el enfoque de la investigación de operaciones con el enfoque de sistemas expertos con el fin de resolver un problema, se puede considerar como un sistema experto vinculado a una base de datos de modelos y algoritmos. El enfoque básico utilizado en un sistema experto tándem es de la siguiente manera; un modelo adecuado es seleccionado o construido para el problema dado. Para resolver el modelo,

un algoritmo óptimo o heurística es seleccionado, la solución generada por el algoritmo se modifica, si es necesario, a fin de incorporar aspectos cualitativos y para obtener una solución que se pueda implementar. Existen 3 variantes de sistemas expertos tándem:

- Sistema experto de modificación de datos.
- Sistema experto basado en modelos.
- Sistema experto de modificación de modelos.

2.2.3. Metodologías para la implementación de sistemas expertos

Existen muchas metodologías para la implementación de sistemas expertos, a continuación se describen brevemente [8]:

- **Sistemas basados en el conocimiento:** KBS por sus siglas en inglés, son sistemas que por lo general están centrados en los humanos, por lo que tienen sus raíces en el campo de la inteligencia artificial (AI, por sus siglas en inglés). Son intentos de entender e iniciar el conocimiento humano en los sistemas computacionales. Tienen cuatro componentes principales: una base de conocimientos, un motor de inferencia, una herramienta para la ingeniería del conocimiento y una interfaz de usuario especifica.

- **Redes neuronales:** ANN por sus siglas en inglés, es un modelo que emula una red neural biológica. Son utilizadas para implementar simulaciones de software para el procesamiento masivamente paralelo, lo que implica que los elementos de procesamiento deben interconectarse en la arquitectura de red. La neurona artificial recibe entradas que son análogas a los impulsos electroquimicos que las dendritas de las neuronas biológicas reciben de otras neuronas. La salida de la neurona artificial corresponde a las señales enviadas desde una neurona biológica sobre su axón. Estas señales artificiales se pueden cambiar de manera similar a los cambios físicos que se producen un una sinapsis neural.

- **Sistemas expertos difusos:** Se desarrollan utilizando el método de la lógica difusa, que se ocupa de la incertidumbre. Esta técnica que utiliza la teoría matemática de los conjuntos difusos, simula el proceso de razonamiento humano normal, permitiéndole a una computadora comportarse de una manera menos precisa y lógica que una computadora convencional. Se utiliza este enfoque porque la toma de decisiones no siempre es una cuestión de blanco o negro, verdadero o falso, sino que a menudo implica zonas grises o intermedias.

- **Metodología orientada a objetos:** Combina en un solo objeto datos junto a los procedimientos específicos que operan sobre estos datos, en donde el objeto combina datos y código de programa. En lugar de pasar los datos a los procedimientos, los programas envían un mensaje a un objeto para efectuar un procedimiento que ya está incrustado en él. Entonces, el mismo mensaje puede ser enviado a muchos objetos diferentes, pero cada uno tendrá que aplicar el mensaje de manera diferente. Los datos de un objeto son encapsulados de otras partes del sistema, por lo que cada objeto es un bloque de software que puede ser utilizado en muchos sistemas sin tener que cambiar el código del programa.

- **Razonamiento basado en casos:** CBR por sus siglas en inglés, su idea básica es adaptar soluciones que fueron utilizadas para resolver problemas anteriores y utilizarlas para resolver nuevos problemas. En el razonamiento basado en casos, las descripciones de las experiencias pasadas de los humanos especialistas, se almacenan en una base de datos para más tarde recuperarlas cuando el usuario se encuentra con un nuevo caso con parámetros similares. El sistema busca los casos almacenados con características del caso similares al nuevo, encuentra el ajuste más cercano, y aplica las soluciones del viejo caso al nuevo caso. Las soluciones exitosas son etiquetadas para el nuevo caso y ambos son almacenados junto con los otros casos en la base de conocimiento. La soluciones sin éxito también se anexan a la base de casos junto con las explicaciones de el por qué las soluciones no funcionaron.

- **Modelado:** La metodología de modelado se ha convertido en una metodología de sistema experto con un enfoque interdisciplinario, con el fin de establecer relaciones formales con el diseño del modelo lógico en diferentes dominios de

conocimiento y problemáticas. Además, la tecnología de modelado puede proporcionar métodos cuantitativos para analizar los datos para representar o adquirir conocimiento experto, con la programación lógica inductiva o algoritmos que la inteligencia artificial, la ciencia cognitiva y otros campos de investigación puedan tener plataformas más amplias para implementar tecnologías para el desarrollo de sistemas expertos.

- **Arquitectura del sistema:** La arquitectura de un sistema experto es similar al dibujo de la arquitectura de una casa, ofrece a los usuarios una idea general de cómo se va a lucir el sistema y como se va a aplicar. La arquitectura muestra las capacidades generales del sistema, las interfaces de los usuarios, las funciones del sistema, el flujo de datos del sistema, la gestión del sistema, el sistema gestor de base de datos, el protocolo necesario, el lenguaje de programación específico, entre otras características. Una vez que el diseño de la arquitectura del sistema y la implementación se han completado, los usuarios pueden manipular y controlar las funciones del sistema en la arquitectura del sistema.

- **Agentes inteligentes:** IA por sus siglas en inglés, es un programa informático que ayuda a los usuarios con tareas rutinarias de la computadora. Son sistemas computacionales que habitan en algún complejo entorno dinámico, sensan y actúan de forma autónoma en este entorno, y al hacerlo, realiza un conjunto de metas o tareas para las que son diseñados, empleando conocimiento o la representación de los objetivos o deseos de los usuarios [31].

- **Ontologías:** Es un sistema de vocabulario, que se utiliza como un concepto fundamental para describir una tarea o dominio de conocimiento para ser identificado. Este vocabulario se utiliza como una base de la comunicación entre los expertos del dominio y los ingenieros del conocimiento. Como consecuencia, se tiene un modelo para tareas y dominios, reutilizable que puede ser representado y se genera un código de programa de computadora para dicha ontología, para la adquisición de conocimiento, la reutilización y el aprendizaje heurístico.

- **Metodología de base de datos:** Una base de datos es una colección de datos organizados de manera eficiente para servir a muchas aplicaciones al

centralizar los datos y minimizar los datos redundantes. Un sistema gestor de base de datos, DBMS por sus siglas en inglés, es el software que permite a una organización centralizar los datos, gestionarlos de manera eficiente, y proporcionar el acceso a los datos almacenados por los programas de aplicación. Algunas grandes bases de datos hacen descubrimiento de conocimiento de una forma computacionalmente costosa, porque algunos dominios de conocimiento, ocultos en la base de datos pueden guiar y restringir la búsqueda de importante conocimiento. Las metodologías de bases de datos modernas tiene que procesar grandes volúmenes, múltiples jerarquías y diferentes formatos de datos para descubrir conocimiento a un nivel de experto.

2.2.4. Sistemas expertos distribuidos

La resolución distribuida de problemas es un subcampo de la inteligencia artificial que se ocupa de las interacciones de grupos de agentes inteligentes que intentan cooperar para resolver problemas. Un sistema experto distribuido (DES por sus siglas en inglés), es un sistema de cómputo basado en red [24], consta de múltiples nodos de procesamiento físicamente separados, teniendo en cada uno al menos un sistema experto (ES por sus siglas en inglés), que se ocupa de la resolución de problemas aplicando técnicas de inteligencia artificial y varios solucionadores de problemas. Se diferencia del área más general de procesamiento distribuido, ya que se ocupa de la distribución del control, así como los datos y puede involucrar una amplia cooperación entre los sistemas expertos [32].

El procesamiento distribuido aborda el problema de la coordinación de una red de agentes computacionales para llevar a cabo un conjunto de tareas, en su mayoría independientes. El objetivo de un sistema experto distribuido es crear un equipo de sistemas expertos cooperativos que actúan juntos para solucionar un único problema complejo [32]. La información del sistema experto es distribuida entre los nodos del sistema, incluyendo las reglas utilizadas por el sistema experto, cada nodo ejecuta la parte que le corresponde del razonamiento global y difunde los resultado obtenidos a través de la red para ser utilizados por los demás nodos en su razonamiento [5].

La cooperación en un sistema experto distribuido se refiere en cómo los sistemas expertos componentes del sistema pueden trabajar juntos para resolver un problema

más allá de sus habilidades individuales. Cada sistema experto en el sistema distribuido es capaz de dar soluciones sofisticadas a un problema y puede trabajar de forma independiente, pero los problemas que enfrenta un sistema experto distribuido no pueden ser completados sin la cooperación. La cooperación es necesaria porque ningún sistema experto tiene la suficiente experiencia, los recursos y la información para resolver uno de los problemas que enfrenta un sistema experto distribuido [24, 32].

Los sistemas expertos distribuidos presentan varios retos durante su diseño e implementación. La complejidad de la programación de los nodos representa un serio problema, pero puede ser contrarrestada utilizando técnicas de inteligencia artificial en la programación. Otro problema es la distribución de la información entre los nodos a través de la red de comunicaciones, permitiendo que cada nodo obtenga su parte del sistema experto para ejecutar; es posible concebir varias políticas para la distribución de tareas, como la minimización del tráfico de datos en la red o la redundancia de reglas en más de un nodo para ofrecer tolerancia a fallas. El principal reto al que debe hacer frente un sistema experto distribuido es la coherencia de la información, se debe garantizar que la información contenida en todos los nodos debe ser la misma, por tal razón el algoritmo de distribución es una parte importante del sistema experto distribuido y debe ser lo más eficiente posible [5].

2.2.5. Resumen

Feigenbaum define a un sistema experto como un programa computacional inteligente que utiliza el conocimiento y procedimientos de inferencia para resolver problemas que son lo suficientemente complicados para requerir una experiencia significativa [1]. La idea básica detrás de un sistema experto es que la experiencia, que es un gran cuerpo de conocimiento especifico de una tarea, se transfiera de un ser humano a una computadora; este conocimiento es utilizado por el sistema experto cuando los usuarios lo consultan para obtener consejos específicos sobre el tema de conocimiento que posee [8].

Un sistema experto está compuesto por un motor de inferencia; el cual tiene asociada una base de conocimiento, que contiene reglas de inferencia y la información relacionada con los hechos sobre un dominio en particular. Una base de conocimiento

consiste en cientos de reglas, el motor de inferencia aplica o dispara las reglas, y calcula los valores de los parámetros que dependen de las reglas; una de las mayores tareas del motor de inferencia es programar las tareas, decidiendo el orden en el que se disparan [1].

Existen varios criterios para determinar cuándo es apropiado utilizar los sistemas expertos: El dominio del problema está bien delimitado. Los tareas a resolver requieren de habilidades cognitivas y no de sentido común. El problema está bien definido y entendido. El problema debe tener una estructura simbólica y debe haber heurísticas disponibles para su solución [26].

Si los criterios anteriores son cumplidos y el sistema experto es desarrollado de manera exitosa, puede ofrecer las siguientes ventajas: Coherencia y fiabilidad. Flexibilidad. Accesibilidad. Eficiencia. Productividad. Tiempo de Respuesta. Procesamiento de Información Incompleta o Incierta. Mejoramiento de la Resolución de Problemas. Beneficios para la educación. Rentable [26].

Basado en el modo de funcionamiento de los sistemas expertos se han identificado dos clases de sistemas expertos: Sistemas expertos autónomos, usan datos y restricciones pertinentes al problema y lo resuelven utilizando procedimientos bastante simples. Sistemas expertos tándem, combinan el enfoque de la investigación de operaciones con el enfoque de sistemas expertos con el fin de resolver un problema, se pueden considerar como sistemas expertos vinculados a una base de datos de modelos y algoritmos [19].

Un sistema experto distribuido (DES por sus siglas en inglés), es un sistema de cómputo basado en red [24], consta de múltiples nodos de procesamiento físicamente separados, teniendo en cada uno al menos un sistema experto (ES por sus siglas en inglés), que se ocupa de la resolución de problemas aplicando técnicas de inteligencia artificial y varios solucionadores de problemas. Se ocupa de la distribución del control, los datos y puede involucrar una amplia cooperación entre los sistemas expertos [32].

2.3. Trabajo relacionado

A continuación se muestran una lista de varios trabajos de sistemas expertos distribuidos; se analizaron con la finalidad de observar las características que poseen,

tomando en cuenta sus fortalezas para incluirlas en el diseño de la arquitectura distribuida y aprendiendo de los problemas que tuvieron.

- *Distributed Web-based Expert System for Launch Operations* [22]: sistema experto distribuido que tiene como objetivo ser un sistema de soporte para la toma de decisiones en operaciones de lanzamiento de cohetes; se compone de 3 sistemas expertos que cooperan entre sí y comparten información. El primero es el sistema experto del clima; se encarga analizar ciertos aspectos del medio ambiente en el que se realiza el despegue o aterrizaje, hora, condiciones de la plataforma o pista, duración de la misión, equipo de pista y tipo de aterrizaje. El segundo es el sistema experto de gases Tóxicos, el cual utiliza un modelo de dispersión de gases tóxicos para calcular la concentración máxima y deposición, durante los lanzamientos normales. El tercero es el sistema experto de evaluación de riesgos de salud humana, el cual evalúa la posibilidad que el viento lleve emisiones tóxicas a las áreas en las que se encuentra el personal civil y militar, utilizando un modelo llamado "Latra", el cual se basa en una simulación de Monte Carlo con una cantidad limitada de datos utilizados en las funciones de respuestas tóxicas para seres humanos. La interoperabilidad entre los sistemas expertos está basada en el protocolo de Internet, (IP por sus siglas en inglés), el cual permite distribuir a los sistemas expertos y comunicarlos mediante la Web. Aparte del proceso de inferencia de los sistemas expertos, los datos también se comparten entre los sistemas expertos; se utilizan Servidores Web Tomcat para distribuir la información y así permitir la cooperación horizontal entre los sistemas expertos.

- *Distributed Expert Systems for Network Performance Optimization* [6]: sistema experto distribuido que utiliza una metodología de optimización mediante simulación para el diseño de redes basadas en colas; está constituido por 2 módulos. El primer módulo es el encargado del modelado de las redes utilizando una base de conocimiento. El segundo módulo es el optimizador, el cual mediante simulaciones se encarga de optimizar la red diseñada por el primer módulo. Los dos módulos no cooperan durante el procesamiento.

- *A bridge crane advanced control system implemented by means of a Distribu-*

ted Expert System [5]: sistema encargado del control de una grúa industrial que está compuesto por 4 nodos genéricos implementados utilizando microcontroladores de bajo consumo energético. Todos los nodos tienen módulos para monitorizar a la grúa y también poseen los componentes necesarios para el funcionamiento del sistema experto distribuido, en donde cada nodo ejecuta la parte que le corresponde del razonamiento global y difunde los resultados a los demás nodos; con este enfoque se permite que el sistema sea tolerable a fallas.

- *Power System voltage control by Distributed Expert Systems* [4]: sistema para controlar el voltaje en un sistema de energía eléctrica, implementado utilizando como base un sistema operativo especializado para sistemas de procesamiento intelectual distribuido; su objetivo es asegurar que el voltaje del sistema de energía eléctrica sea estable; cuando existe un decremento del voltaje, sea utilizan dispositivos que se encargan de compensar dicha caída. Los dispositivos de compensación son controlados por los nodos del sistema expertos distribuidos, los cuales se encargan de monitorizar cada parte del sistema de energía eléctrica; cuando detectan una caída del voltaje analizan si es posible recuperar el nivel de voltaje por su propia cuenta, en caso contrario solicitan el apoyo de otros nodos y sus respectivos dispositivos de compensación de voltaje.

- *A Distributed Expert System for Stability Analysis* [23]: sistema encargado de analizar la estabilidad de otro sistema. Se compone de 3 módulos: pizarrón, fuentes de conocimiento y mecanismo de control; es implementado utilizando agentes inteligentes; cuando un agente termina el proceso de inferencia de un evento, envía sus conclusiones a lo demás agentes. El sistema experto distribuido está implementado utilizando una versión modificada de Clips, que permite utilizar llamados a procedimientos remotos (*RPC*, por sus siglas en inglés).

- *Teaching Resource Grid System Model Based on Multi-Agent Distributed Expert System* [24]: sistema para la localización y recuperación de recursos para la enseñanza en una Grid utilizando un modelo de sistema experto distribuido basado en múltiples agentes inteligentes. El sistema está conformado por agentes inteligentes; en ellos se distribuye el conocimiento y los procesos de inferencia. Cada agente puede trabajar de forma autónoma, pero si recibe una petición

para la cual no tiene los recursos necesarios, entonces resuelve las partes para las cuales es competente, después solicita ayuda a otro agente que sea capaz de resolver el problema, realizándose la cantidad de peticiones a otros agentes que sean necesarias, finalmente se recolectan todos los resultados y se entrega una respuesta.

- *Path finder algorithm used by Distributed Expert System for traffic control* [7]: sistema para la búsqueda de rutas mediante un sistema experto distribuido en el control del tráfico ferroviario; los nodos con los sistemas expertos se encuentran repartidos en todos los segmentos de vías de las estaciones. Cada tren tiene asociado un agente inteligente, que posee la información acerca de la ruta a seguir y los tiempos de recorrido; el agente del tren se encarga de preguntar la mejor ruta para la siguiente estación al sistema experto local, ubicado en la estación actual; la ruta no puede interferir con las rutas de otros trenes previamente establecidas.

- *System architecture of a Distributed Expert System for the management of a national data network* [2]: arquitectura encargada de la administración de una red de telecomunicaciones nacional, la cual abarca una amplia superficie. Los nodos incluyen el sistema experto para el razonamiento, también los dispositivos necesarios para monitorizar y administrar los elementos de la red de telecomunicaciones. Cada nodo es responsable de un segmento de la red; cuando existe un evento que repercute en más de un segmento, los nodos encargados deben cooperar para tomar las decisiones pertinentes.

- Muhadith: *A Cloud Based Distributed Expert System for Classification of Ahadith* [25]: sistema para la clasificación de Ahadith, el plural de Hadith, una escritura religiosa. El sistema está diseñado utilizando una arquitectura orientada a Servicios Web; se encarga de decidir si una escritura es auténtica y la clasifica con base en distintos criterios. El sistema se compone de varios nodos que se encargan del proceso de inferencia; cuando se recibe una petición de procesamiento de hechos, es necesario distribuirlos a todos los nodos y posteriormente combinar todas las respuestas en una sola.

Capítulo 3

Arquitectura distribuida

El objetivo del presente trabajo de tesis, definido en él capítulo 1, es determinar si es posible agregar las características: Escalabilidad, tolerancia a fallos y alta disponibilidad, propias de un sistema distribuido, a un sistema experto, mediante el uso de una arquitectura distribuida. En este capítulo se explica el análisis y diseño de dicha arquitectura distribuida; se realiza el análisis de las entidades necesarias en la arquitectura, detallando las funciones y responsabilidades individuales que tienen, también se analiza la forma en la que deben interactuar para lograr una cooperación correcta entre ellas.

3.1. Análisis

El objetivo del presente trabajo de tesis es determinar si es posible crear una arquitectura distribuida para el control de un sistema experto que permite agregarle las siguientes características: escalabilidad, tolerancia a fallas y alta disponibilidad; propias de los sistemas distribuidos.

A continuación se muestra el análisis del problema presentado en la sección 1.2, él cual se pretende resolver mediante el uso de varias estrategias, las cuales se incorporan a la arquitectura que se propone en este trabajo de tesis.

3.1.1. Modelo arquitectónico

Tomando en cuenta el análisis previo de los sistemas expertos distribuidos que existen en la literatura hemos notado que lo esencial en las arquitecturas de este tipo es distribuir las funciones y responsabilidades del sistema. Es recomendable que en las arquitecturas siempre existan al menos dos instancias que sean capaces de encargarse de dichas funciones, pues si una falla la otra puede seguir brindando el servicio, pero es conveniente que haya más instancias pues de esta forma la carga de trabajo se puede repartir entre todas.

La división de responsabilidades se realiza de la siguiente forma. Es necesaria la existencia de alguna entidad encargada de atender las peticiones de los usuarios y redirigirlas a otra entidad. También se requiere una instancia encargada de administrar todo el funcionamiento de la arquitectura y además de recibir desde otra entidad las peticiones de los usuarios. La arquitectura que se propone debe trabajar con una gran cantidad de procesos de sistemas expertos, por tal motivo es necesaria una entidad para administrarlos y además responda a las órdenes de las entidades encargadas de la administración de la arquitectura. Separar las funciones de la arquitectura permite tener un esquema más flexible, pues cada entidad se puede modificar manteniendo las interfaces públicas, dando un enfoque modular a la arquitectura que se propone.

Un aspecto muy importante de la arquitectura es el acceso al almacenamiento persistente, en este caso es un sistema gestor de bases de datos; al ser un servicio centralizado, pues en esta versión de la arquitectura distribuida no se contempla utilizar un sistema gestor de bases de datos distribuidas, puede limitar la cantidad de instancias de las entidades a las que puede atender. Por esta razón es importante determinar quién debe acceder al servicio, pues esto puede afectar la escalabilidad del sistema.

3.1.2. Comunicación

Un aspecto importante en los sistemas expertos distribuidos, es cómo los usuarios pueden acceder a los servicios que brindan, es muy importante que los sistemas puedan atender a los usuarios sin importar la plataforma o sistema operativo que utilicen. Por esta razón es necesario utilizar un protocolo de comunicación que sea

ampliamente utilizado y que no sea dependiente de la plataforma; también es importante que el mecanismo de comunicación permita atender a varios usuarios a la vez.

La arquitectura propuesta está compuesta por varias entidades, las cuales colaboran para brindar los servicios a los usuarios, por lo que requieren comunicarse entre sí. Algunas de ellas se pueden encontrar en la misma máquina y otras en distintas máquinas; es necesario definir mediante un esquema físico en donde se especifique cómo se distribuirán las entidades y de esta forma determinar el modo de comunicación que se requiere utilizar para conectar a cada entidad con las entidades que colabora.

Los archivos de hechos que el sistema recibe se necesitan enviar a todas las instancias encargadas de administrar a los procesos de los sistemas expertos; los cuales utilizan los archivos de hechos para el proceso de inferencia. Por este motivo es necesario distribuir los archivos de hechos a una gran cantidad de destinatarios, lo cual puede ocasionar que el sistema presente problemas de escalabilidad; para evitarlo es necesario utilizar un mecanismo de comunicación basado en él protocolo de transmisión *Multicast* [33]. El protocolo *Multicast* permite enviar mensajes a varios destinatarios utilizando un esquema emisor/suscriptor.

3.1.3. Formatos de datos

La información que se transfiere entre las entidades de la arquitectura que se propone se necesita estructurar, con el objetivo de permitir que las entidades que envían y reciben la información puedan comunicarse entre sí de una manera correcta. Como se especificó en la sección 3.1.1, primero es necesario hacer el diseño del diagrama físico de las entidades de la arquitectura distribuida. Con este diseño se determina qué entidades se localizan en la misma máquina. Este paso es importante, debido a que de esto depende el mecanismo de comunicación a utilizar para conectar a cada una de las entidades.

Existe una amplia variedad de mecanismos de comunicación con distintos enfoques, ventajas y desventajas. Algunos mecanismos solo permiten enviar cadenas de texto, en caso de necesitar enviar datos binarios, se requiere codificar dicha información, de tal manera que se pueda enviar como texto y después en el destinatario se

decodifica para obtener los datos binarios originales. Otros mecanismos de comunicación permiten el envío de datos binarios, por lo que ya no es necesario codificar los datos.

3.1.4. Interacción

La distribución de funciones y responsabilidades entre múltiples entidades ocasiona la necesidad de cooperación e interacción entre ellas. Los objetivos generales de las entidades que componen la arquitectura se describieron en la sección 3.1.1. La manera en la que interactúan las entidades de la arquitectura es dependiente del mecanismo que utilizan para comunicarse con las entidades que colaboran.

Una de las características esenciales de un *Middleware* es su relativa facilidad para funcionar con una amplia variedad de *Software*. Con la finalidad de que la arquitectura sea compatible con una mayor cantidad de sistemas expertos, se requiere que no exista una alta dependencia al sistema experto que utiliza; pero es necesario definir un conjunto de requisitos a cumplir por el sistema experto que se quiera utilizar, garantizando que sea compatible y permita el funcionamiento de la arquitectura de la forma en que se diseñó.

3.1.5. Escalabilidad

La arquitectura debe permitir agregar nuevas instancias en cualquier momento, sin que el usuario tenga que hacer modificaciones en las instancias que ya se encuentran en funcionamiento, esto permite que el sistema puede ser escalable; permitiendo agregar nuevas instancias para reemplazar las instancias de entidades que hayan fallado y no se pudieron recuperar. Otra ventaja de poder agregar más instancias es, al aumentar la cantidad de instancias que pueden atender al usuario en el sistema, lo cual, aumenta la disponibilidad de los servicios que provee el sistema de la arquitectura.

Distribuir los archivos de hechos como se describió en la sección 3.1.2, puede repercutir negativamente en la capacidad de escalabilidad de la arquitectura, pues a mayor cantidad de instancias de las entidades encargadas de administrar los procesos de los sistemas expertos, se requiere un mayor tiempo de transmisión, lo que puede producir la saturación de la red de comunicación. Para solventar este problema se

requiere utilizar un protocolo de comunicación *Multicast*, con el que se puede enviar información a múltiples destinatarios al mismo tiempo.

3.1.6. Detección y recuperación de fallas

Una de las características que se agrega a los sistemas expertos a través del sistema distribuido, es la tolerancia a fallas, pero el sistema no puede evitar que un sistema experto falle; para evitar que el usuario note que se presentó la falla de alguna instancia o la computadora en donde se ejecuta, el sistema debe encargarse de recuperar la información del usuario y asignarla en otra instancia que se encuentre en funcionamiento. Para lograr este objetivo es necesario utilizar un sistema de almacenamiento persistente de información que pueda ser accedido por las instancias de control del sistema; un sistema gestor de base de datos cumple con este propósito y puede atender varias peticiones a la vez. La arquitectura necesita tener la capacidad de afrontar la caída de varias instancias de las entidades que la componen, asegurando poder recuperar toda la información que poseen, por esta razón es importante que la información del usuario y del funcionamiento siempre sea respaldada.

3.2. Diseño

El objetivo del trabajo de tesis es determinar si es posible agregar las siguientes características: Escalabilidad, tolerancia a fallas y alta disponibilidad; como se observó durante el análisis de los trabajos relacionados, es posible agregar dichas propiedades a un sistema experto mediante una arquitectura distribuida. Distribuir los servicios entre varias entidades es una estrategia que utiliza en varios de los sistemas expertos distribuidos que se analizaron; es necesario que existan varias instancias de dichas entidades, pues si alguna falla siempre exista otra que la reemplace.

La arquitectura que se propone recibe peticiones *REST* [14], para que los usuarios puedan utilizar sus servicios sin importar la plataforma que utilizan e incorpora el *Middleware ZeroMQ* [34] para comunicarse con los procesos que contienen al sistema experto, el cual se encarga del proceso de inferencia; lo anterior se puede observar en la figura 3.1.

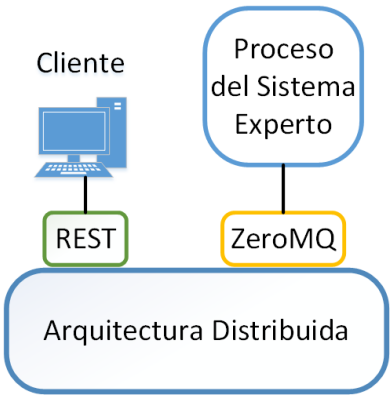

Figura 3.1: Interacción con la arquitectura distribuida.

A continuación se detalla el diseño de la arquitectura que se propone, el cual se basa en él análisis del problema que realiza en la sección 3.1; el diseño se aborda mediante la división en varios aspectos importantes de la arquitectura.

3.2.1. Modelo arquitectónico

El diseño de la arquitectura que se propone está basado en un modelo descentralizado de varios niveles, las entidades que componen la arquitectura se localizan en distintos niveles de la arquitectura. A continuación se explican las entidades que componen la arquitectura.

La arquitectura distribuida propuesta se compone de dos entidades, las cuales se pueden observar en la figura 3.2. La arquitectura propuesta tiene dos entidades que son la entidad de acceso y la entidad del sistema experto. La entidad de acceso, tiene como objetivo permitir a los usuarios acceder al sistema y también es la encargada de la administración del sistema. La entidad del sistema experto, tiene como objetivo administrar y controlar a los procesos del sistema experto; cada proceso tiene embebido el sistema experto para el proceso de inferencia, es posible que todos los procesos compartan la memoria de la base de conocimiento, pero eso depende del sistema experto que se elija.

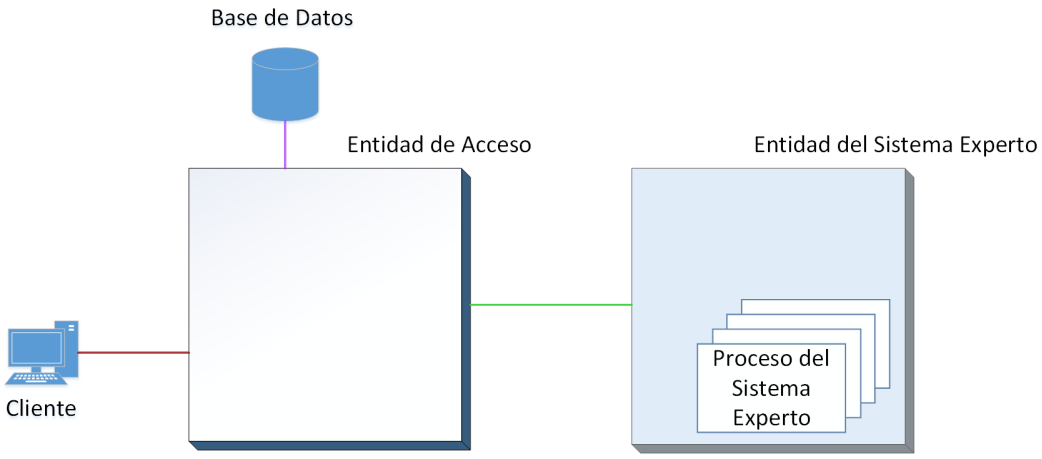

Figura 3.2: Entidades de la arquitectura distribuida.

Desde un punto de organización físico, se puede considerar como un nodo a cada instancia de las entidades; una computadora puede contener uno o más nodos, dependiendo de sus especificaciones de *Hardware*, pues cada nodo requiere gran cantidad de recursos computacionales. La comunicación entre los nodos se realiza mediante el uso del *Middleware* TAO [15] el cual es una implementación de *CORBA* [10–12], que sirve como *Middleware* de comunicación en la implementación de sistemas distribuidos.

A continuación se muestran algunas características de las 2 entidades que componen la arquitectura distribuida que se propone, también se explican los componentes que las integran.

Entidad de acceso

La entidad de acceso es la entrada del usuario al sistema mediante peticiones *REST*, también se encarga de administrar el funcionamiento de la arquitectura distribuida. La entidad de acceso se compone por 2 procesos que trabajan en conjunto; el primer proceso es el encargado de recibir las peticiones *REST* y el segundo proceso se encarga de la administración del sistema de la arquitectura distribuida. Las instancias de esta entidad también se pueden nombrar como **nodos de acceso**.

El proceso encargado de recibir las peticiones *REST* permite que varios usuarios soliciten a una misma instancia le ejecución de algún servicio del sistema de la ar-

quitectura distribuida. El proceso también se encarga de iniciar y comunicarse con el proceso de control del acceso; verifica si no existe un proceso de control del acceso previo iniciado con el mismo identificador único que tiene asignado para su actual ejecución; de existir, lo termina e inicia uno nuevo. La comunicación con el proceso de control se realiza utilizando el *Middleware ZeroMQ*; se utiliza un *Socket* bidireccional síncrono; con el cual cualquiera de los dos procesos puede enviar un mensaje, pero requieren esperar hasta que haya concluido la transmisión del mensaje.

El proceso de control del acceso contiene un objeto *CORBA* con un identificador y una prioridad únicos, el proceso se encarga de controlar y administrar todos los componentes de la arquitectura distribuida. El objeto *CORBA*, dentro del proceso de control, detecta de manera automática a los otros objetos *CORBA* en funcionamiento, mediante el servicio de nombres. También se encarga de conectarse con el sistema gestor de bases de datos para almacenar y recuperar la información del funcionamiento de la arquitectura, con la finalidad de agregar la tolerancia a fallas; el diseño del proceso contempla la integración de una biblioteca que le permita conectarse a distintas implementaciones de sistemas gestores de bases de datos de una manera transparente. Las entidades de la arquitectura se comunican utilizando los objetos *CORBA* que tienen en su interior, encapsulando todo el protocolo de comunicación y evitando que el usuario tenga que involucrarse con una configuración compleja.

Cada proceso de control del acceso se encarga de verificar que las otras instancias en la arquitectura se encuentren en funcionamiento; en caso de detectar la falla de alguna, el proceso de control del acceso de mayor prioridad se encarga de recuperar la información y asignarla a otra instancia capaz de reemplazarla. Con esta estrategia se agrega tolerancia a fallas a la arquitectura distribuida.

Cada proceso de control del acceso tiene asociado un proceso *REST*, el cual recibe por parte de los usuarios las acciones a realizar, las cuales son por el primero. El proceso de control también recibe solicitudes de otros procesos de control que se encuentran en otras instancias, por esta razón, es necesario que el proceso pueda procesar varias tareas al mismo tiempo. Para lograr lo anterior, el diseño propuesto utiliza el patrón *Threads Pool* [35], en él cual se inicializa una cantidad determinada de hilos de ejecución y después son asignados conforme se solicitan; al terminar su

uso se liberan para ser utilizados de nuevo.

Un servicio importante dentro de la arquitectura distribuida es la plataforma para la distribución de archivos, se basa en un protocolo *Multicast* utilizando el patrón de diseño emisor/suscriptor. La plataforma está diseñada para utilizar el *Middleware* de comunicación RabbitMQ[36], utilizando un clúster de servidores para mejorar las prestaciones del servicio. El proceso de control del acceso posee un módulo encargado de la administración de los canales de comunicación, así como gestionar el envío de la información a las entidades del sistema experto. Para mayor información consultar el capítulo 6.

Entidad del sistema experto

La entidad del sistema experto se encarga de administrar y controlar los procesos del sistema experto que se utiliza en el proceso de inferencia. La entidad se compone por dos tipos de procesos; el proceso que administra el funcionamiento de la entidad y los procesos del sistema experto. La comunicación entre estos procesos se realiza utilizando el *Middleware ZeroMQ*. Las instancias de esta entidad también se pueden nombrar como **nodos expertos**.

Cada instancia de la entidad del sistema experto posee solo un proceso de control; el proceso posee un objeto *CORBA*, el cual recibe las peticiones de otras instancias en la arquitectura. Al existir solo un proceso, es el único que puede atender todas las peticiones remotas, para lograrlo se utiliza el patrón *Threads Pool*, que permite realizar varias tareas a la vez.

El proceso de control del sistema experto es el encargado de tener el registro de los archivos de reglas que se han asignado al nodo experto; también tiene el control de los procesos del sistema experto que él inicia y se encarga de iniciar un nuevo proceso por cada archivo de reglas que recibe; esto tiene como finalidad aumentar la compatibilidad con diversos sistemas expertos, pues hay algunos que no pueden remover reglas de maneras específicas de su memoria de trabajo.

El proceso de control del sistema experto posee un módulo encargado de conectarse con la plataforma de distribución de archivos, cuyo diseño se basa en un protocolo *Multicast* utilizando el patrón de diseño emisor/suscriptor. La plataforma está diseñada para utilizar el *Middleware* de comunicación RabbitMQ, utilizando un

clúster de servidores para mejorar las prestaciones del servicio. Una discusión más detallada se encuentra en el capítulo 6.

Cada proceso experto tiene el sistema experto embebido, encargado del proceso de inferencia. Cuando el proceso de control del sistema experto inicia, recibe el nombre del ejecutable y todos los parámetros necesarios para la inicialización de cada proceso experto, de esta forma el usuario tiene la libertad de elegir el sistema experto con el que funcionará la arquitectura distribuida propuesta.

ZeroMQ es un *Middleware* de comunicación, el cual puede ser utilizado en más de 40 lenguajes de programación. Todos los programas de los sistemas expertos que se deseen utilizar con la arquitectura que se propone, necesitan utilizar un *Socket ZeroMQ*, mediante el cual se reciben los mensajes del proceso de control del sistema experto y se regresan las respuestas. Utilizar el *Middleware ZeroMQ* para comunicarse con los procesos del sistema experto, brinda a la arquitectura la capacidad de utilizar una amplia variedad de sistemas expertos, debido a la existencia del *Middleware* para una gran cantidad de lenguajes de programación.

Todo programa de un sistema experto que se quiera utilizar en conjunto con la arquitectura distribuida que se propone, debe entender y responder a los siguientes seis comandos: comprobación de funcionamiento del proceso, eliminación de todas las reglas, limpieza de la memoria de conocimiento, carga de archivo de reglas y procesamiento de archivos de hechos; en caso contrario, no se puede utilizar con la arquitectura. La especificación de los comandos y sus respuestas son abordadas en el capítulo 7.

Cualquier sistema experto que se desee utilizar con la arquitectura distribuida que se propone, debe permitir la ejecución de archivos *scripts*; esto tiene como finalidad incrementar la compatibilidad de la arquitectura con un mayor número de sistemas expertos. Por esta razón no existe una salida directa desde el sistema distribuido; mediante la ejecución de *scripts*, los sistemas expertos a utilizar, le permiten al usuario conectarse con entidades ajenas a la arquitectura distribuida e informar los resultados del proceso de inferencia.

El usuario puede utilizar distintos sistemas expertos con la Arquitectura propuesta, pero hay varios requisitos que debe cumplir el programa con el sistema experto que se desea utilizar. Existen dos casos de uso. El primer caso es cuando se utiliza

un sistema experto distribuido, debe cumplir los siguientes requisitos: debe distribuir la memoria de la base de conocimiento pero sin distribuir la memoria en donde se almacenan las reglas, pueda ejecutar varios procesos del sistema experto distribuido en una misma computadora y se base en archivos de reglas [1]. El segundo caso es cuando se quiere utilizar un sistema experto sin un proceso de inferencia distribuido, debe cumplir los siguientes requisitos: pueda ejecutar varios procesos del sistema experto distribuido en una misma computadora y se base en archivos de reglas; es importante notar que todos los procesos expertos recibirán los hechos para su procesamiento pero los resultados no se comparten entre ellos.

En la figura 3.3 se puede observar el diagrama lógico de los procesos que componen la arquitectura distribuida que se propone, los cuales componen los niveles por los que las peticiones de los usuarios pasan para ser procesadas; también se muestra como se organizan las comunicaciones entre todos los procesos.

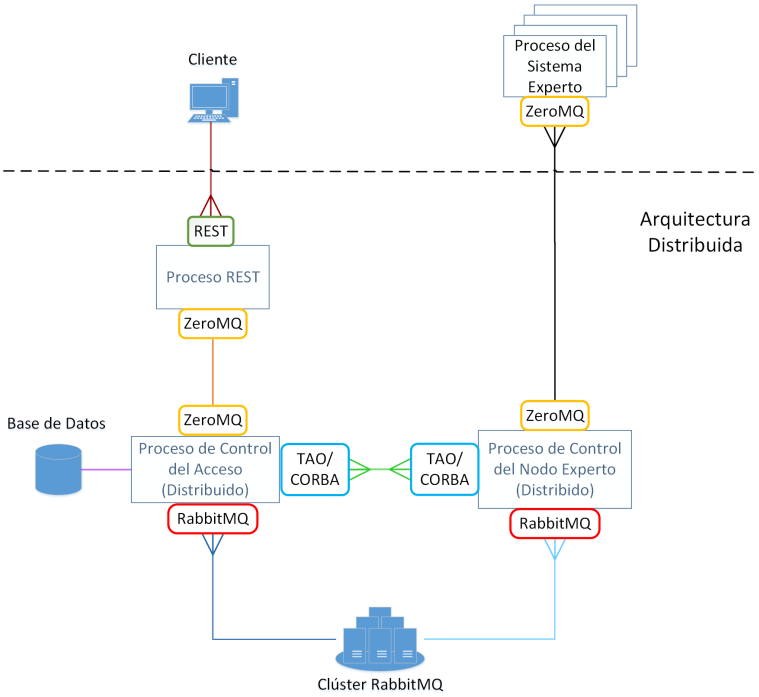

Figura 3.3: Diagrama lógico de los procesos en la arquitectura distribuida.

En la figura 3.4 se puede observar el diagrama lógico de las entidades que componen a la arquitectura distribuida, las cuales agrupan a los procesos. El diseño de

la arquitectura contempla la posibilidad de ejecutar más de una instancia de las entidades en una máquina, pero eso depende de las especificaciones de *Hardware* del equipo; las instancias consumen muchos recursos computacionales, ya que están diseñadas para atender una gran cantidad de peticiones.

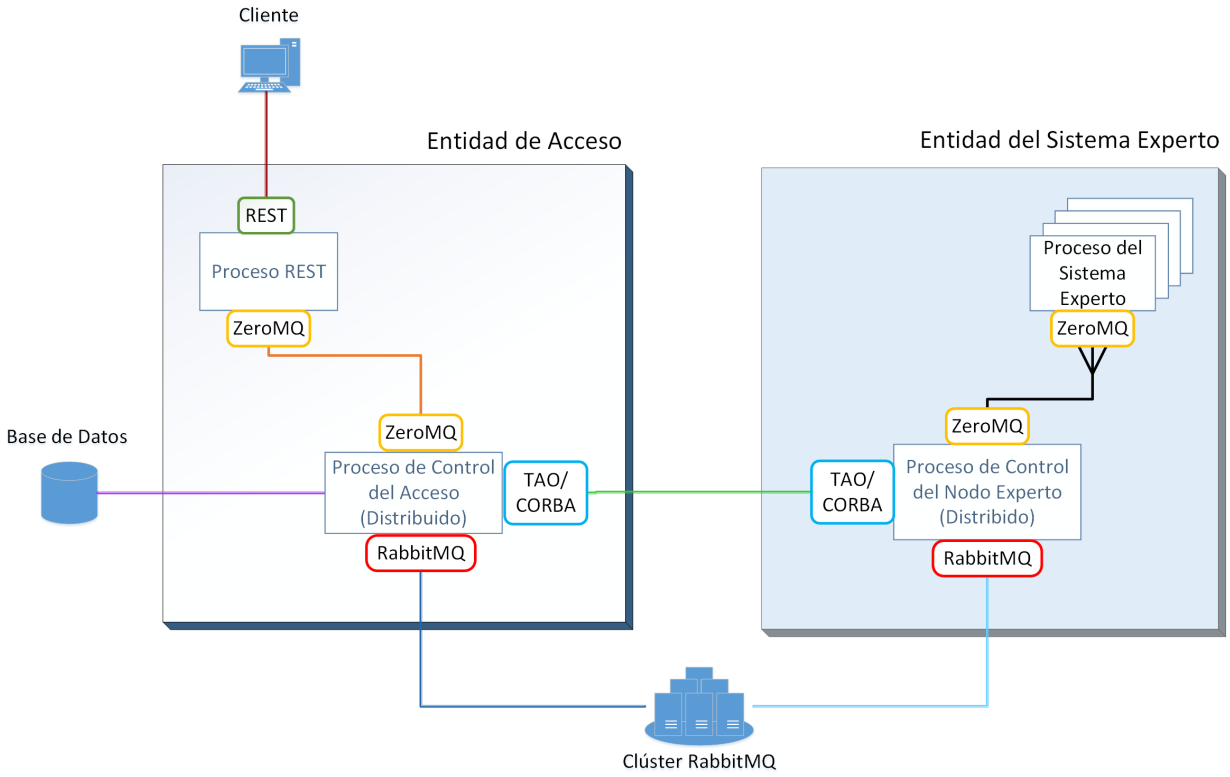

Figura 3.4: Diagrama lógico del agrupamiento de los procesos en las entidades de la arquitectura distribuida.

3.2.2. Comunicación

En la sección 3.2.1 se muestra el diseño que tiene la arquitectura distribuida que se propone, en donde se utilizan distintos mecanismos de comunicación para poder transmitir toda la información necesaria entre todas las instancias de las entidades, las cuales tienen que colaborar para poder brindar todos los servicios del sistema. A continuación se detallan los mecanismos de comunicación que se utilizan entre cada una de las entidades y los procesos que la conforman.

Usuario - proceso *REST*

El diseño de la arquitectura distribuida que se propone está planeado para aceptar peticiones de usuario sin importar el sistema operativo o plataforma utilizada, por esta razón se utiliza una tecnología ampliamente utilizada y además independiente de la plataforma. Las comunicaciones *HTTP* mediante peticiones *REST* son ampliamente utilizadas y se pueden usar desde una amplia variedad de plataformas; por esta razón es el mecanismo de comunicación elegido para recibir las peticiones de los usuarios. El proceso *REST* incluye un servidor que acepta peticiones *REST* con mensajes *HTTP* y las redirige al proceso de control del acceso. En el capítulo 5 se aborda con mayor detalle las características del módulo que recibe las peticiones de los usuarios.

Proceso *REST* - proceso de control del acceso

Un proceso *REST* sólo se comunica con un único proceso de control del acceso y se encuentra en la misma máquina; el proceso *REST* redirige las peticiones de los usuarios al proceso de control y regresa la respuesta adecuada al usuario. Para permitir esta comunicación se utiliza un *Socket* bidireccional del *Middleware ZeroMQ*, el cual sólo puede transmitir un mensaje a la vez en cualquiera de ambos sentidos; por esta razón se utiliza un patrón productor/consumidor para administrar la utilización del canal. La arquitectura utiliza el mismo mecanismo de comunicación para conectarse con los procesos del sistema experto, el módulo de comunicación se explica con mayor detalle en el capítulo 7.

Proceso de control del acceso - proceso de control del acceso

Cada proceso de control de Acceso posee un objeto *CORBA* en su interior, el cual es inicializado con un identificador y una prioridad únicos. La comunicación entre los objetos se realiza mediante la solicitud de una invocación remota de un método en otro objeto **CORBA** dentro de otro proceso de control de acceso; todo el mecanismo de comunicación lo gestiona el *Middleware TAO* de la especificación *CORBA*. El servicio de nombres de *CORBA* le permite a los procesos de control de acceso, detectar automáticamente las otras instancias presentes en él sistema de la arquitectura, pues

todas poseen un objeto *CORBA*.

Proceso de control del acceso - proceso de control del sistema experto (*CORBA*)

Como se mencionó anteriormente, cada proceso de control en la arquitectura posee un objeto *CORBA*, mediante el cual los procesos de las otras instancias en él sistema se comunican con él. Todos los procesos de control de acceso se pueden comunicar con cualquier proceso de control de sistema experto, pero no es necesario en sentido contrario, pues quien solicita el inicio de cualquier acción son los procesos de control de acceso.

Proceso de control del acceso - proceso de control del sistema experto (*RabbitMQ*)

La distribución de los archivos de hechos hacia todos los procesos del sistema experto, es una funcionalidad muy importante, pues se necesita un mecanismo de comunicación para permitir enviar los datos a todas las entidades del sistema experto; lo que podría provocar problemas de escalabilidad. El diseño de la arquitectura distribuida utiliza el *Middleware RabbitMQ*, el cual permite utilizar un protocolo *Multicast* mediante el patrón emisor/ suscriptor, mediante el cual, un mensaje se puede enviar a múltiples destinatarios con un sólo envío. Para más información consultar capítulo 6.

Proceso de control del sistema experto - proceso del sistema experto

Cada proceso de control del sistema experto controla y administra varios procesos del sistema experto, pero cada proceso del sistema experto, sólo responde a las peticiones de un proceso de control del sistema experto; además, todos se ejecutan en la misma máquina. Se utiliza el mecanismo de comunicación del *Middleware ZeroMQ* para conectar a estos procesos utiliza, como en él caso del proceso *REST* y el proceso de control del acceso; pero en este caso es un poco más complejo, una entidad del sistema experto puede contener una gran cantidad de procesos del sistema experto. Para más información consultar el capítulo 7.

3.2.3. Formatos de datos

La arquitectura distribuida que se propone está compuesta para varios procesos, los cuales tienen que comunicarse para cooperar y trabajar en conjunto. Como se mencionó en la sección 3.2.2, los procesos se comunican a través de distintos mecanismos de comunicación, cada uno con cualidades y especificaciones distintas; por esta razón, los mensajes a enviar son estructurados de maneras distintas. A continuación se especifica cómo es la estructura general de los mensajes que manejan los procesos que componen la arquitectura distribuida.

Usuario - proceso *REST*

El proceso *REST* recibe las peticiones de los usuarios codificadas en HTTP, puede responder de 2 formas distintas; el proceso posee comandos que responden con una página Web en HTML, también posee comandos que sólo regresan texto plano como respuesta, en ambos casos la respuesta se codifica en HTML. Para más información consulte el capítulo 5.

Proceso *REST* - proceso de control del acceso

El proceso *REST* se comunica con el proceso de control del acceso utilizando un *Socket* del *Middleware ZeroMQ*. El proceso *REST* recibe los archivos que el proceso de control del acceso necesita distribuir por el sistema de la arquitectura, pero ambos se ejecutan en la misma máquina, por esta razón sólo es necesario que el proceso *REST* envíe mensajes indicando las acciones a tomar; estos mensajes pueden ser enviados utilizando cadenas de texto. Los mensajes de acciones son conjuntos de cadenas de texto que se concatenan, separadas por cadenas especiales de separación llamadas *tokens*, las cuales permiten que el proceso de control del acceso separe las cadenas y obtenga toda la información original del proceso *REST*. Una discusión con mayor detalle se encuentra en el capítulo 7.

Proceso de control del acceso - proceso de control del acceso

El *Middleware TAO* permite la invocación de métodos remotos, RMI por sus siglas en inglés [9, 28]. Los objetos *CORBA* pueden recibir una gran variedad de tipos de

datos, pues depende de la interfaz que se le especifique mediante *IDL* [10–12].

Proceso de control del acceso - proceso de control del sistema experto (*CORBA*)

El mecanismo de comunicación es el mismo al utilizado entre los procesos de control del acceso y aplican las mismas características.

Proceso de control del acceso - proceso de control del sistema experto (*RabbitMQ*)

El *Middleware RabbitMQ* que se utiliza en la arquitectura sólo permite enviar mensajes conformados por cadenas texto, por esta razón se utiliza la misma estrategia que en el caso de la comunicación entre el proceso *REST* y el proceso de control del acceso. Existe una diferencia, en este mecanismo es necesario poder enviar archivos, los cuales pueden ser binarios; para solucionar este inconveniente, los datos son codificados en *Base64 [37]* para ser enviados como una cadena texto y posteriormente son decodificados, obteniendo la información binaria original. Para más información consultar capítulo 6.

Proceso de control del sistema experto - proceso del sistema experto

La comunicación entre el proceso de control del sistema experto y los procesos del sistema experto se realiza mediante el uso del *Middleware ZeroMQ*. Se presentan las mismas características de los mensajes utilizados entre el proceso *REST* y el proceso de control del acceso. En el capítulo 7 se explica con mayor detalle la estrategia que se utiliza.

3.2.4. Interacción

Los servicios que el sistema de la arquitectura brinda, se componen de un conjunto de acciones que los procesos de las entidades en el sistema realizan, los cuales colaboran para el funcionamiento correcto de los servicios. A continuación se muestran los algoritmos que siguen los procesos al interactuar cuando se procesan las peticiones de los usuarios y otras acciones para el funcionamiento del sistema de la arquitectura.

Inicio de una instancia de la entidad de acceso

1. Se inicia el proceso *REST*, el cual recibe todos los parámetros que necesita para funcionar y además recibe todos los parámetros para poder iniciar el proceso de control del acceso al cual estará asociado.

2. El proceso *REST* le solicita al sistema operativo anfitrión un puerto de comunicación libre, en el cual el proceso de control del acceso iniciara el *Socket ZeroMQ* para comunicarse con el proceso *REST*. Después se inicia el proceso de control del acceso y el proceso *REST* se conecta al *Socket*.

3. Al iniciar el proceso de control del acceso crea el objeto *CORBA* y lo registra en el servicio de nombres.

4. El proceso de control del acceso inicializa todos los módulos que le permiten realizar sus funciones.

5. El Proceso *REST* inicia los servicios para atender las peticiones de los usuarios.

6. Cuando el proceso de control del acceso termina de iniciar todos sus módulos, empieza a buscar los objetos *CORBA* que se han registrado en el servicio de nombres; intenta comunicarse con ellos y si responden los agrega a sus registros de referencias remotas, en caso contrario los elimina del servicio de nombres. Cuando se detecta una instancia de la entidad de acceso, se le solicita la prioridad con la que fue iniciado, pues es un dato que es utilizado en otras acciones; en caso de ser una entidad del sistema experto, sólo se agrega.

7. Una vez que el proceso de control del acceso ha concluido la primera búsqueda de objetos *CORBA*, solicita al sistema gestor de bases de datos toda la información del funcionamiento del sistema; después procesa toda la información que obtiene, para que de esta forma tenga los mismos registros que las instancias de la entidad de acceso que ya se encontraban en ejecución.

8. El módulo de la plataforma de distribución de archivos en el proceso de control del acceso, se conecta al Clúster *RabbitMQ* con los parámetros que se agregan cuando inicia la ejecución del programa; también recibe la cantidad de canales

con los que trabajará, los cuales son registrados en el servicio de *RabbitMQ*. Es importante que antes de iniciar cualquier instancia de la entidad del sistema experto, también conocidas como nodos expertos, se inicie una instancia de la entidad de acceso; pues son las encargadas en registrar los canales y sin ellos los nodos expertos no pueden iniciar correctamente. Los canales de comunicación son compartidos por todas las instancias de la entidad de acceso.

Nota: algunos módulos se inicializan de forma concurrente.

En la figura 3.5 se muestran las acciones que se realizan en el inicio de una instancia de la entidad de acceso.

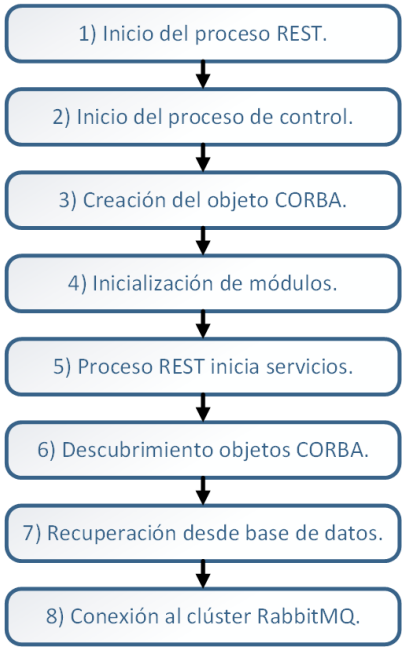

Figura 3.5: Inicio de una instancia de la entidad de acceso.

Inicio de una instancia de la entidad del sistema experto

1. Se inicia el proceso de control del sistema experto, el cual recibe todos los parámetros que necesita para funcionar y para iniciar cada proceso del sistema experto a los cuales estará asociado.

2. El proceso de control del sistema experto crea los directorios en el sistema

de archivos, en donde se almacenan los archivos que recibe, y utiliza en los procesos del sistema experto.

3. Al iniciar el proceso de control del sistema experto, éste crea el objeto *CORBA* y lo registra en él servicio de nombres. Después inicializa todos los módulos que le permiten realizar sus funciones.

4. El proceso de control del acceso inicializa todos los módulos que le permiten realizar sus funciones.

Nota: algunos módulos se inicializan de forma concurrente.

En la figura 3.6 se muestran las acciones que se realizan en el inicio de una instancia de la entidad experto.

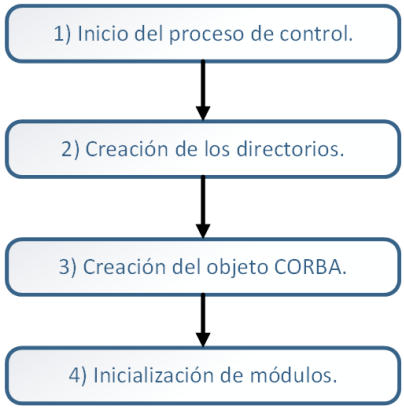

Figura 3.6: Inicio de una instancia de la entidad experto.

Registro de la definición de un archivo de reglas

1. El usuario envía a alguna instancia de la entidad de acceso, también llamada nodo de acceso, la petición del registro del archivo de definición. La petición es procesada por el Proceso *REST* en la instancia, en ese momento se recibe el archivo.

2. El proceso *REST* solicita al proceso de control del acceso, el procesamiento del archivo de definición, en la petición se adjunta la ruta en el sistema de archivos local en la que se encuentra el archivo.

3. El proceso de control del acceso lee el contenido del archivo, obtiene el nombre del archivo de reglas al cual está asociado la definición y los nombres de los archivos *scripts* que utiliza dicho archivo de reglas.

4. El proceso de control del acceso solicita a su módulo del almacenamiento persistente el registro de la información; si no existe el registro para el archivo de reglas, se crea un nuevo registro, y si existe se actualiza la información.

5. El proceso de control del acceso verifica si ya tiene un registro previo del archivo de reglas, si no lo tiene, crea el registro nuevo con los datos recibidos, en caso contrario sólo lo actualiza.

6. El proceso de control del acceso propaga los datos del archivo de definición hacia las demás instancias de la entidad de acceso.

7. El proceso de control del acceso le solicita al módulo del almacenamiento persistente registrar que se han propagado los datos del archivo de definición a los otros nodos de acceso.

Nota: un archivo de definición contiene el nombre del archivo de reglas y una lista de nombres de archivos *scripts*, los cuales son utilizados por las reglas que contiene el archivo de reglas asociado.

En la figura 3.7 se muestran las acciones que se realizan en el registro de la definición de un archivo de reglas.

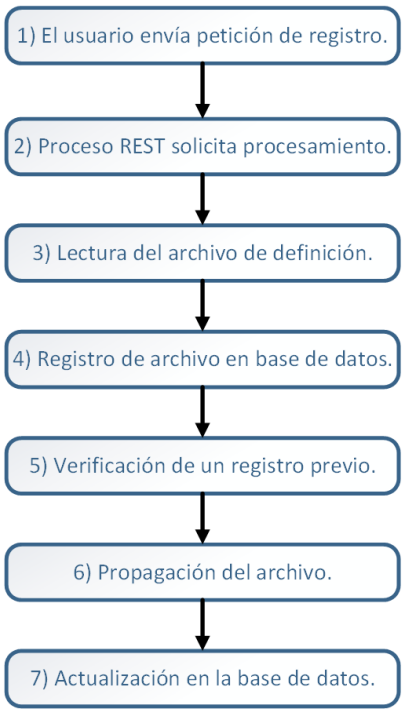

Figura 3.7: Registro de la definición de un archivo de reglas.

Registro de un archivo de reglas

1. El usuario envía a alguna instancia de la entidad de acceso, la petición del registro de un archivo de reglas. La petición es procesada por el Proceso *REST* en la instancia, en ese momento se recibe el archivo. Es posible recibir el archivo de reglas en un nodo de acceso distinto al que recibió el archivo de definición.

2. El Proceso *REST* solicita al proceso de control del acceso, el procesamiento del archivo de reglas, en la petición se adjunta la ruta en el sistema de archivos local en la que se encuentra el archivo. El usuario puede elegir en qué instancia de la entidad del sistema experto, también llamados nodos expertos, desea utilizar para trabajar el archivo de reglas que envía; para que esto sea posible, es necesario que en la petición también agregue el identificador del nodo experto destino.

3. El proceso de control del acceso le solicita al módulo del almacenamiento persistente respaldar el archivo de reglas; se verifica si ya existe un registro previo

del archivo de reglas, ya sea por el procesamiento previo de un archivo de definición o por el registro de una versión anterior del archivo de reglas. Si ya existe el registro, se actualizan los datos, en caso contrario se crea el registro. El módulo regresa el identificador del archivo de reglas.

4. Dependiendo de la solicitud del usuario y si existe un registro previo del archivo de reglas, se realiza una de las siguientes acciones:

   a) Si el usuario no especificó un nodo experto destino y no existía registro previo del archivo de reglas, se registra el archivo de reglas en el almacenamiento persistente; se incluyen datos como el identificador del nodo de acceso encargado del registro y la cantidad de reglas que contiene el archivo.

   b) Si el usuario especificó un nodo experto destino y no existía registro previo del archivo de reglas, se registra el archivo de reglas en el almacenamiento persistente; se incluyen datos como el identificador del nodo experto destino, el identificador del nodo de acceso encargado del registro y la cantidad de reglas que contiene el archivo.

   c) Si el usuario no especificó un nodo experto destino y existía registro previo del archivo de reglas, se actualiza el registro del archivo de reglas en él almacenamiento persistente; se actualizan datos como el identificador del nodo de acceso encargado del registro y la cantidad de reglas que contiene el archivo.

   d) Si el usuario especificó un nodo experto destino y existía registro previo del archivo de reglas, se actualiza el registro del archivo de reglas en el almacenamiento persistente; se actualizan datos como el identificador del nodo experto destino, el identificador del nodo de acceso encargado del registro y la cantidad de reglas que contiene el archivo.

5. Dependiendo de la solicitud del usuario, y si existe un registro previo del archivo de reglas, se realiza una de las siguientes acciones:

   a) Si el usuario no especificó un nodo experto destino y no existía registro previo del archivo de reglas, el proceso de control del acceso elige el nodo

experto destino utilizando la política de asignación en operación al momento del registro del archivo de reglas. Se crea el registro del archivo de reglas en los módulos de control del proceso y le solicita al módulo del almacenamiento registrar que ha concluido la etapa de procesamiento del archivo.

b) Si el usuario especificó un nodo experto destino y no existía registro previo del archivo de reglas, se crea el registro del archivo de reglas en los módulos de control del proceso y le solicita al módulo de almacenamiento registrar que ha concluido la etapa de procesamiento del archivo.

c) Si el usuario no especificó un nodo experto destino y existía registro previo del archivo de reglas, se actualiza el registro del archivo de reglas en los módulos de control del proceso y le solicita al módulo del almacenamiento registrar que ha concluido la etapa de procesamiento del archivo.

d) Si el usuario especificó un nodo experto destino y existía registro previo del archivo de reglas, se elimina el archivo de reglas en el nodo experto y se termina el proceso del sistema experto contenedor; se remueve el registro en los módulos del proceso y se replica la acción a los otros nodos de acceso. Después se crea el registro del archivo de reglas con el nodo experto seleccionado en los módulos de control del proceso y le solicita al módulo del almacenamiento registrar que ha concluido la etapa de procesamiento del archivo.

6. El proceso de control del acceso inicia un servicio de transferencia, el cual le solicita al proceso de control del sistema experto del nodo experto destino, iniciar un servicio para la transferencia de archivos. Ambos servicios de transferencia trabajan en conjunto para transferir el archivo de reglas y los archivos *scripts* que son utilizados por las reglas que contiene, primero se transfieren los archivos *scripts*. Al terminar la transferencia de los archivos, se detienen y liberan los servicios de transferencia.

7. Cuando los archivos son recibidos en el nodo experto destino, se determina si ya existe un proceso del sistema experto para dicho archivo de reglas. Si no existe un proceso previo, se inicia un nuevo proceso asociado con un *Socket ZeroMQ*

y carga el archivo de reglas en su sistema experto. Si existe un proceso previo, se le solicita eliminar todas las reglas en funcionamiento y cargar el archivo de reglas en su sistema experto.

8. El proceso de control del acceso le solicita al módulo del almacenamiento registrar que ha concluido la etapa de transferencia del archivo de reglas.

En la figura 3.8 se muestran las acciones que se realizan en el registro de un archivo de reglas.

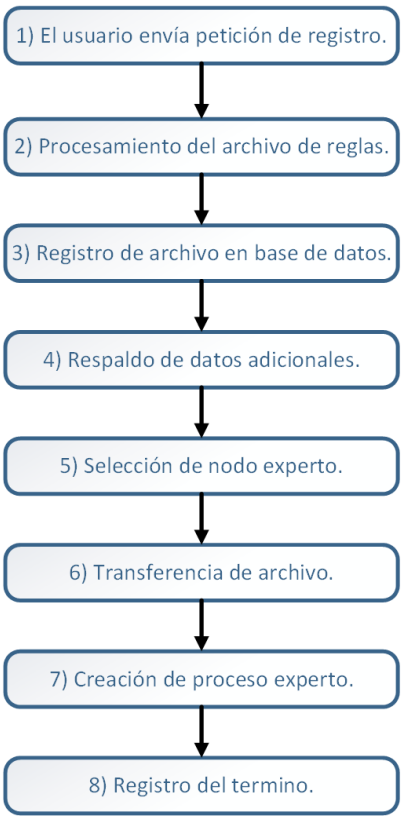

Figura 3.8: Registro de un archivo de reglas.

Registro de un archivo de hechos

1. El usuario envía a alguna instancia de la entidad de acceso, la petición del registro de un archivo de hechos. La petición es procesada por el Proceso *REST* en la instancia, en ese momento se recibe el archivo.

2. El proceso de control del acceso genera un nombre único para el archivo de hechos y solicita a su módulo del almacenamiento persistente el registro del archivo de hechos.

3. El proceso de control del acceso solicita al módulo de la plataforma de distribución de archivos la transferencia del archivo de hechos a todos los nodos expertos, internamente el módulo decide cuál es el canal con menor carga de trabajo y asigna la transferencia a dicho canal.

4. El proceso de control del sistema experto de cada nodo experto recibe el archivo de hechos, le indica a cada proceso del sistema experto que cargue el archivo e inicie el proceso de inferencia.

5. El proceso de control del acceso le solicita al módulo del almacenamiento registrar que ha concluido la etapa de transferencia del archivo de hechos.

En la figura 3.9 se muestran las acciones que se realizan en el registro de un archivo de hechos.

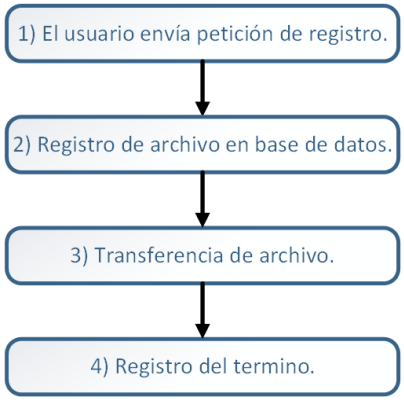

Figura 3.9: Registro de un archivo de hechos.

3.2.5. Escalabilidad

La arquitectura permite agregar nuevas instancias de las entidades en cualquier momento, sin que sea necesario cambiar la configuración en las instancias que se encuentran en funcionamiento, esto permite tener una mejor escalabilidad del sistema.

De otra forma, al agregar nuevas instancias sería cada vez más complicado, pues la cantidad de instancias a las que sería necesaria actualizar su configuración se incrementaría conforme se agregan instancias. La capacidad para agregar nuevas instancias permite ir agregando nuevos recursos al sistema, esto es conveniente en caso de que aumente la carga de trabajo y las instancias actuales no puedan atender todas las peticiones.

Agregar instancias de la entidad de acceso permite al sistema distribuido propuesto, atender mayor cantidad de peticiones de los usuarios. Pero existen algunos problemas que limitarían la escalabilidad. El primer problema radica en la cantidad limitada de conexiones del sistema gestor de bases de datos, el cual es un servicio centralizado. El segundo problema se presenta por la necesidad de los procesos de control de acceso de replicar las acciones que han realizado, a los otros procesos de control de acceso; esto ocasiona el aumento de mensajes de coordinación conforme se agregan más instancias de la entidad de acceso.

Aumentar la cantidad de instancias de la entidad del sistema experto puede presentar problemas de escalabilidad, al llegar al límite de las capacidades de los servicios de *RabbitMQ*, aun utilizando un clúster para el envío de mensajes; mientras que no presenta problemas para las demás funciones de administración, diferentes al procesamiento de los archivos de hechos, pues cada solicitud sólo se encarga a una instancia de la entidad del sistema experto.

3.2.6. Detección y recuperación de fallas

Los sistemas distribuidos están compuestos por una gran cantidad de elementos y cualquiera de ellos puede fallar, es importante tomar esto en cuenta durante la etapa de diseño. El diseño de la arquitectura contempla respaldar toda la información del funcionamiento del sistema; cuando un proceso solicita la ejecución de un método en un objeto remoto y falla, en ese momento se descarta dicha instancia e inicia el proceso de recuperación y reasignación de la información de la que se encargaba dicha instancia. Esta estrategia tiene como objetivo minimizar el tiempo de recuperación ante las fallas.

A continuación se mencionan y describen las fallas que la arquitectura puede detectar, así como las estrategias para volver a su funcionamiento normal.

Falla o sin conexión del Proceso *REST*

Cuando el Proceso *REST* de una instancia de la entidad de acceso falla o no hay conexión hacia ella, la arquitectura no posee un mecanismo para reiniciarlo; el usuario necesita solicitar los servicios a otro nodo de acceso. Es importante mencionar que el proceso de control del acceso puede seguir funcionando, trabajando en conjunto con los otros procesos de control del acceso; si el administrador del sistema reinicia el Proceso *REST*, esté terminará el proceso de control del acceso anterior e iniciará uno nuevo.

El proceso de control del acceso no responde a las peticiones del Proceso *REST*

Es posible que el proceso de control del acceso no responda a las peticiones del Proceso *REST*, esto puede ser ocasionado por la falla del proceso. Cuando esto ocurre, el Proceso *REST* termina todo proceso en el sistema que se ha iniciado con los parámetros utilizados para el Proceso *REST* anterior; después, inicia un nuevo proceso asociado a otro puerto del sistema. Mientras se restablece el proceso de control del acceso, el Proceso *REST* no atiende peticiones de los usuarios.

Nodo de acceso no responde a peticiones remotas

Es posible que un nodo de acceso deje de responder a las peticiones remotas de otros nodos de acceso. Cuando esto ocurre, el nodo de acceso de mayor prioridad será el encargado de recuperar la información que se encontraba en él nodo de acceso que falló y continuará con su procesamiento. A continuación se muestran las acciones que ejecuta el nodo de acceso de mayor prioridad al detectar la falla de otro nodo de acceso; si el nodo de acceso de mayor prioridad es el que falló, el siguiente en prioridad es el encargado de realizar los procedimientos.

1. El nodo de acceso en funcionamiento elimina del servicio de nombres, la referencia del objeto *CORBA* del nodo de acceso que falló.

2. El nodo de acceso en funcionamiento obtiene los archivos de definición, hechos y reglas, los cuales no fueron procesados ni transferidos con éxito por el nodo de

acceso que falló; el nodo de acceso solicita la información y archivos al módulo del almacenamiento persistente.

3. El nodo de acceso procesa los archivos de definición obtenidos por el módulo del almacenamiento persistente.

4. El nodo de acceso trabaja con los archivos de reglas obtenidos por el módulo del almacenamiento persistente, continua en el punto en donde se quedó el nodo de acceso que falló. Si el archivo de reglas no está asignado a un nodo experto, el nodo de acceso elige uno basándose en la política de asignación en funcionamiento y lo registra en él almacenamiento persistente. Si el archivo de reglas no se ha transferido, el nodo de acceso lo transfiere al nodo experto destino y después se registra en el almacenamiento persistente el término de la transferencia. Cuando el nodo experto obtiene el archivo de reglas, verifica si necesita crear un nuevo proceso del sistema experto o requiere volver a cargar el archivo de reglas en el proceso del sistema experto existente.

5. El nodo de acceso trabaja con los archivos de hechos obtenidos por el módulo del almacenamiento persistente, utiliza el módulo de la distribución de archivos para transferir los archivos a todos los nodos expertos; al concluir la transferencia de cada archivo, se registra en el almacenamiento persistente.

Nodo experto no responde a peticiones remotas

Es posible que un nodo experto deje de responder a las peticiones remotas de los nodos de acceso. Cuando esto ocurre, el nodo de acceso de mayor prioridad será el encargado de recuperar la información que se encontraba en el nodo experto que falló y asignarla en otros nodos expertos. A continuación se muestran las acciones que ejecuta el nodo de acceso de mayor prioridad al detectar la falla del nodo experto.

1. El nodo de acceso elimina del servicio de nombres, la referencia del objeto *CORBA* del nodo experto que falló.

2. El nodo de acceso obtiene los archivos de reglas y *scripts*, los cuales se encontraban en el nodo experto que falló; el nodo de acceso solicita la información y archivos al módulo del almacenamiento persistente.

3. El nodo de acceso distribuye los archivos de reglas en los nodos expertos que se encuentran en funcionamiento; cada vez que se elige en qué nodo experto se asigna el archivo de reglas, se transfieren los archivos *scripts* que utilizan las reglas y después se transfiere el archivo de reglas.

4. Cada nodo experto que se ha elegido, recibe los archivos *scripts* y archivos de reglas que le corresponden; después se inician los Procesos del sistema experto necesarios y cargan los archivos de reglas.

3.3. Herramientas para la implementación

La arquitectura distribuida que se presenta tiene como componente el *Middleware* TAO [15], el cual se utiliza como la capa de comunicación del sistema distribuido; todas las instancias en la arquitectura que se propone se comunican utilizando los objetos *CORBA* [10–12] que provee el Middleware y utiliza el lenguaje de programación C++ [17]. Toda comunicación remota entre las instancias se realiza mediante la ejecución de un método del objeto remoto en la instancia, que es resuelto de manera transparente por TAO, sólo es necesario tener la referencia del objeto *CORBA*. Para que el *Middleware* pueda resolver las referencias, es necesario el Servicio de Nombres, que tiene como objetivo resolver y generar las referencias a cada uno de los objetos distribuidos *CORBA* que se registran en él; el problema rádica que al ser un servicio centralizado, si llega a fallar el middleware no podría comunicar a las instancias del sistema distribuido.

La instancia de acceso, en el diseño propuesto, posee un proceso con el servidor REST [14] que atiende las peticiones de los usuarios. Para la implementación del programa que provee el servidor *REST* se utiliza la biblioteca *Civetweb* [1], la cual puede atender múltiples peticiones a la vez y se utiliza con el lenguaje de programación C++.

La arquitectura distribuida que se propone en el presente trabajo de tesis está compuesta por múltiples instancias de las entidades que realizan todas las operaciones para proveer los servicios de la arquitectura. Cada entidad está integrada por dos o más procesos en una misma computadora, los cuales deben comunicarse para trabajar

[1] `http://sourceforge.net/projects/civetweb/`

cooperativamente; para este fin, se utiliza el *Middleware* de comunicación ZeroMQ [34], la cual permite trabajar con *Sockets* bidireccionales locales. ZeroMQ se utiliza en el diseño propuesto pues puede ser utilizada en más de 40 lenguajes de programación; esto permite utilizar una gran variedad de sistemas expertos con la arquitectura distribuida; esta es la principal razón para su utilización en él sistema.

Cada proceso que compone la arquitectura distribuida propuesta está compuesto por múltiples módulos, cada uno de los cuales debe atender una gran variedad de peticiones de los servicios que provee. Para solucionar el problema anterior se utiliza el patrón *Threads Pool* [35], el cual presenta una estrategia eficiente para este tipo de problemas; en la implementación de los programas de la arquitectura distribuida se utilizó la biblioteca Boost [17] y la clase *Threads Pool* [2].

Una parte esencial del diseño de la arquitectura distribuida que se propone es el respaldo y recuperación de la información del funcionamiento del sistema. Existen varios sistemas gestores de bases de datos que son de los más utilizados, e.g., Postgresql [3], Oracle[4] y MySQL [5]; es importante que la arquitectura se pueda utilizar con cualquiera de ellos. Para solucionar este problema se utiliza la biblioteca Libzdb[6], la cual es una biblioteca para la conexión a bases de datos utilizando el patrón Pool de conexiones.

Una parte esencial de arquitectura distribuida que se propone, es la plataforma para la distribución de archivos, sin ella la escalabilidad estaría comprometida. La plataforma basa su funcionamiento en un clúster *RabbitMQ*, cada nodo de la arquitectura puede acceder a los servicios del *Middleware RabbitMQ* [36] desde cualquier nodo del clúster. Los nodos de la arquitectura utilizan la biblioteca SimpleAmqpClient [7] para conectarse al clúster *RabbitMQ*.

[2] `http://threadpool.sourceforge.net/`
[3] `http://www.postgresql.org/`
[4] `http://www.oracle.com/index.html`
[5] `http://www.mysql.com/`
[6] `http://www.tildeslash.com/libzdb/`
[7] `https://github.com/alanxz/SimpleAmqpClient`

3.4. Fortalezas

La distribución de los servicios en varias instancias asegura que siempre exista alguna capaz de atender las peticiones de los usuarios. El diseño de arquitectura distribuida que se propone permite agregar nuevas instancias en el momento que el administrador del sistema lo decida, con la finalidad de aumentar la disponibilidad de los servicios; además no es necesario que modifique la configuración de las instancias que ya se encuentran en funcionamiento.

El diseño de la arquitectura distribuida que se propone permite la recuperación de la información de una instancia cuando falla, el proceso es automático; una vez que se recupera la información, se asigna en las instancias que se encuentren en funcionamiento. Esta característica tiene como objetivo disminuir la pérdida de los archivos de reglas en caso de que se presente alguna falla, esto es posible mediante el respaldo de la información en un sistema gestor de bases de datos.

La arquitectura distribuida utiliza el servicio de nombres de *CORBA*, el cual permite a las instancias de la entidad de acceso detectar automáticamente la presencia de todas las instancias que se encuentran en funcionamiento en él sistema. Con esta estrategia se permite que las instancias no requieran una configuración especial en donde se indique con quienes va a trabajar y cooperar.

La plataforma de distribución de archivos permite que se pueda agregar una gran cantidad de instancias de la entidad del sistema experto, también conocidos como nodos expertos, sin ocasionar problemas para la transferencia de los archivos de reglas a todos los nodos expertos. La plataforma de distribución utiliza el patrón emisor/suscriptor, de esta forma, el envío de un mensaje es entregado a todos los nodos expertos; de esta forma se evita la necesidad de enviar mensajes cada nodo experto y saturar la red de comunicación.

3.5. Debilidades

La utilización del servicio de nombres de *CORBA* brinda una funcionalidad importante de la arquitectura distribuida que se propone, pero al ser un servicio centralizado genera problemas de escalabilidad y tolerancia a fallas. En caso de que falle el servicio, todas las instancias de la arquitectura no podrían seguir funcionando

correctamente, perdiendo toda capacidad del sistema distribuido.

El servicio de nombres *CORBA* puede ocasionar problemas de escalabilidad en la arquitectura distribuida, la cantidad de instancias funcionando en la arquitectura distribuida está limitada a la cantidad de objetos *CORBA* que se pueden registrar en el servicio; es necesario que todas las instancias registren su objeto en el servicio de nombres.

La arquitectura que se propone distribuye todas las acciones de administración entre todos los nodos de acceso; es necesario que cada nodo de acceso replique todas las acciones que realiza, a todos los otros nodos de acceso. Esta estrategia puede ocasionar problemas de escalabilidad, pues el incremento en la cantidad de nodos de acceso, genera un incremento en la cantidad mensajes de administración; lo cual puede llegar a un punto en él que no se pueda escalar más.

3.6. Resumen

Lo esencial en las arquitecturas distribuidas es repartir las funciones y responsabilidades del sistema. Es recomendable que en las arquitecturas siempre existan al menos dos instancias que sean capaces de encargarse de dichas funciones, pues si una falla la otra puede seguir brindando el servicio, pero es conveniente que haya más instancias pues de esta forma la carga de trabajo puede ser repartida entre todas.

Un aspecto importante en los sistemas expertos distribuidos, es cómo los usuarios pueden acceder a los servicios que brindan; es muy importante que los sistemas puedan atender a los usuarios sin importar la plataforma o sistema operativo que utilicen. Por esta razón es necesario utilizar un protocolo de comunicación que sea ampliamente utilizado y que no sea dependiente de la plataforma; también es importante que el mecanismo de comunicación permita atender a varios usuarios a la vez.

Una de las características que necesita agregar el sistema distribuido, al sistema experto que controla, es la tolerancia a fallas, pero el sistema no puede evitar que el sistema experto falle; para evitar que el usuario note que se presentó una falla del sistema experto o la computadora en donde se ejecuta, el sistema distribuido debe encargarse de recuperar la información del usuario y asignarla a otro sistema experto

que se encuentre en funcionamiento. Para lograr este objetivo es necesario utilizar un sistema de almacenamiento de información que pueda ser accedido por las instancias de control del sistema; un sistema gestor de base de datos cumple con este propósito y puede atender varias peticiones a la vez.

La arquitectura distribuida que se propone está compuesta por dos entidades distintas:

- Entidad de acceso: es la entrada del usuario al sistema mediante peticiones *REST* y también se encarga de administrar el funcionamiento de la arquitectura distribuida. La entidad de acceso se compone por 2 procesos que trabajan en conjunto; el primer proceso es el encargado de recibir las peticiones *REST* y el segundo proceso se encarga de la administración del sistema de la arquitectura distribuida.

- Entidad del sistema experto: se encarga de administrar y controlar los procesos del sistema experto que se utiliza en él proceso de inferencia. La entidad se compone por dos tipos de procesos; el proceso que administra el funcionamiento de la entidad y los procesos del sistema experto. La comunicación entre estos procesos se realiza utilizando el *Middleware ZeroMQ*.

La comunicación entre cada una de las instancias de las arquitectura distribuida está basada en él *Middleware TAO* de la especificación *CORBA*. Toda comunicación remota entre las instancias se realiza mediante la ejecución de un método del objeto remoto en la instancia, que es resuelto de manera transparente por *TAO*; sólo es necesario tener la referencia del objeto *CORBA*. Para que el *Middleware* pueda resolver las referencias, es necesario el servicio de nombres, que tiene como objetivo resolver y generar las referencias a cada uno de los objetos distribuidos *CORBA* que se registran en él; el problema radica en que al ser un servicio centralizado, si llega a fallar el sistema distribuido no podría establecer la comunicación a las instancias del sistema distribuido.

Una de las debilidades más significativas del diseño propuesto es el servicio de nombres de *CORBA*, puede comprometer todas las características distribuidas que se le agregan al sistema experto, esto debido a la naturaleza centralizada del servicio.

Capítulo 4

Control y administración de la arquitectura distribuida

En este capítulo se aborda con mayor detalle y profundidad los mecanismos que la arquitectura distribuida propuesta tiene, se explica cómo se mantiene el control del sistema distribuido, la forma en la que las entidades se comunican entre sí y como se organizan para mantener el control sobre cada una de las instancias en el sistema distribuido. También se explican las estrategias que se utilizan cuando alguna de las instancias falla y no se puede recuperar; las cuales tienen como objetivo agregar la tolerancia a fallas.

En el presente capítulo se describe cómo funciona el proceso de la distribución de la carga de trabajo entre las instancias encargadas del proceso deínferencia, la arquitectura distribuida propuesta utiliza las siguientes políticas: a) *Round-Robin*, b) por cantidad de archivos de reglas en los nodos expertos, c) por cantidad de reglas en los nodos expertos o d) la elección por parte del usuario acerca del nodo experto en el cual el archivo de reglas será asignado; las primeras 3 son una configuración global de la arquitectura distribuida y la cuarta es una opción que tiene el usuario en el momento de realizar una solicitud del registro de un archivo de reglas al sistema.

4.1. Estrategia

La arquitectura distribuida que se propone está diseñada para agregar varias características de un sistema distribuido a un sistema experto; las características que se agregan son: escalabilidad, tolerancia a fallas y alta disponibilidad. Para agregar las características deseadas se utiliza el patrón divide y vencerás [38]; el diseño de la arquitectura divide y reparte todos los servicios entre todas las instancias de las entidades que componen la arquitectura distribuida.

Distribuir todos los servicios entre todas las instancias de las entidades permite agregar las características deseadas de las siguientes formas. La escalabilidad se añade a la arquitectura distribuida mediante la capacidad para agregar nuevas instancias de las entidades en cualquier momento; la cantidad de instancias se limita al número de objetos *CORBA* [10–12] que su servicio de nombres soporta. La tolerancia a fallas se agrega al diseño que se propone mediante la capacidad de recuperación de toda la información que contiene una instancia que falló, la información se recupera y se reparte entre las instancias que se encuentren en funcionamiento. La alta disponibilidad de la arquitectura está basada en la premisa que siempre existirá una instancia de cada entidad, de esta forma siempre existirá alguien capaz de atender las peticiones de los usuarios. Con estas estrategias se agregan las características deseadas al diseño de arquitectura distribuida propuesto.

4.2. Análisis

La arquitectura distribuida propuesta consta de dos entidades compuestas y cada una tiene dos procesos, los cuales cooperan entre sí en la misma instancia y con otros procesos que residen en otras instancias; cada proceso debe atender varias peticiones de distintos tipos y ejecutar una gran cantidad de procedimientos con distintas funcionalidades al mismo tiempo.

Los programas de los procesos que conforman la arquitectura distribuida propuesta están diseñados utilizando un enfoque de programación modular [39], todos los procedimientos con funcionalidades afines o similares se agrupan en módulos. Cada módulo tiene un objetivo específico, cuando necesita realizar alguna tarea con un enfoque distinto al del módulo, se apoya en los servicios de otros módulos para alcan-

zar sus objetivos. Los módulos deben ser lo más independientes posible entre sí, pues si es necesario modificar el funcionamiento de alguno, no debe ocasionar problemas en los módulos que lo utilizan; esta estrategia de diseño permite que los sistemas se mejoren sin afectar su funcionamiento actual, pues sólo se necesita reemplazar los módulos a mejorar.

La arquitectura distribuida provee de los siguientes servicios.

- Recibir las peticiones *REST* [14] de los usuarios, regresando una respuesta a cada solicitud.
- En las instancias de la entidad de acceso, el proceso *REST* inicia y administra el proceso de control del acceso; en caso de falla, inicia uno nuevo.
- Gestionar y administrar la comunicación entre los procesos de la entidad de acceso, el proceso *REST* y el proceso de control del acceso.
- Descubrir los objetos *CORBA* de las otras instancias presentes en el sistema distribuido.
- Gestionar y comprobar el estado de los objetos *CORBA* que se han descubierto en el sistema distribuido.
- Gestionar los archivos *scripts*, reglas y hechos que los usuario registran en el sistema distribuido.
- Registrar los archivos de definicón de reglas, en los cuales se declaran los archivos *scripts* que utilizan las reglas de un archivo.
- Asignar las nuevas peticiones de los archivos de reglas en las instancias de la entidad del sistema experto, basándose en la política de asignación en uso.
- Recuperar la información de trabajo de las instancias que han fallado y reasignarla en otras instancias que se encuentren en funcionamiento.
- Distribuir los archivos de hechos a todos los nodos expertos para ser utilizados en el proceso de inferencia del sistema experto.

- Cada instancia de la entidad del sistema experto se encarga de iniciar y administrar los procesos del sistema experto con los que trabaja.

Estos servicios se distribuyen entre los distintos módulos que componen los programas de las entidades de la arquitectura distribuida, los cuales deben cooperar entre sí para lograr que el sistema distribuido trabaje correctamente. Algunos módulos tienen como objetivo encapsular los mecanismos que les permiten a los procesos comunicarse con otros procesos, ya sea que pertenezcan a la misma instancia o de otras instancias; varias instancias pueden ejecutarse en la misma máquina, lo cual no representa problemas administrativos. La cantidad de instancias que una máquina puede ejecutar, está limitada por sus especificaciones de *hardware*.

La arquitectura distribuida tiene como estrategia tener en todo momento, varias instancias de las entidades, esto tiene como objetivo garantizar una alta disponibilidad de todos los servicios del sistema distribuido. Las instancias de la entidad de acceso son de vital importancia para la arquitectura, pues se encargan de la administración y control de los servicios del sistema distribuido; si en algún momento, no existieran instancias de esta entidad, el sistema no podría seguir funcionando.

Las instancias de la entidad de acceso requieren coordinarse y tener la misma información del estado del sistema; por esta razón es necesario que toda acción que realice una instancia de la entidad se replique a las demás instancias. Una instancia puede fallar antes de poder replicar una acción, por tal motivo, siempre se registran las acciones en el almacenamiento persistente.

4.3. Diseño

El diseño de la arquitectura distribuida que se propone en el presente trabajo de tesis utiliza un enfoque modular, agrupando todos los servicios similares en módulos; el patrón permite realizar cambios en un módulo en particular sin afectar el funcionamiento de los otros módulos. Las instancias de la arquitectura distribuida están conformadas por los módulos que cooperan entre sí, lo cual permite que funcionen todos los procedimientos necesarios para proveer los servicios de la arquitectura.

A continuación se muestra la organización de los módulos en las entidades de la arquitectura distribuida y los procesos que las conforman.

- Entidad de acceso
  - Proceso *REST*
    - Módulo de administración *REST*
    - Módulo de comunicación
  - Proceso del control de acceso
    - Módulo de control y administración
    - Módulo de comunicación
    - Módulo de descubrimiento de nodos
    - Módulo contenedor de nodos
    - Módulo de distribución *Multicast*
- Entidad de sistema experto
  - Proceso de Control de nodo experto
    - Módulo de control y administración
    - Módulo de comunicación
    - Módulo de distribución *Multicast*
  - Proceso de sistema experto
    - Módulo de comunicación
    - *Framework* del sistema experto

En la figura 4.1 se puede observar con mayor detalle la organización de los módulos en la arquitectura distribuida, así como la forma en que interactúan entre sí; también se muestra cada *Middleware* que utiliza para funcionar. Es importante resaltar que algunos módulos se encuentran presentes en ambos procesos de las entidades, esto ocurre con algunos módulos que están diseñados para transferir información.

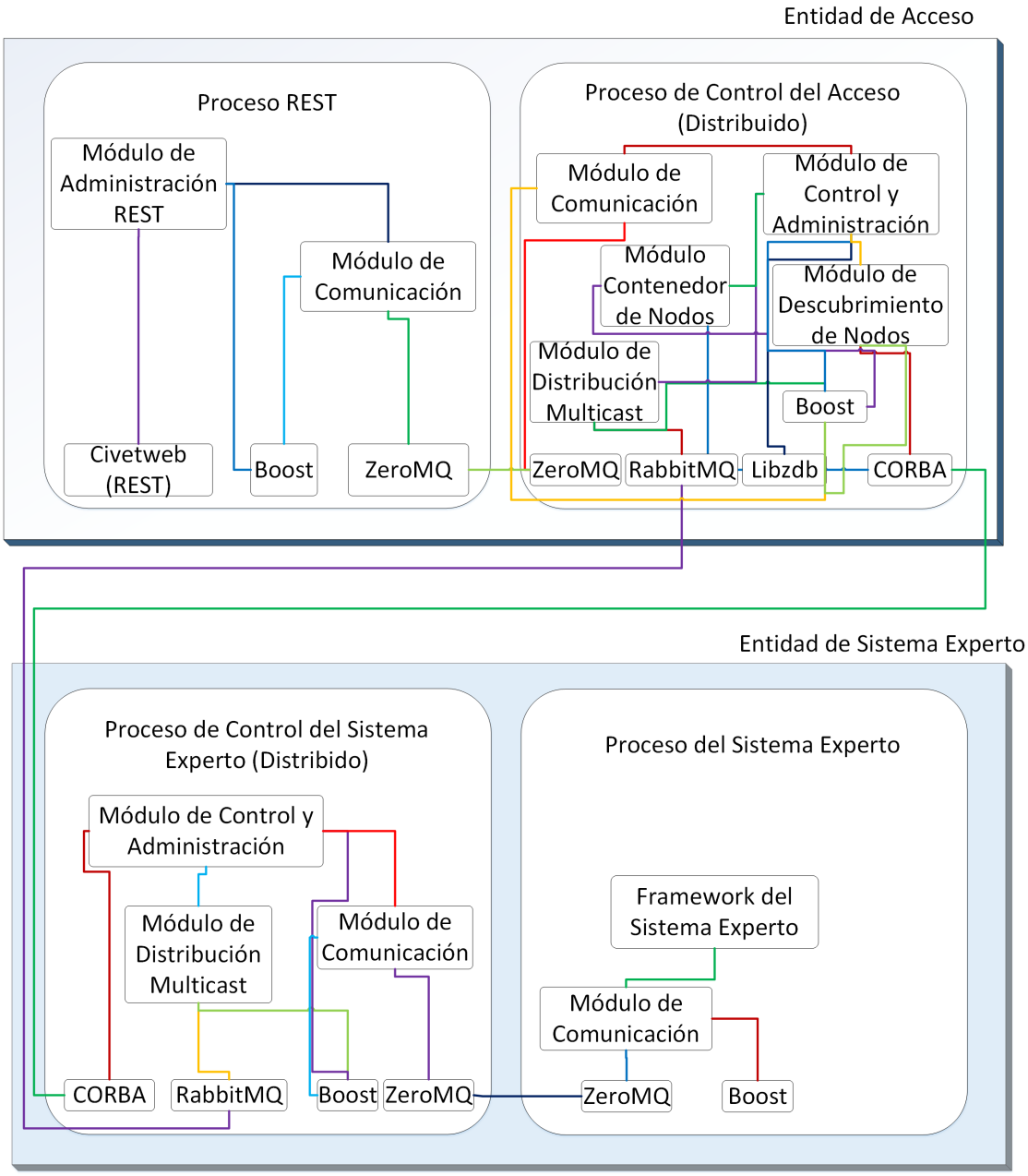

Figura 4.1: Módulos de la arquitectura distribuida.

4.3.1. Módulos de la entidad de acceso

En la arquitectura distribuida que se propone, la entidad de acceso tiene dos objetivos principales; ser la entrada del usuario al sistema y administrar el funcionamiento de todos los componentes de la arquitectura. La entidad se compone de dos procesos; el proceso *REST*, que atiende las peticiones de los usuarios y el proceso de control del acceso, el cual administra a la instancia de la entidad. A continuación se explican las características de los módulos que conforman la entidad de acceso; algunos módulos se encuentran en ambos procesos, en este caso, se explica el funcionamiento en cada proceso.

Módulo de administración *REST*

Módulo que se encuentra en el proceso *REST*, es el encargado de iniciar y administrar el servidor *REST*, el cual se basa en la biblioteca Civetweb[1]; el módulo configura el servicio *REST* en función de los parámetros que recibe al inicio de la ejecución desde la línea de comandos. También es el encargado de procesar las peticiones *REST* y codificar los mensajes adecuados con la información de las solicitudes, para transferirlas al proceso de control del acceso. Es necesario que el módulo pueda realizar varias tareas al mismo tiempo, por tal razón, utiliza la biblioteca Boost [17] utilizando la clase Threadspool [2].

Módulo de comunicación

Módulo que se encuentra presente en ambos procesos de la entidad de acceso, su misión es permitir la comunicación entre ellos. Utiliza el *Middleware* ZeroMQ [34] para establecer el canal de comunicación, el cual se establece mediante un *Socket* bidireccional, en el cual solo un mensaje se puede transferir a la vez y puede transmitir información desde cualquiera de ambos procesos.

El módulo se encuentra en ambos procesos y tiene funciones específicas en cada uno.

- **Proceso REST:** el módulo en este proceso se encarga de obtener un puerto

[1] `http://sourceforge.net/projects/civetweb/`
[2] `http://threadpool.sourceforge.net/`

disponible en la máquina en que se ejecuta el nodo de acceso, después inicia el proceso del control de acceso indicándole el puerto con el cual se comunicaran ambos procesos y se conecta al Socket del nuevo proceso. El módulo verifica constantemente que el proceso del control de acceso responda a las peticiones, si no es así, solicita al sistema operativo anfitrión la terminación de dicho proceso e inicia uno nuevo; al terminar dicho proceso las otras instancias de acceso en el sistema detectan la falla y recuperan la información del proceso. El módulo administra el canal de comunicación entre los procesos, el cual sólo puede transferir un mensaje a la vez y el proceso puede recibir múltiples peticiones *REST* a la vez; por tal razón, se encarga de controlar el acceso al canal de comunicación.

- **Proceso de control del acceso:** el módulo en este proceso inicia el *Socket* del *Middleware ZeroMQ* con el número de puerto decidido por el proceso *REST*; a través del *Socket* el módulo recibe los mensajes del proceso *REST*, los interpreta y se realiza las acciones pertinentes.

Módulo de control y administración

Módulo que se encuentra en el proceso de control del acceso, se encarga de controlar todas operaciones del funcionamiento de la arquitectura distribuida que se propone; recibe del proceso *REST* los mensajes de las peticiones de los usuarios y realiza las operaciones necesarias. El módulo contiene el objeto distribuido *CORBA*, mediante el cual las otras instancias en la arquitectura distribuida se comunican con la instancia, cada objeto *CORBA* en el sistema distribuido tiene un identificador único. El módulo tiene varios submódulos que se encargan de administrar, respaldar y recuperar los archivos de reglas, *scripts* y hechos, con los cuales trabajan los sistemas expertos; el respaldo de la información se realiza mediante un sistema gestor de bases de datos.

Módulo de descubrimiento de nodos

Módulo en el proceso de control del acceso se encarga de descubrir los objetos *CORBA* de las nuevas instancias en el sistema distribuido; utiliza el servicio de nombres para obtener la lista de objetos *CORBA* registrados en él, verifica que dichos objetos

respondan a peticiones remotas.

Módulo contenedor de nodos

Módulo que se encuentra en el proceso de control del acceso, tiene como objetivo administrar las referencias a los objetos *CORBA* y verificar que se encuentren en funcionamiento. Almacena toda la información de las otras instancias en el sistema distribuido; esto le permite saber la cantidad de archivos de reglas que existen en las instancias del sistema experto, lo cual le brinda la capacidad de decidir en qué instancia asignar los nuevos archivos de reglas, para lo cual utiliza la política de asignación que se encuentra seleccionada. Las políticas de asignación que se pueden utilizar son las siguientes:

- **Cantidad de archivos de reglas en los nodos expertos:** el módulo tiene el registro de la cantidad de archivos de reglas que tiene cada nodo experto en el sistema distribuido, entonces cuando llega un nuevo archivo de reglas se elige el nodo con menor cantidad de archivos de reglas.
- **Cantidad de reglas en los nodos expertos:** el módulo tiene el registro de la cantidad de reglas que tiene cada nodo experto en la sistema distribuido, cuando llega un nuevo archivo de reglas se elige el nodo con menor cantidad de archivos de reglas.
- **Round-Robin:** el usuario asigna una lista de los nodos expertos que desea que se utilicen con esta política, todas las instancias de acceso apuntan a la misma posición en la lista. Cada vez que llega un nuevo archivo de reglas se elige el nodo experto que se encuentra en la posición seleccionada de la lista y se les pregunta a todos las instancias de acceso si están de acuerdo con la elección, en caso de que alguno no esté de acuerdo, se propone una nueva posición en la lista y se repite el proceso; al final se mueven todos a la siguiente posición de la seleccionada y aceptada.
- **Elección del usuario:** el usuario puede elegir el nodo experto en el cual se asignará el archivo de reglas; se utiliza el identificador único del nodo, el cual es su nombre.

Módulo de distribución *Multicast*

Módulo cuya función es la distribución de los archivos de hechos desde las instancias de la entidad de acceso, también llamadas nodos de acceso, a todos los nodos expertos. Utiliza un protocolo *Multicast* [33] para enviar la información, el cual permite que con sólo una petición de envío, la información se transfiera a múltiples destinatarios; evitando el congestionamiento de la red y reduciendo el tiempo necesario para transferir la información a todos los nodos expertos. El diseño del módulo utiliza el *Middleware* de comunicación RabbitMQ[36], se utiliza un Clúster *RabbitMQ* para asegurar que siempre exista un servidor para el envío de la información. El módulo se encarga de gestionar la conexión desde el nodo de acceso a todos los nodos expertos, encapsulando todo el funcionamiento de la transferencia del archivo, lo cual lo hace en conjunto con el módulo de distribución *Multicast* de los nodos expertos. Esto permite que los otros módulos soliciten transferir un archivo, dejando que el módulo se encargue de todos el proceso. Las características del módulo se discuten con mayor detalle en el capítulo 6.

4.3.2. Módulos de la entidad de sistema experto

El diseño de la arquitectura distribuida propuesto contiene la entidad del sistema experto, la cual tiene como objetivo administrar y gestionar los procesos expertos, en los cuales se lleva a cabo el proceso de inferencia con los archivos de reglas y hechos que recibe el sistema distribuida. La entidad se compone de dos tipos procesos; el proceso de control del sistema experto y el proceso del sistema experto. A continuación se explican las características de los módulos que conforman la entidad; algunos módulos se encuentran en ambos procesos, en este caso se explica el funcionamiento en cada proceso.

Módulo de control y administración

Módulo que se encuentra en el proceso de control del sistema experto; se encarga de gestionar los archivos de reglas, hechos y *scripts*, que utilizan los procesos del sistema experto que administra. El módulo contiene el objeto distribuido *CORBA*, mediante el cual las otras instancias del sistema distribuido se comunican con la instancia.

Módulo de comunicación

El módulo se encuentra presente en ambos procesos de la entidad del sistema experto, su misión es permitir la comunicación entre ellos. Utiliza el *Middleware ZeroMQ* [34] para establecer el canal de comunicación. El canal se establece mediante un *Socket* bidireccional, en el cual solo un mensaje se puede transferir a la vez y se puede transmitir información desde cualquiera de ambos procesos. El módulo se explica con mayor detalle en el capítulo 7.

El módulo se encuentra en ambos procesos y tiene funciones específicas en cada uno.

- **Proceso de control del nodo experto:** el módulo en este proceso se encarga de obtener un puerto disponible en la máquina en que se ejecuta el nodo experto, después inicia el proceso del sistema experto indicándole el puerto con el cual se comunicaran ambos procesos y se conecta al *Socket ZeroMQ* del nuevo proceso; este proceso se realiza por cada archivo de reglas que recibe la instancia. Utiliza un submódulo para la gestión de la comunicación con el proceso del sistema experto, agregando una instancia del submódulo por cada proceso del sistema experto asociado. El submódulo verifica constantemente que el proceso del sistema experto responda a las peticiones, si no es así, solicita al sistema operativo anfitrión la terminación de dicho proceso e inicia uno nuevo; de manera automática carga el archivo de reglas que tenía asociado el proceso anterior.

- **Proceso de sistema experto:** módulo que inicia el *Socket* del *Middleware ZeroMQ* con el número de puerto decidido por el proceso de control del sistema experto; a través del *Socket* el módulo recibe los mensajes del proceso de control, los interpreta y transmite las peticiones al *Framework* del sistema experto con el que trabaja. El módulo utiliza el *Middleware ZeroMQ*, pues tiene la ventaja de poder ser utilizado en más de 40 lenguajes de programación; ampliando la variedad de sistemas expertos que se pueden utilizar con la arquitectura distribuida propuesta.

Módulo de distribución *Multicast*

Módulo cuya función es la distribución de los archivos de hechos desde una instancia de la entidad de acceso, también llamada nodo de acceso, a todos los nodos expertos. Utiliza un protocolo *Multicast* [33] para enviar la información, el cual permite que con sólo una petición de envío, la información se transfiera a múltiples destinatarios; evitando el congestionamiento de la red y reduciendo el tiempo necesario para transferir la información a todos los nodos expertos. El diseño del módulo utiliza el *Middleware* de comunicación *RabbitMQ*, se utiliza un Clúster *RabbitMQ* para asegurar que siempre exista un servidor para el envío de la información. El módulo trabaja en conjunto con el módulo de distribución *Multicast* del nodo de acceso emisor, con el fin de recibir los archivos a transferir; se encarga de todo el proceso y en caso de existir alguna falla en la transmisión, se encarga de recuperar la información que no haya llegado. Notifica al módulo de control y administración cuando el archivo se ha recibido por completo. El funcionamiento del módulo se aborda con mayor detalle en el capítulo 6.

Framework del sistema experto

El *Framework* contiene el núcleo del sistema experto que es utilizado en cada uno de los procesos expertos, da acceso a los procedimientos y funciones de inferencia; varía dependiendo del sistema experto elegido para la implementación del programa a utilizar con la arquitectura que se propone. El usuario puede utilizar distintos sistemas expertos con la arquitectura propuesta, pero hay varios requisitos que debe cumplir el programa con el sistema que se desea utilizar, existen dos casos de uso. El primer caso es cuando se utiliza un sistema experto distribuido, debe cumplir los siguientes requisitos: requiere distribuir la memoria de la base de conocimiento pero sin distribuir la memoria en donde se almacenan las reglas, pueda ejecutar varios procesos del sistema elegido en una misma computadora y se base en archivos de reglas[1]. El segundo caso es cuando se quiere utilizar sistema experto sin un proceso de inferencia distribuido, debe cumplir los siguientes requisitos: pueda ejecutar varios procesos del sistema elegido en una misma computadora y se base en archivos de reglas; es importante notar que todos los procesos expertos recibirán los hechos para su procesamiento pero los resultados no se comparten entre ellos.

4.4. Fortalezas

El uso del patrón de diseño modular en el diseño propuesto permite que la complejidad del sistema se reduzca e incremente en cierto grado la facilidad para realizar cambios en el diseño de los componentes de la arquitectura distribuida que se propone. Es posible realizar cambios en el funcionamiento de alguno de los módulos, sin afectar el funcionamiento de otros [39].

La utilización de un servidor *REST* para recibir las peticiones de los usuarios permite que no importe la plataforma que utilicen, el sistema es capaz de atenderlos; también es capaz de atender a múltiples usuarios a la vez. El diseño propuesto establece un conjunto de páginas web que el servidor *REST* despliega cuando el usuario utiliza un navegador Web, esto con el objetivo que el usuario pueda interactuar con el sistema a través de una interfaz gráfica; en caso contrario necesitaría un programa que conozca los comando *REST* que reconocen los nodos de acceso.

El módulo de control y administración del proceso de control de acceso que se encuentra en la instancia de acceso, posee un submódulo encargado de conectarse con un sistema gestor de bases de datos; aquí es en donde se respalda y recupera la información del funcionamiento de la arquitectura. Se utiliza la biblioteca Libzdb[3], la cual permite utilizar los siguientes sistemas gestores de bases de datos: Postgresql [4], Oracle[5] y MySQL[6]; para elegir cualquiera de ellos, solo es necesario modificar la cadena de conexión. El submódulo encapsula el proceso completo de respaldo y recuperación; e. g., si es necesario cambiar la biblioteca a utilizar, el cambio no afectaría a los otros módulos si la interfaz no cambia.

La plataforma de distribución de archivos utiliza el protocolo *Multicast*, el cual permite transferir información de una manera rápida y sin congestionar la red, esto se debe a la capacidad del protocolo para enviar un mensaje a varios destinarios a la vez. Se utiliza el *Middleware* RabbitMQ por su capacidad de utilizar un *Cluster* para el proceso de envío de mensajes, de esta forma no solo es un servidor el que se encarga de la transferencia de la información.

El uso del *Middleware ZeroMQ* de comunicación en los módulos que permiten

[3] `http://www.tildeslash.com/libzdb/`
[4] `http://www.postgresql.org/`
[5] `http://www.oracle.com/index.html`
[6] `http://www.mysql.com/`

la comunicación entre los procesos, tiene como objetivo permitir que se pueda utilizar una amplia variedad de sistemas expertos con la arquitectura distribuida; esto es posible gracias a la posibilidad de utilizar *ZeroMQ* en más de 40 lenguajes de programación.

4.5. Debilidades

Cada módulo de la arquitectura distribuida que se propone está diseñado para atender una gran cantidad de peticiones, para esto se utiliza el patrón *Threads Pool* [35]; en el cual al inicio de un proceso, se crean todos los *Threads*. El patrón genera una gran carga en el sistema operativo anfitrión y limita la cantidad de instancias que se pueden ejecutar en él. Los nodos a utilizar en la arquitectura distribuida deben tener *hardware* de altas especificaciones, también es posible realizar modificaciones al sistema operativo anfitrión para que no limite la cantidad de *Threads*.

En el diseño que se propone las instancias de acceso respaldan la información del funcionamiento en una base de datos, utilizando un sistema gestor de bases de datos; el problema radica cuando se tiene un servidor centralizado, el cual tiene una cantidad limitada de conexiones. La cantidad de nodos de acceso se limita a la cantidad de conexiones que el sistema gestor de bases de datos, esto limita la escalabilidad del diseño que se propone.

4.6. Resumen

La arquitectura distribuida que se propone está diseñada para agregar varias características de un sistema distribuido a un sistema experto; las características que se agregan son: Escalabilidad, tolerancia a fallas y alta disponibilidad. Para agregar las características deseadas se utiliza el patrón divide y vencerás; el diseño de la arquitectura divide y reparte todos los servicios entre todas las instancias de las entidades que componen la arquitectura distribuida.

Los programas de los procesos que conforman la arquitectura distribuida propuesta están diseñados utilizando el enfoque de programación modular; cada módulo tiene un objetivo específico, cuando necesita realizar alguna tarea con un enfoque distinto

al del módulo, se basa en los servicios de otros módulos para alcanzar sus objetivos. Los módulos deben ser lo más independientes posible entre sí, pues si es necesario modificar el funcionamiento de alguno, no debe ocasionar fallas en los módulos que lo utilizan; esta estrategia de diseño permite que los sistemas se mejoren sin afectar su funcionamiento actual, pues sólo se necesita reemplazar los módulos a mejorar. El diseño de la arquitectura distribuida que se propone en el presente trabajo de tesis utiliza un enfoque modular, agrupando todos los servicios similares en módulos; el patrón permite realizar cambios en un módulo en particular sin afectar el funcionamiento de los otros módulos.

En la arquitectura distribuida que se propone, la entidad de acceso tiene dos objetivos principales; ser la entrada del usuario al sistema y administrar el funcionamiento de todos los componentes de la arquitectura. La entidad se compone de dos procesos; el proceso *REST* que atiende las peticiones de los usuarios y el proceso de control del funcionamiento de la entidad. También existe la entidad de sistema experto, la cual tiene como objetivo administrar y gestionar los procesos del sistema experto, en los cuales se lleva a cabo el proceso de inferencia con los archivos de reglas y hechos que recibe el sistema. La entidad se compone de dos procesos; el proceso de control del sistema experto y el proceso del sistema experto.

El diseño que se propone tiene como objetivo distribuir la carga de trabajo. Para distribuir la carga que pueden representar los archivos de reglas los nodos de acceso pueden utilizar 4 políticas para distribuir los archivos de reglas entre los nodos expertos: *Round-Robin*, cantidad de archivos de reglas en los nodos expertos, cantidad de reglas en los nodos expertos y la elección del usuario; la política es seleccionado por el administrador del sistema.

El diseño de la arquitectura distribuida que tiene las siguientes fortalezas: el uso del patrón de diseño modular, la recepción de peticiones **REST** de los usuarios, el uso de varios sistemas gestores de bases de datos mediante una cadena de conexión, la transferencia de archivos de reglas mediante el protocolo *Multicast* y la capacidad de utilizar múltiples sistemas expertos con la arquitectura. El diseño que se presenta también tiene algunas debilidades: cantidad limitada de instancias que se pueden ejecutar en un sistema operativo anfitrión y la cantidad de nodos de acceso que puede aceptar el sistema gestor de bases de datos que se utiliza.

Capítulo 5

Servidor REST

En este capítulo se explica la estrategia que se utiliza para el diseño del módulo del servidor *REST* y se describen las características de funcionamiento del módulo. En el capítulo también se especifican los comandos que entiende el servidor *REST* y las respuestas dependiendo del comando recibido.

5.1. Enfoque

La entidad de acceso es la entrada de los usuarios al sistema distribuido. Requiere utilizar un mecanismo de comunicación que sea ampliamente utilizado y que no dependa de alguna plataforma; esto tiene como finalidad, permitir que los usuarios puedan utilizar los servicios del sistema distribuido, sin importar el sistema operativo que utilicen.

Cada instancia de la entidad de acceso puede recibir múltiples peticiones de varios usuarios al mismo tiempo y todas se deben atender; esto ocasiona la necesidad del procesamiento concurrente de todas las peticiones, las cuales generan una respuesta que se debe regresar al usuario solicitante.

5.2. Diseño

Dentro de la entidad de acceso en el proceso *REST*, se encuentra el servidor REST, el cual tiene como objetivo recibir mensajes codificados en *HTTP* a través de peti-

ciones *REST* [14], mediante las cuales los usuarios se comunican con la arquitectura distribuida que se propone. Utilizar mensajes *HTTP* y peticiones *REST* como mecanismo de comunicación permite que los usuarios accedan a los servicios del sistema distribuido sin importar la plataforma o sistema operativo que utilicen; esta es la principal razón por la que se decidió utilizar el mecanismo de comunicación.

El servidor *REST* es capaz de atender múltiples peticiones de los usuarios a la vez, esto es posible gracias al uso de la biblioteca *Civetweb* [1]; las cuales son redirigidas al proceso de control del acceso, mediante el canal de comunicación que utiliza el *Middleware ZeroMQ*.

El proceso *REST* inicia los servicios del servidor *REST* utilizando un número de puerto específico, el cual se recibe como parámetro cuando el administrador del sistema distribuido inicia la instancia, por esta razón es importante que el puerto se encuentre libre cuando inicia el proceso.

El servidor *REST* es capaz de recibir archivos codificados en los mensajes *HTTP*, para esto se utiliza la codificación especial *multipart/form-data* [40]; pero también codifica todos los parámetros adicionales, por esta razón se utiliza un método que obtiene los parámetros desde los datos codificados. Los parámetros adicionales se utilizan cuando se registra un archivo de reglas y el usuario especifica el nombre del nodo experto en donde se asignará el archivo, este comando incluye un archivo y un parámetro adicional.

El servidor *REST* atiende cada petición de los usuarios en un *thread*, lo que permite atender varias peticiones a la vez; esto es posible por el uso de la biblioteca *Civetweb*. En la figura 5.1 se puede observar a cinco usuarios que solicitan un servicio a un nodo de acceso, las cuales son procesadas por *threads* generados por *Civetweb*.

[1] `http://sourceforge.net/projects/civetweb/`

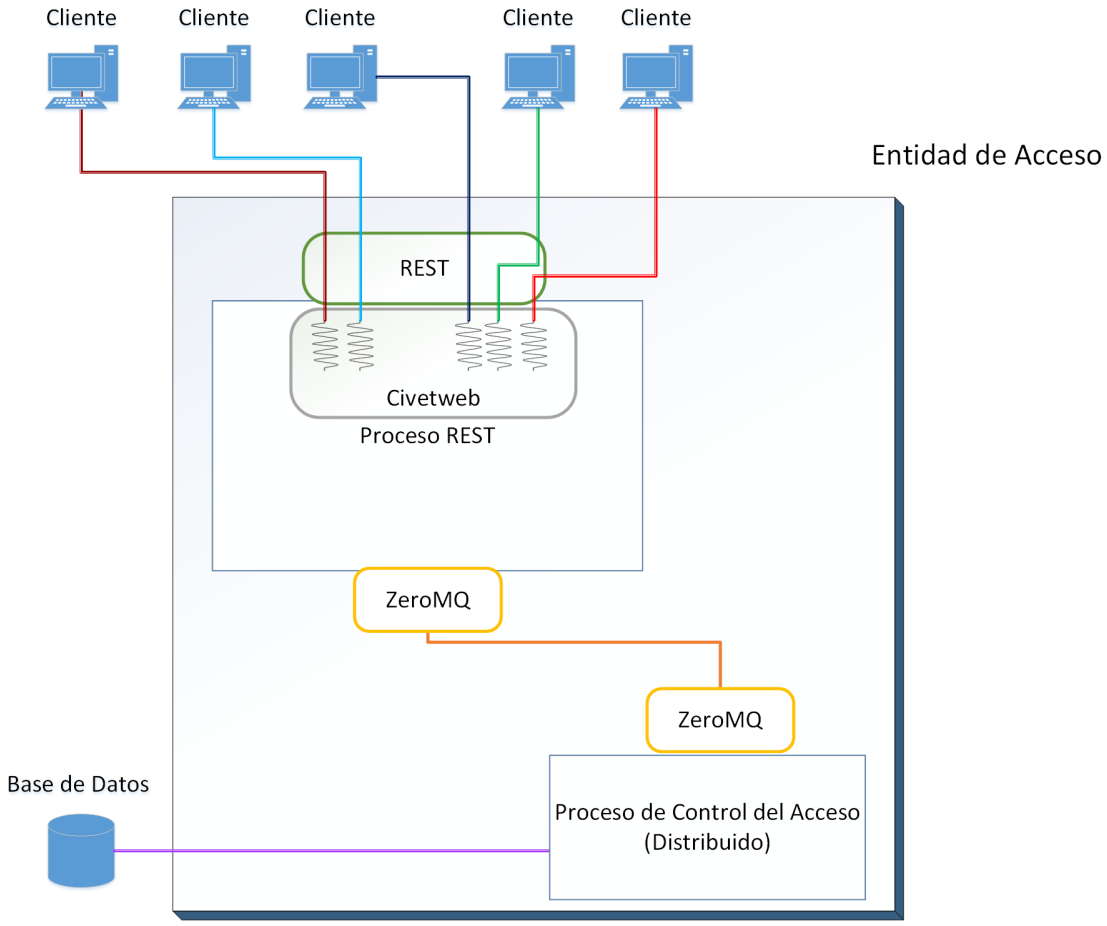

Figura 5.1: Servidor *REST*.

5.3. Comandos y respuestas

A continuación se describen los comandos que entiende el servidor *REST*, también se especifican los parámetros que necesitan y la respuestas que se pueden regresar por cada comando. El servidor *REST* es capaz de desplegar páginas *web* cuando se utiliza un navegador web para utilizar los servicios del sistema distribuido, existen comandos que regresan los resultados mediante páginas *web*; estos comandos no son explicados, pues existen los mismos comandos sin una interfaz gráfica *web*.

Comando y Tipo de Petición	**available** (GET)
Descripción	Comando con el cual el proceso *REST* verifica que puede comunicarse con el proceso de control del acceso al cual está asociado. Este comando es importante, pues si no tiene comunicación, no puede redirigir las peticiones de los usuarios al sistema distribuido; el usuario antes de enviar una petición, necesita comprobar que el proceso *REST* del nodo acceso se puede comunicar con el sistema distribuido.
Parámetros	
sin parámetros	
Respuestas	
Respuesta	Descripción
online	El proceso *REST* puede comunicarse con el sistema distribuido.
offline	El proceso *REST* no puede comunicarse con el sistema distribuido.

Tabla 5.1: Especificación del comando **available**.

Comando y Tipo de Petición	**definition_upload** (POST)
Descripción	Comando del proceso *REST* que permite a los usuarios registrar un archivo de definición de un archivo de reglas, en el cual se definen los archivos *scripts* que utilizan las reglas del archivo, con esta información se puede transferir los archivos *scripts* en el mismo procedimiento que transfiere el archivo de reglas a un nodo experto. Sólo se puede recibir un archivo por petición.
Parámetros	
Nombre	Descripción
file	Parámetro del tipo *file* que es transmitido utilizando la codificación *multipart/form-data*.
Respuestas	
Respuesta	Descripción
ok	El proceso *REST* trabajó la petición del registro del archivo de definición correctamente.
error	El proceso *REST* tuvo problemas para trabajar la petición del registro del archivo de definición.
no-file-error	El proceso *REST* no recibió los datos del archivo de definición en la petición.
offline	El proceso *REST* no puede comunicarse con el sistema distribuido.

Tabla 5.2: Especificación del comando **definition_upload**.

Comando y Tipo de Petición	**script_upload** (POST)
Descripción	Comando del proceso *REST* que permite a los usuarios registrar o actualizar un archivo *script* en el sistema distribuido. Sólo se puede recibir un archivo por petición.
Parámetros	
Nombre	Descripción
file	Parámetro del tipo *file* que es transmitido utilizando la codificación *multipart/form-data*.
Respuestas	
Respuesta	Descripción
ok	El proceso *REST* trabajó la petición del registro del archivo *script* correctamente.
error	El proceso *REST* tuvo problemas para trabajar la petición del registro del archivo *script*.
no-file-error	El proceso *REST* no recibió los datos del archivo *script* en la petición.
offline	El proceso *REST* no puede comunicarse con el sistema distribuido.

Tabla 5.3: Especificación del comando **script_upload**.

Comando y Tipo de Petición	**expert_node_allocation_policies** (GET)
Descripción	Comando al cual el proceso *REST* regresa la política de asignación de archivos de reglas, configurada en el sistema distribuido.
Parámetros	
sin parámetros	
Respuestas	
Respuesta	Descripción
per-rules-files-count-allocation-policy	La política de asignación configurada en el sistema distribuido es aquella que está basada en la cantidad de archivos de reglas que se han asignado en los nodos expertos.
per-rules-count-allocation-policy	La política de asignación configurada en el sistema distribuido es aquella que está basada en la cantidad de reglas que se han asignado en los nodos expertos.
round-robin-list-allocation-policy	La política de asignación configurada en el sistema distribuido es aquella que está basada en la lista de nodos expertos que se itera mediante el algoritmo Round-Robin.
error	El proceso *REST* tuvo problemas para trabajar la solicitud de la política de asignación configurada.
offline	El proceso *REST* no puede comunicarse con el sistema distribuido.

Tabla 5.4: Especificación del comando **expert_node_allocation_policies**.

Comando y Tipo de Petición	**round_robin_policy_list_upload** (POST)
Descripción	Comando del proceso *REST* que permite a los usuarios registrar o actualizar la lista de nodos expertos que se utiliza para la política de asignación Round-Robin.
Parámetros	
Nombre	Descripción
file	Parámetro del tipo *file* que es transmitido utilizando la codificación *multipart/form-data*.
Respuestas	
Respuesta	Descripción
ok	El proceso *REST* trabajó la petición del registro del archivo de la lista de nodos expertos.
error	El proceso *REST* tuvo problemas para trabajar la petición del registro del archivo de la lista de nodos expertos.
no-file-error	El proceso *REST* no recibió los datos del archivo de la lista de nodos expertos.
offline	El proceso *REST* no puede comunicarse con el sistema distribuido.

Tabla 5.5: Especificación del comando **round_robin_policy_list_upload**.

Comando y Tipo de Petición	**change_current_allocation_policy** (POST)
Descripción	Comando del proceso *REST* que permite a los usuarios seleccionar una de las tres políticas de asignación que tiene el sistema distribuido: por cantidad de archivos de reglas en los nodos expertos, por cantidad de reglas en los nodos expertos y una lista de nodos expertos que se itera con un algoritmo Round-Robin.
Parámetros	
Nombre	Descripción
current_policy	Parámetro del tipo *text* que indica la política de asignación de los archivos de reglas en los nodos expertos, con la cual trabajará el sistema distribuido; las constantes del parámetro y las políticas son las siguientes: 1 - cantidad de archivos de reglas 2 - cantidad de reglas 3 - Round-Robin
Respuestas	
Respuesta	Descripción
ok	El proceso *REST* trabajó la petición de la selección de la política de asignación.
policy-error	No se recibió la selección de la política de asignación.
error	El proceso *REST* tuvo problemas para seleccionar de la política de asignación.
policy-not-found	La política de asignación seleccionada no es válida.
offline	El proceso *REST* no puede comunicarse con el sistema distribuido.

Tabla 5.6: Especificación del comando **change_current_allocation_policy**.

Comando y Tipo de Petición	**rules_upload** (POST)
Descripción	Comando del proceso *REST* que permite a los usuarios registrar un archivo de reglas en el sistema distribuido; se puede recibir el identificador de un nodo experto, en el cual se asignará el archivo de reglas. Sólo se puede recibir un archivo por petición.
Parámetros	
Nombre	Descripción
file	Parámetro del tipo *file* que es transmitido utilizando la codificación *multipart/form-data*.
expert_node_name	Parámetro opcional del tipo *text*, en el cual se recibe el identificador de un nodo experto, en el cual se asignará el archivo de reglas ignorando la política de asignación en funcionamiento. Si no se recibe el parámetro, el nodo experto se elige en función de la política de asignación en funcionamiento.
Respuestas	
Respuesta	Descripción
ok	El proceso *REST* trabajó la petición del registro del archivo de reglas correctamente.
error	El proceso *REST* tuvo problemas para trabajar la petición del registro del archivo de reglas.
no-file-error	El proceso *REST* no recibió los datos del archivo de reglas en la petición.
unknown-node	No existe en el sistema distribuido el nodo experto con el identificador recibido en la petición.
offline	El proceso *REST* no puede comunicarse con el sistema distribuido.

Tabla 5.7: Especificación del comando **rules_upload**.

Comando y Tipo de Petición	**facts_upload** (POST)
Descripción	Comando del proceso *REST* que permite a los usuarios registrar un archivo de hechos en el sistema distribuido.
Parámetros	
Nombre	Descripción
file	Parámetro del tipo *file* que es transmitido utilizando la codificación *multipart/form-data*.
Respuestas	
Respuesta	Descripción
ok	El proceso *REST* trabajó la petición del registro del archivo de hechos correctamente.
error	El proceso *REST* tuvo problemas para trabajar la petición del registro del archivo de hechos.
no-file-error	El proceso *REST* no recibió los datos del archivo de hechos en la petición.
offline-distribution-platform	El proceso *REST* no tiene acceso a la plataforma de distribución de archivos.
offline	El proceso *REST* no puede comunicarse con el sistema distribuido.

Tabla 5.8: Especificación del comando **facts_upload**.

Comando y Tipo de Petición	**retrieve_expert_nodes_names** (GET)
Descripción	Comando con el cual el proceso *REST* retorna una lista de los identificadores de los nodos expertos en el sistema distribuido, los identificadores son separados por una coma ‘,’.
Parámetros	
sin parámetros	
Respuestas	
Respuesta	Descripción
lista de identificadores	Lista de los identificadores de los nodos expertos en el sistema distribuido separados por comas.
offline	El proceso *REST* no puede comunicarse con el sistema distribuido.

Tabla 5.9: Especificación del comando **retrieve_expert_nodes_names**.

5.4. Fortalezas

Utilizar peticiones *REST* y mensajes codificados en *HTTP*, los cuales son los mecanismos que hacen funcionar a los sitios web de Internet, permite que los servicios del sistema distribuido puedan ser utilizados por los usuarios sin importar la plataforma o sistema operativo que utilicen. Así como los servidores web pueden atender a va-

rios usuarios a la vez, el servidor *REST* de cada nodo de acceso es capaz de atender múltiples peticiones al mismo tiempo.

Los nodos de acceso tienen la capacidad de atender las peticiones de los usuarios a través de un navegador web, los nodos de acceso despliegan páginas web para que los usuarios puedan utilizar y administrar los servicios del sistema distribuido.

5.5. Deficiencias

El servidor *REST* de cada nodo de acceso sólo puede atender las peticiones de los usuarios a través de un puerto específico, es necesario que los usuarios conozcan la dirección *IP* y el puerto del servidor *REST*. Es necesario definir un puerto para que todos los nodos de acceso sean inicializados de una manera homogénea y de esta forma evitar problemas a los usuarios, pues sólo es necesario que conozcan un solo un puerto de funcionamiento.

Cuando el servidor *REST* de un nodo acceso no puede atender las peticiones de un usuario, es necesario que el usuario vuelva a realizar la solicitud del servicio a otro nodo de acceso, por esto es necesario que los usuarios conozcan las direcciones *IP* de varios nodos de acceso.

5.6. Resumen

El proceso *REST* se ubica en la entidad de acceso y tiene como objetivo recibir mensajes codificados en *HTTP* a través de peticiones *REST*, mediante el cual los usuarios acceden a los servicios del sistema distribuido sin importar la plataforma o sistema operativo que utilicen. Los servicios del servidor *REST* trabajan en un número de puerto específico, el cual se recibe como parámetro cuando el administrador del sistema distribuido inicia la instancia, por esta razón es importante que el puerto se encuentre libre. El servidor *REST* es capaz de recibir archivos codificados en los mensajes *HTTP*, para esto se utiliza la codificación especial *multipart/form-data*.

El servidor *REST* atiende cada petición de los usuarios en un *Thread*, lo que permite atender varias peticiones a la vez; esto es posible por el uso de la biblioteca *Civetweb*.

El servidor *REST* entiende varios comandos y puede regresar distintas respuestas dependiendo del comando que recibe. Los comandos permiten a los usuarios controlar y utilizar los servicios del sistema distribuido.

Capítulo 6

Plataforma para la distribución de archivos

En este capítulo se explica el funcionamiento de la plataforma de distribución de archivos, la cual tiene como propósito transferir los archivos de hechos a todos los nodos expertos del sistema distribuido, de una manera rápida y sin saturar la red de comunicación. La plataforma para la distribución de archivos incluye en su diseño un protocolo *Multicast* [33] y el patrón emisor/subscritor [41], mediante los cuales un mensaje con datos puede ser enviado a todos los nodos expertos, evitando saturar los recursos de la red de comunicación.

6.1. Enfoque

El objetivo principal de la plataforma para la distribución de archivos es la transferencia de los archivos de hechos, los cuales se reciben en los nodos de acceso y se deben enviar a todos los nodos expertos del sistema distribuido; la transferencia debe ser lo más rápido posible y evitando congestionar la red de comunicación.

La plataforma para la distribución de archivos permite que cualquier nodo de acceso transfiera los archivos de hechos que recibe, a todos los nodos expertos; esto ocasiona la necesidad de transmitir una gran cantidad de información. La plataforma de distribución permite elegir la cantidad de canales de comunicación con los que trabajará, esto se realiza cuando se inicia cada uno de los nodos de acceso y es

necesario que la configuración sea la misma en todos los nodos de acceso.

La plataforma para la distribución de archivos requiere conectar todos los nodos de acceso hacia todos los nodos expertos, mediante un mecanismo de un sólo sentido de transferencia; además es necesario que los servicios de la plataforma de distribución no sean centralizados, pues esto limitaría la escalabilidad de la arquitectura distribuida.

6.2. Diseño

La plataforma para la distribución de archivos utiliza el *Middleware* de comunicación *RabbitMQ*[36], el cual mediante el patrón emisor/subscriptor, permite enviar un mensaje a múltiples destinatarios, evitando congestionar la red de comunicación y permiten realizar la transferencia a todos los destinatarios en un menor tiempo.

El diseño de la plataforma de distribución contempla el uso de un módulo en el proceso de control del acceso en los nodos de acceso y un módulo en el proceso de control del sistema experto en los nodos expertos, mediante los cuales los nodos se conectan a un clúster *RabbitMQ*. Cuando se inician los nodos de acceso, dentro de los parámetros de inicio, se especifica la cantidad de canales con los que trabajará la plataforma de distribución; la configuración de la cantidad de canales debe ser la misma en todos los nodos de acceso y los nodos expertos.

Los nodos de acceso son los encargados de registrar los canales de la plataforma para la distribución de archivos, los cuales se registran en el clúster *RabbitMQ*; por esta razón es necesario que antes de iniciar algún nodo experto, se inicie un nodo de acceso, una vez iniciados los canales, los siguientes nodos de acceso sólo se registran como emisores.

6.2.1. Arquitectura

El diseño de la arquitectura distribuida que se propone incluye una plataforma para la distribución de archivos, la cual utiliza un clúster *RabbitMQ* para los servicios de transferencia de archivos a todos los nodos expertos. Los nodos de acceso crean los canales de comunicación en los servicios de *RabbitMQ* y se registran como emisores; los nodos expertos se conectan al clúster y registran como subscriptores de todos los

canales. De esta forma, cada vez que un nodo de acceso envíe un mensaje a través de un canal, este se enviará a todos los nodos expertos.

Unos de los objetivos del presente trabajo de tesis es agregar la característica de la escalabilidad a un sistema experto mediante un sistema distribuido. Como se mencionó en la sección, la distribución de los archivos de hechos puede impactar negativamente a la capacidad de escalar la arquitectura distribuida que se propone; para solucionarlo se utiliza la plataforma para la distribución de archivos, pero es necesario que también pueda escalar conforme requiera el sistema distribuido. La plataforma de distribución de archivos provee sus servicios gracias a un clúster *RabbitMQ*[1], al cual se le pueden agregar más nodos conforme se requieran y los nodos del sistema distribuido pueden utilizar los servicios *RabbitMQ* desde cualquier nodo del clúster[2].

Las instancias de las entidades de la arquitectura utilizan los módulos de la plataforma para la distribución de archivos para acceder a sus servicios; los módulos utilizan la envoltura *SimpleAmqpClient*[3] para utilizar los servicios del *Middleware RabbitMQ*.

En la figura 6.1 se puede observar el diagrama de la arquitectura de la plataforma para la distribución de archivos; se muestra un sistema distribuido con 4 nodos de acceso, 3 nodos expertos y una plataforma para la distribución de archivos configurada con 2 canales de comunicación.

[1] `https://www.rabbitmq.com/clustering.html`

[2] `http://docs.openstack.org/high-availability-guide/content/_configure_rabbitmq.html`

[3] `https://github.com/alanxz/SimpleAmqpClient`

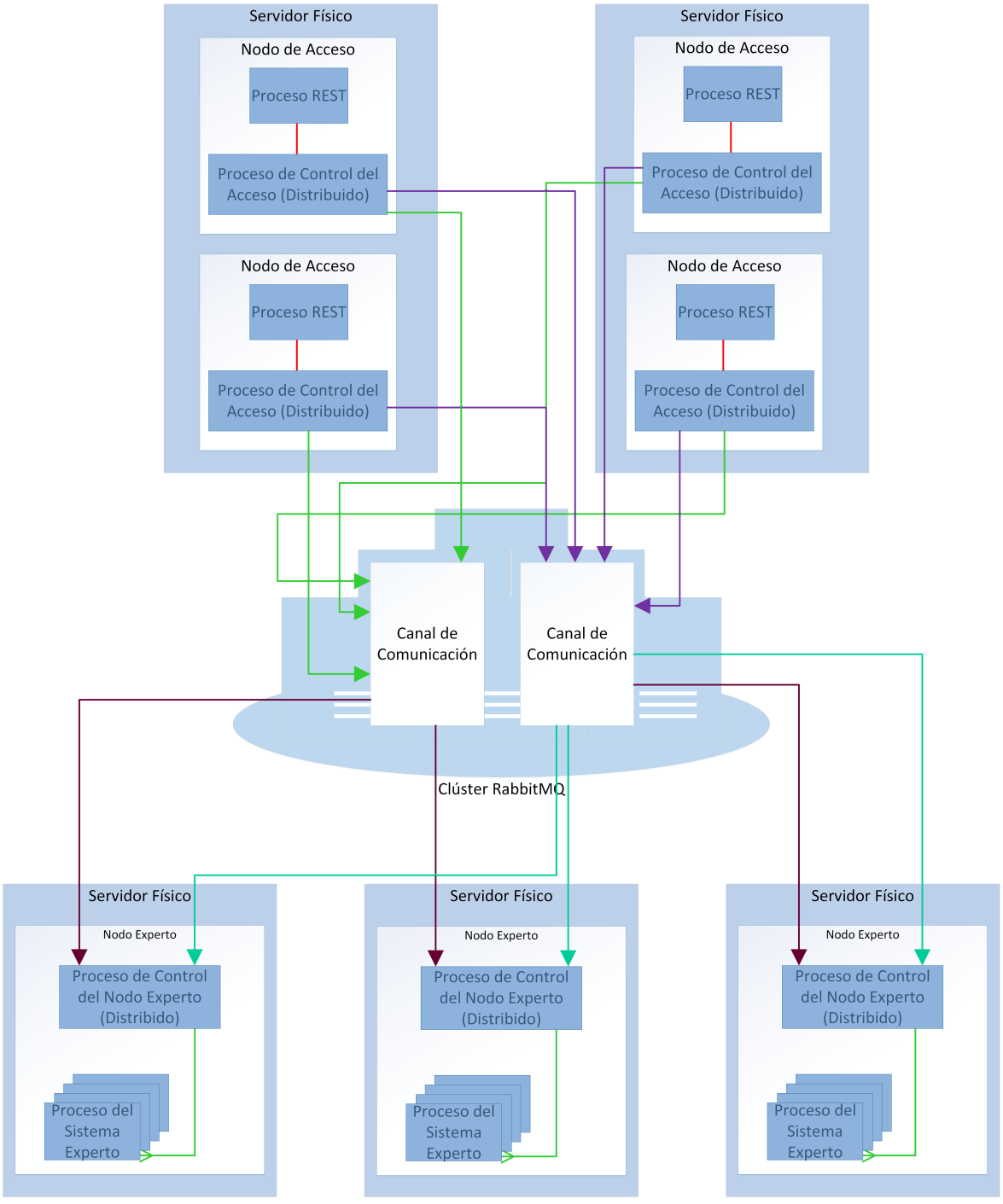

Figura 6.1: Diagrama de la arquitectura de la plataforma para la distribución de archivos.

6.2.2. Funcionamiento

La plataforma para la distribución de archivos tiene como objetivo, transferir los archivos de hechos que reciben los nodos de acceso hacia todos los nodos expertos. A continuación se describe el procedimiento que utilizan los módulos de la plataforma de distribución en ambas entidades de la arquitectura distribuida, el cual empieza una vez que el proceso de control del acceso recibe el comando de registro del archivo de hechos, por parte del proceso *REST*.

1. El módulo de la plataforma para la distribución de archivos se ubica en el proceso de control de acceso, presente en todos los nodos de acceso, se encarga de recibir las peticiones para transferir los archivos de hechos y sus rutas en el sistema local de archivos.

2. El módulo de la plataforma en el nodo de acceso, crea un nombre único para el archivo recibido, para evitar conflictos con archivos recibidos con el mismo nombre en otros nodos de acceso. Después solicita al módulo del almacenamiento persistente que respalde el archivo.

3. El módulo de la plataforma en el nodo de acceso prepara una estructura de control en la cual se almacenan, entre otros datos, el nombre único del archivo así como la cantidad de mensajes que se necesita para enviar los datos del archivo, basándose en la cantidad de datos que se pueden enviar en un mensaje, el cual está definido dentro del módulo de la plataforma.

4. El módulo de la plataforma en el nodo de acceso, envía un mensaje que indica el inicio del procedimiento de transferencia, el mensaje se compone del nombre único del archivo de hechos y la cantidad de segmentos en los que se dividen los datos binarios del archivo a transferir.

5. El módulo de la plataforma para la distribución de archivo en el proceso de control del sistema experto, recibe el mensaje que indica el inicio de la transferencia de un archivo de reglas; entonces se crea la estructura de control para recibir los segmentos de datos del archivo.

6. El módulo de la plataforma en el nodo de acceso, determina cuál es el canal de comunicación con menor cantidad de archivos en proceso de transferencia; cada canal es administrado por un submódulo. El módulo de la plataforma solicita al submódulo transferir el archivo y le provee la referencia a la estructura de control del archivo. En el caso que el canal de transmisión presente problemas, el módulo de la plataforma selecciona otro canal y continúa con la transferencia del archivo en el segmento en que falló el canal anterior; esto es posible pues en la estructura de control se especifica que segmento del archivo está en proceso de transferencia.

7. Cada segmento de un archivo es enviado a través de un mensaje, el cual incluye los siguientes datos: el nombre único del archivo, el número de segmento y los datos. Utilizar la envoltura SimpleAmqpClient para acceder a los servicios de RabbitMQ, presenta una complicación, ya que permite enviar mensajes basados en cadenas de texto; para hacer frente a esta dificultad, se utiliza la codificación base64 [37] para los datos binarios de los archivos a distribuir.

8. El módulo de la plataforma en el nodo de experto, recibe cada mensaje con un segmento de datos del archivo y lo almacena en la estructura de control, indicando el número de segmento.

9. Cuando el módulo de la plataforma en el nodo de acceso, envía el último segmento de datos, envía un mensaje de control indicando que el archivo se ha enviado por completo.

10. Cuando el módulo de la plataforma en el nodo de experto, recibe el mensaje de control que indica el término de todos los segmentos y ya posee todos los segmentos, escribe los datos del archivo en el sistema local de archivos, en orden basado en el número de segmento, pues los mensajes no siempre se reciben en el orden correcto; posteriormente se notifica al módulo de control del proceso de control del sistema experto que se ha recibido un archivo de hechos.

11. El módulo de la plataforma en el nodo de acceso, elimina la estructura de control para la transferencia del archivo y termina el procedimiento.

12. El módulo de la plataforma en el nodo de experto, mantiene la estructura de control para recibir el archivo por un tiempo, el cual está definido en el módulo. Mantener la estructura un tiempo adicional tiene como finalidad, evitar volver a recibir archivos que ya se transfirieron y por alguna falla en el nodo de acceso emisor, no se registró que se distribuyeron. Después de que transcurre el tiempo definido, la estructura de control es eliminada.

6.3. Fortalezas

La plataforma para la distribución de archivos permite enviar los archivos de hechos a todos los nodos expertos, en un menor tiempo comparado que con mecanismo que envía el archivo a cada nodo experto. Además, disminuye la congestión de la red, pues se requiere una menor cantidad de mensajes para transferir a todos los nodos expertos.

La plataforma de distribución funciona basada en el uso de un clúster *RabbitMQ*, el cual permite escalar la plataforma a las necesidades del sistema distribuido que se propone. El clúster RabbitMQ puede agregar más nodos en cualquier momento; además, permite que los nodos del sistema distribuido accedan a los servicios *RabbitMQ* desde cualquier nodo del clúster.

Los módulos de la plataforma de distribución, están preparados para afrontar la caída de los canales de comunicación; cuando el canal por el que se está transfiriendo un archivo, el módulo utiliza otro canal para continuar con la transferencia del archivo.

6.4. Deficiencias

Cuando un nodo de acceso falla, otro nodo de acceso es el encargado de recuperar los archivos que se encontraba procesando. Cuando el nodo acceso en funcionamiento, recupera los archivos de hechos que se encontraban transmitiéndose a los nodos expertos, no puede recuperar el estado de transferencia; por este motivo, necesita reiniciar los procedimientos de distribución de los archivos de hechos recuperados.

El uso de la envoltura *SimpleAmqpClient* para implementar los módulos de la

plataforma para la distribución de hechos, evita poder enviar mensajes con datos binarios de una manera directa; por esta razón, es necesario codificar los datos. Los datos binarios de los segmentos de los archivos se convierten a cadenas de texto, para esto se utiliza la codificación base64, pero este proceso puede ocasionar que el procedimiento requiera más tiempo.

6.5. Resumen

La plataforma para la distribución de archivos que se utiliza en la arquitectura distribuida, tiene como propósito transferir los archivos de hechos a todos los nodos expertos del sistema distribuido, de una manera rápida y sin saturar la red de comunicación. La plataforma envía los archivos utilizando un protocolo *Multicast*, el cual permite, que al enviar un mensaje, este llegue a cada uno de los nodos expertos. Utilizar esta estrategia permite incrementar la escalabilidad de la arquitectura distribuida que se propone, pues en caso contrario, sería un procedimiento muy lento; además, se perdería la escalabilidad, pues la distribución tomaría más tiempo conforme se agreguen más nodos expertos al sistema distribuido.

La plataforma para la distribución de archivos utiliza el *Middleware* de comunicación *RabbitMQ*, el cual mediante el patrón emisor/subscriptor, permite enviar un mensaje a múltiples destinatarios, evitando congestionar la red de comunicación y permiten realizar la transferencia a todos los destinatarios en un menor tiempo. La plataforma de distribución funciona basada en el uso de un clúster *RabbitMQ*, el cual permite escalar la plataforma a las necesidades del sistema distribuido que se propone. El clúster RabbitMQ puede agregar más nodos en cualquier momento; además, permite que los nodos del sistema distribuido accedan a los servicios *RabbitMQ* desde cualquier nodo del clúster.

Capítulo 7

Comunicación con el proceso experto

En este capítulo se explica el funcionamiento del módulo que permite comunicar el proceso de control del sistema experto, que se encuentra en cada nodo experto, con los procesos del sistema experto; los cuales se encargan de realizar la inferencia con las reglas y hechos que el usuario registra en el sistema distribuido. Cada nodo experto puede administrar uno o más procesos del sistema experto.

El diseño de la arquitectura distribuida que se propone, permite que sea posible utilizar una gran variedad de sistemas expertos con el sistema distribuido, pero es necesario que los programas a utilizar cumplan varios requisitos. En el capítulo se definen los comandos y respuestas, los cuales deben implementar los programas de los sistemas expertos que se quieran utilizar con el sistema distribuido propuesto en el presente trabajo de tesis.

7.1. Enfoque

El módulo de comunicación con los procesos del sistema experto, es una parte importante en el diseño de la arquitectura distribuida que se propone, pues es el que permite utilizar gran variedad de sistemas expertos con el sistema distribuido. El módulo debe utilizar un mecanismo de comunicación que se pueda utilizar con una amplia variedad de lenguajes de programación; además debe permitir un canal de

comunicación bidireccional entre el proceso de control del sistema experto y cada proceso del sistema experto.

El módulo de comunicación con los procesos del sistema experto, debe ser capaz de comunicar de comunicar al proceso de control del sistema experto, con todos los procesos del sistema experto que tiene bajo su control. El proceso de control requiere que el módulo de comunicación se capaz de enviar comandos hacia un proceso experto en especifico, así como a todos los procesos expertos en la instancia.

7.2. Diseño

7.2.1. Arquitectura

El módulo de comunicación con los procesos del sistema experto, debe controlar los canales de comunicación de cada proceso del sistema; el control de cada canal de comunicación se delega a un submódulo, siguiendo el patrón de diseño modular. A continuación se describe la arquitectura y el funcionamiento del submódulo que controla cada canal de comunicación.

Submódulo de comunicación con un proceso del sistema experto

El submódulo de comunicación con un proceso del sistema experto, es el encargado de gestionar el envío de mensajes a través del canal de comunicación que existe entre el proceso de control del sistema experto y un proceso del sistema experto.

La comunicación con los procesos del sistema experto requiere un mecanismo de comunicación que se pueda utilizar con una amplia variedad de lenguajes de programación, por esta razón se utiliza el *Middleware ZeroMQ* [34]. Además, posee *Sockets* que pueden establecer una comunicación bidireccional, lo que cubre las necesidades para la comunicación con los procesos del sistema experto.

Cada proceso del sistema experto en un nodo experto puede recibir varios comandos a la vez, pero el *Socket ZeroMQ* sólo permite enviar un mensaje a la vez, por este motivo es necesario controlar cómo se envían los mensajes a través del canal de comunicación; el submódulo de control del canal de comunicación debe controlar el envío de mensajes por el canal.

Los *Sockets* del *Middleware ZeroMQ* que se utilizan con el sistema distribuido, presentan problemas para detectar una falla en el canal de comunicación y poder recuperarse después del evento; por esta razón el submódulo utiliza la siguiente estrategia: Los hilos de ejecución que necesitan enviar un mensaje, no son los que se encargan de ejecutar los métodos del *Socket ZeroMQ* existe un hilo que se encarga de interactuar con el *Socket* para enviar los mensajes, si falla el envío causa el bloqueo del hilo, entonces se detiene el hilo y se le reemplaza por otro hilo funcional. El *Socket* sólo puede enviar un mensaje a la vez, por eso es necesario controlar a los hilos que necesitan enviar un mensaje mediante mecanismos de control de concurrencia, en específico variables condicionales, las cuales pausan a los hilos hasta que el recurso para enviar el mensaje, se encuentre disponible. Una vez un hilo asigne su mensaje para ser enviado, espera hasta que el hilo que trabaja con el *Socket*, regrese una respuesta.

El submódulo posee un hilo de ejecución de verificación, que se encarga de verificar que el hilo mensajero haya enviado el mensaje y regresado una respuesta; cuando se detecta que la respuesta ha tardado más de lo permitido, el hilo mensajero es reemplazado y se reinicia el proceso de transmisión. La estrategia que verifica el tiempo de respuesta a un mensaje, tiene como objetivo evitar que el canal de comunicación no pueda enviar los mensajes. El módulo maneja 2 niveles de complejidad para los mensajes; los mensajes de baja complejidad tienen un menor tiempo de espera y los de alta complejidad un mayor tiempo de espera. De esta forma, se permite controlar el tiempo de espera para cada mensaje.

En la figura 7.1 se muestra un diagrama del submódulo, en el cual se encuentran 5 hilos de ejecución que necesitan enviar un mensaje. Los cuales esperan su turno para asignar su mensaje y el hilo mensajero lo envíe a través del *Socket ZeroMQ*; una vez que un hilo emisor logra asignar su mensaje, espera hasta que el hilo mensajero asigne la respuesta, después, el hilo emisor obtiene la respuesta y se le permite a otro hilo emisor asignar su mensaje. El hilo de verificación siempre verifica que una respuesta no tarde más de lo permitido, si esto ocurre, se reemplaza el hilo mensajero por uno nuevo.

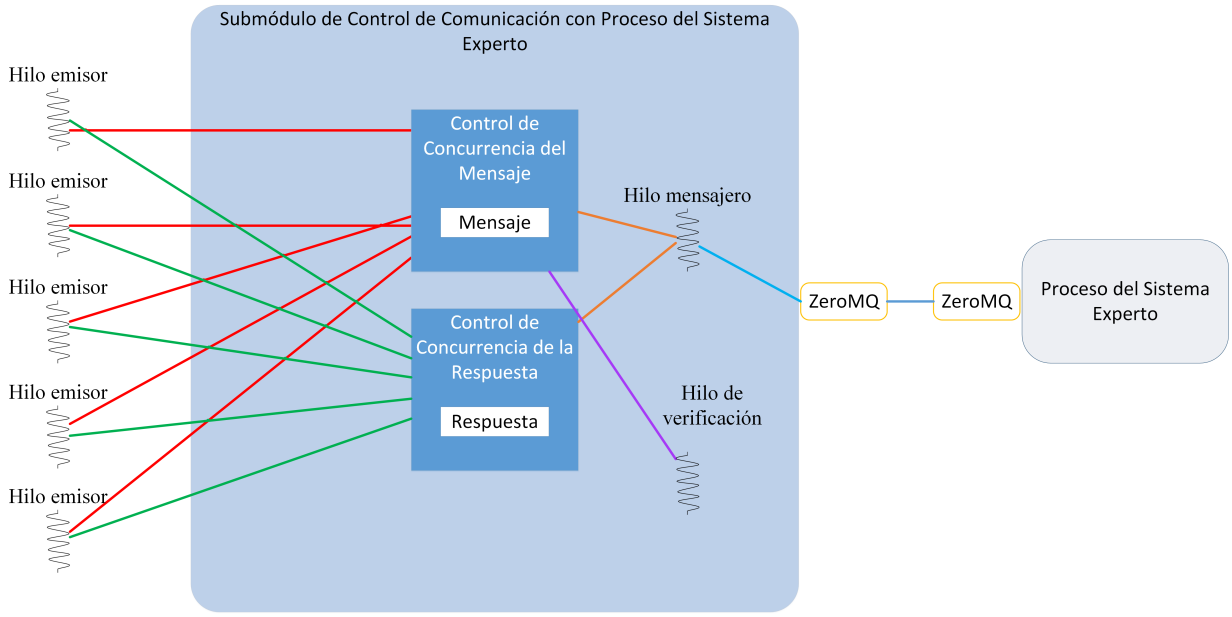

Figura 7.1: Submódulo de comunicación con un proceso del sistema experto.

La estrategia utilizada en el diseño del submódulo de comunicación con un proceso del sistema experto, es la misma que se utiliza en el módulo de comunicación que permite al proceso *REST* enviar los mensajes con los comandos al proceso de control del acceso, en la entidad de acceso.

Módulo para la comunicación con los procesos del sistema experto

El módulo para la comunicación con los procesos expertos tiene como objetivo comunicar a los módulos del proceso de control del sistema, que se encuentra en la entidad del sistema experto. El módulo contiene los submódulos de comunicación con un proceso del sistema experto, los cuales administran los canales de comunicación, que permiten enviar los comandos a los procesos del sistema experto.

El módulo para la comunicación con los procesos del sistema experto crea un nuevo submódulo de comunicación con un proceso del sistema, cada vez que llega un archivo de reglas nuevo al nodo experto, el cual gestiona el envío de mensajes hacia el proceso del sistema.

En la figura 7.2 se muestra el módulo para la comunicación con los procesos del sistema experto, que tiene asociados 4 procesos del sistema experto; por este

motivo, el módulo contiene 4 submódulos de comunicación con un proceso del sistema experto.

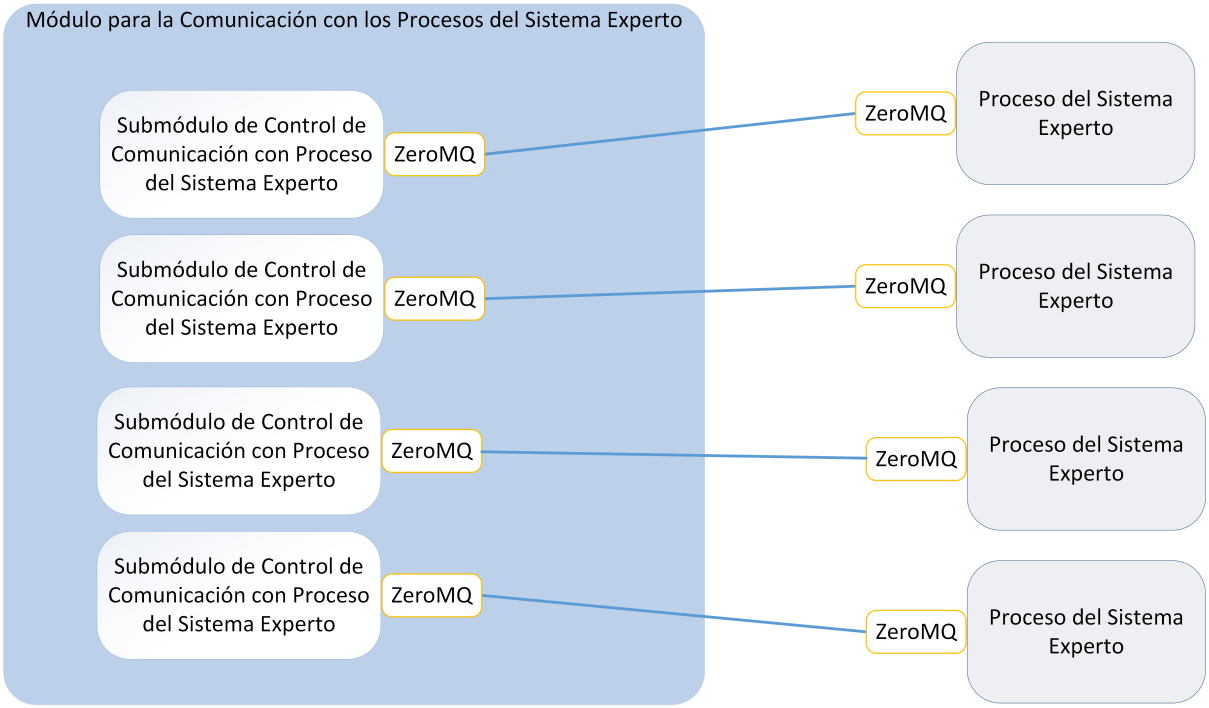

Figura 7.2: Módulo para la comunicación con los procesos del sistema experto.

7.2.2. Comandos y respuestas

El proceso de control del sistema experto que se encuentra en cada nodo experto del sistema distribuido, requiere comunicarse con cada proceso del sistema experto que administra y controla. El mecanismo de comunicación que le permite al proceso de control, enviar y recibir mensajes a cada proceso del sistema experto, es un *Socket* bidireccional del *Middleware ZeroMQ*.

El proceso de control del sistema experto y los procesos del sistema experto que administra, siempre se ejecutan en la misma máquina; por lo tanto, pueden acceder al mismo sistema de archivos y no es necesario transferir archivos a través del mecanismo de comunicación entre ellos. Al no existir la necesidad para transferir datos binarios a través del canal de comunicación entre los procesos, el envío de mensajes con cadenas de texto es suficiente para poder transmitir las órdenes a los

procesos del sistema experto.

Uno de los principales objetivos del diseño de la arquitectura distribuida que se propone, es la capacidad para utilizar una gran variedad de sistemas expertos; por esta razón se utiliza el *Middleware ZeroMQ*. Los programas del sistema experto que se quieran utilizar para los procesos del sistema experto, deben utilizar un *Socket ZeroMQ*; el cual escucha en un puerto que es elegido por el proceso de control del sistema experto y enviado en los parámetros de inicio del proceso. Además, los programas del sistema experto deben entender una lista de comandos y regresar una respuesta definida. A continuación se definen dichos comandos y respuesta; cuya es estructura se compone del comando y los parámetros concatenados utilizando una cadena token de separación.

Comando y Tipo de Petición	**chk-alive-cmd** (Baja complejidad)
Descripción	Comando que sirve para verificar que el proceso del sistema experto asociado, se encuentre en funcionamiento y el proceso de control del sistema experto puede comunicarse con él.
Parámetros	
sin parámetros	
Respuestas	
Respuesta	Descripción
online-chk-alive-resp	El proceso de control del sistema experto puede comunicarse con el proceso del sistema experto.
Cualquier otra respuesta	Cualquier otra respuesta o la ausencia de ella, significa que no se puede comunicar con el proceso del sistema experto.

Tabla 7.1: Especificación del comando **chk-alive-cmd**.

Comando y Tipo de Petición	**clear-knowledge-memory-cmd** (Alta complejidad)
Descripción	Comando que le indica al proceso del sistema experto que debe limpiar la memoria de conocimiento.
Parámetros	
sin parámetros	
Respuestas	
Respuesta	Descripción
ok	El proceso del sistema experto logró limpiar la memoria de conocimiento.
error	El proceso del sistema experto tuvo problemas para limpiar la memoria de conocimiento.

Tabla 7.2: Especificación del comando **clear-knowledge-memory-cmd**.

Comando y Tipo de Petición	**load-rules-file-cmd** (Alta complejidad)
Descripción	Comando que le indica al proceso del sistema experto que debe cargar un archivo de reglas en la memoria de cocimiento.
Token de concatenación	<<;>>
Parámetros	
Ruta del archivo de reglas	Ruta absoluta del archivo de reglas en el sistema de archivos del nodo experto.
Respuestas	
Respuesta	Descripción
ok	El proceso del sistema experto logró cargar el archivo de reglas en la memoria de conocimiento.
error	El proceso del sistema experto tuvo problemas para cargar el archivo de reglas en la memoria de conocimiento.
inexistent-file	El archivo de reglas no se encuentra en la ruta absoluta indicada.
unknown-command	El proceso del sistema experto indica que no reconoce el comando, pues no recibió la ruta absoluta del archivo de reglas.

Tabla 7.3: Especificación del comando **load-rules-file-cmd**.

Comando y Tipo de Petición	**unload-rules-cmd** (Alta complejidad)
Descripción	Comando que le indica al proceso del sistema experto que debe eliminar las reglas que se encuentran en la memoria de conocimiento.
Parámetros	
sin parámetros	
Respuestas	
Respuesta	Descripción
ok	El proceso del sistema experto logró eliminar las reglas que se encuentran en la memoria de conocimiento.
error	El proceso del sistema experto problemas para eliminar las reglas que se encuentran en la memoria de conocimiento.

Tabla 7.4: Especificación del comando **unload-rules-cmd**.

Comando y Tipo de Petición	**process-facts-file-cmd** (Alta complejidad)
Descripción	Comando que le indica al proceso del sistema experto que debe procesar un archivo de hechos, utilizando las reglas que tiene cargadas en la memoria de conocimiento.
Token de concatenación	<<;>>
Parámetros	
Ruta del archivo de hechos	Ruta absoluta del archivo de hechos en el sistema de archivos del nodo experto.
Respuestas	
Respuesta	Descripción
ok	El proceso del sistema experto realizó con éxito el procedimiento de inferencia con el archivo de hechos.
error	El proceso del sistema experto tuvo problemas para realizar el procedimiento de inferencia con el archivo de hechos.
inexistent-file	El archivo de hechos no se encuentra en la ruta absoluta indicada.
unknown-command	El proceso del sistema experto indica que no reconoce el comando, pues no recibió la ruta absoluta del archivo de hechos.

Tabla 7.5: Especificación del comando **process-facts-file-cmd**.

Comando y Tipo de Petición	**end-cmd** (Baja complejidad)
Descripción	Comando que le indica al proceso del sistema experto que termina se ejecución.
Parámetros	
sin parámetros	
Respuestas	
Respuesta	Descripción
ok	El proceso del sistema experto recibió el comando.
Sin respuesta	El proceso del sistema experto no recibió el comando.

Tabla 7.6: Especificación del comando **end-cmd**.

7.3. Fortalezas

El *Middleware* de comunicación *ZeroMQ* permite utilizar una gran variedad de sistemas expertos con la arquitectura distribuida, lo que incrementa los casos en los que se puede utilizar el sistema distribuido que se propone.

El submódulo de comunicación con el proceso del sistema experto es el encargado de gestionar el envío de comandos al proceso, sólo permite que un comando se

ejecute a la vez y hace esperar al siguiente comando, hasta que el primero tenga una respuesta; esto permite que los programas de los sistemas expertos no tengan que implementar mecanismos para el control de concurrencia, pero eso depende de quién implemente dicho programa.

El módulo para la comunicación con los procesos del sistema experto es el encargado de trabajar con cada submódulo de comunicación con un proceso del sistema experto, de esta forma encapsula toda la complejidad de las comunicaciones con los procesos del sistema experto; lo que facilita el trabajo para los otros módulos que se encuentran en el proceso de control del sistema experto.

7.4. Deficiencias

El mecanismo de comunicación que se utiliza para comunicarse con los procesos del sistema experto sólo puede transmitir un mensaje a la vez, lo cual limita la cantidad de acciones que cada proceso del sistema experto puede realizar a la vez; para solucionar este problema se puede utilizar el mismo enfoque utilizado en las entidades del sistema distribuido, el cual es el uso del patrón de diseño, llamado grupo de hilos de ejecución. Utilizar el patrón anterior permite que el proceso del sistema experto pueda realizar varias tareas al mismo tiempo; esto hace posible que cada vez que llega un nuevo comando el proceso, se asigna a un hilo de ejecución del grupo para su procesamiento, brindando la capacidad de recibir una mayor cantidad de comandos.

El programador de cada programa ejecutable de sistema experto a utilizar con el sistema distribuido, es el responsable de la decisión del uso de un enfoque de múltiples hilos de ejecución; existen algunos *Frameworks* de sistemas expertos que sólo pueden ejecutar una tarea a la vez, en este caso el programador no requiere agregar algún control de concurrencia, el canal sólo permite un mensaje a la vez y esto limita el envío de comandos hasta recibir la respuesta correspondiente.

7.5. Resumen

El módulo de comunicación con los procesos del sistema experto permite utilizar gran variedad de sistemas expertos con el sistema distribuido. El módulo utiliza un mecanismo de comunicación que se puede utilizar con una amplia variedad de lenguajes de programación; además permite un canal de comunicación bidireccional entre el proceso de control del sistema experto y cada proceso del sistema experto.

El módulo de comunicación con los procesos del sistema experto, controla los canales de comunicación de cada proceso del sistema experto; el control de cada canal de comunicación se delega a un submódulo, siguiendo con el patrón de diseño modular. El submódulo de comunicación con un proceso del sistema experto, es el encargado de gestionar el envío de mensajes a través del canal de comunicación que existe entre el proceso de control del sistema experto y un proceso del sistema experto. El módulo para la comunicación con los procesos del sistema experto tiene como objetivo comunicar a los módulos del proceso de control del sistema experto, que se encuentra en la entidad del sistema experto; el módulo contiene los submódulos de comunicación con un proceso del sistema experto, los cuales administran los canales de comunicación, que permiten enviar los comandos a los procesos del sistema experto.

Uno de los principales objetivos del diseño de la arquitectura distribuida , es poder utilizar una gran variedad de sistemas expertos; por esta razón se utiliza el *Middleware ZeroMQ*. Los programas del sistema experto que se quieran utilizar para los procesos del sistema experto, deben utilizar un *Socket ZeroMQ* y deben entender una lista de comandos y regresar una respuesta definida.

Capítulo 8

Pruebas del sistema

Capítulo en el que se muestran los resultados de las pruebas que se le realizaron al sistema distribuido implementado con la arquitectura propuesta. Las pruebas tienen como objetivo verificar que se haya cumplido el objetivo del presente trabajo de tesis; comprobar que el sistema distribuido implementado tenga las siguientes características: Escalabilidad, tolerancia a fallas y alta disponibilidad.

Las pruebas se realizaron utilizando programa que utiliza como sistema experto a CLIPS [13], el cual no es un sistema experto distribuido. Las pruebas no están enfocadas en probar el funcionamiento como sistema experto distribuido, sino las capacidades de la arquitectura distribuida para proveer las características: Escalabilidad, tolerancia a fallas y alta disponibilidad. El programa del sistema experto que se utiliza para las pruebas, sólo está enfocado en mostrar el tiempo necesario para que el sistema distribuido ejecute sus funciones.

Cuando se utiliza un sistema experto distribuido con la arquitectura que se propone, cada proceso experto debe tener la capacidad de distribuir su memoria de su conocimiento, pero no deben compartir espacio de reglas; ésta es una de las tareas de la arquitectura propuesta.

8.1. Definición de las pruebas

En esta sección se definen las pruebas a las que se somete el sistema distribuido implementado, las pruebas están destinadas para verificar el funcionamiento del sis-

tema, también comprueban que el sistema distribuido cumpla con las características originalmente buscadas: Escalabilidad, tolerancia a fallas y alta disponibilidad; lo cual es el objetivo del presente trabajo de tesis. Para verificar la escalabilidad del sistema; las pruebas se realizan con configuraciones que utilizan distintas cantidades de nodos de acceso y nodos expertos; también se utilizaron distintas cantidades de archivos de reglas y hechos, con el fin de probar distintos niveles de carga de trabajo.

Inicialización del sistema distribuido

Pruebas que tienen como objetivo verificar que el sistema distribuido inicie correctamente; cada nodo de acceso del sistema debe detectar y obtener acceso a todos los demás nodos presentes en el sistema distribuido. En estas pruebas todos los nodos de acceso y los nodos expertos se inician al mismo tiempo, las pruebas concluye cuando cada nodo de acceso ha descubierto e iniciado la comunicación con todos los otros nodos del sistema distribuido; para estas pruebas se registra el tiempo que dura todo el proceso de inicialización.

Descubrimiento de nodos de acceso nuevos

Pruebas en las que se verifica que los nodos de acceso que ya se encuentran en funcionamiento en el sistema distribuido, sean capaces de detectar y comunicarse con los nodos de acceso que se agreguen al sistema. En estas pruebas se mide el tiempo que tardan los nodos de acceso en detectar a todos los nodos de acceso recién agregados; los nuevos nodos se inician a la misma hora, de esta forma, es posible registrar el tiempo total que dura el descubrimiento y la conexión.

Descubrimiento de nodos expertos nuevos

Pruebas en las que se verifica que los nodos de acceso que ya se encuentran en funcionamiento en el sistema distribuido, sean capaces de detectar y comunicarse con los nodos expertos que se agreguen al sistema. En estas pruebas se mide el tiempo que tardan los nodos de acceso en detectar a todos los nodos expertos recién agregados; los nuevos nodos se inician a la misma hora, de esta forma, es posible registrar el tiempo total que dura el descubrimiento y la conexión.

Registro de archivos de reglas

Pruebas para el registro y la distribución de los archivos de reglas en el sistema distribuido; se verifica el funcionamiento del sistema cuando se utilizan las distintas políticas de distribución de reglas. En estas pruebas se muestra como se distribuyen los archivos de reglas entre todos los nodos expertos presentes en el sistema distribuido; también se registra el tiempo que toma todo el proceso completo, así como algunos subprocesos claves.

Detección de la falla de nodos de acceso

Pruebas en las que se verifica la capacidad de los nodos de acceso para detectar a los nodos de acceso que han fallado y que no responden a las peticiones. En estas pruebas se mide el tiempo que transcurre desde que los nodos de acceso fallan, hasta que se detecta la falla de los nodos de acceso y son removidos del sistema distribuido. Para estas pruebas los nodos fueron detenidos al mismo tiempo.

Detección de la falla de nodos expertos

Pruebas en las que se verifica la capacidad de los nodos de acceso para detectar los nodos expertos que han fallado y no responden a las peticiones. En estas pruebas se mide el tiempo que transcurre desde que los nodos expertos fallan, hasta que se detecta la falla de los nodos expertos y son removidos del sistema distribuido. Para estas pruebas los nodos fueron detenidos al mismo tiempo.

Distribución de archivos en el sistema distribuido

Pruebas que están enfocadas en comprobar el funcionamiento de la plataforma para la distribución de archivos en el sistema distribuido. Las pruebas verifican que los archivos se distribuyen a todos los nodos expertos en el sistema distribuido; también se mide el tiempo que tardan los archivos en ser enviados.

Recuperación de archivos en nodos de acceso fallidos

Pruebas cuyo propósito es comprobar la capacidad del sistema distribuido, para recuperar los archivos que se encuentran procesando nodos de acceso que fallan en el sistema; en las pruebas se verifica que todos los archivos sean recuperados y procesados por alguno de los otros nodos de acceso que continúan funcionando. Además de constatar que la recuperación funciona correctamente, las pruebas permiten medir el tiempo necesario para completar el proceso de recuperación.

Recuperación de archivos en nodos expertos fallidos

Pruebas cuyo propósito es comprobar la capacidad del sistema distribuido, para recuperar los archivos que se encuentran procesando nodos expertos que fallan en el sistema; en las pruebas se verifica que todos los archivos sean recuperados y procesados por alguno de los nodos de acceso en el sistema distribuido. Los nodos de acceso asignan los archivos recuperados en otros nodos expertos que continúan funcionando en el sistema. Las pruebas permiten medir el tiempo necesario para completar el proceso de recuperación.

8.2. Plataforma de pruebas

Las pruebas del sistema distribuido se realizaron en una máquina con un procesador de cuatro núcleos *Intel Core i7 4700MQ* con 2.4 GHz de frecuencia y 16179 MB de memoria *RAM*; la cual ejecuta Windows 7 64 bits. Para comprobar el funcionamiento distribuido del sistema es necesario realizar pruebas con varias máquinas; por esta razón se utilizaron máquinas virtuales conectadas entre sí a través de una red *LAN* virtual, la cual permite utilizar el protocolo *Multicast*.

En el proceso de pruebas se utilizan 4 máquinas virtuales, las cuales se ejecutan en el sistema de virtualización *Virtualbox* [1]. La primera máquina virtual además de ejecutar el sistema distribuido, también se utiliza para ejecutar las herramientas con las que se realizaron las pruebas del sistema; la máquina virtual fue configurada para virtualizar 4 procesadores y 4096 MB de memoria *RAM*. Dos máquinas virtuales se

[1] `https://www.virtualbox.org/`

utilizan para ejecutar el sistema distribuido y fueron configuradas para virtualizar 2 procesadores y 3072 MB de memoria *RAM*. La última máquina virtual contiene el sistema gestor de la base de datos que utiliza el sistema distribuido, configurada con 1 procesador y 512 MB de memoria *RAM*.

Las máquinas virtuales en donde se ejecuta el sistema distribuido, utilizan *GNU/Linux Fedora* 20 de 64 bits con *Kernel* 3.16.2-200.fc20.x86_64. La máquina virtual que ejecuta el sistema gestor de bases de datos utiliza *Windows XP* de 32 bits como sistema operativo.

La comunicación entre todas las máquinas virtuales se realiza a través de una red virtual, a la cual se conecta cada una de las máquinas y permite conexiones *Multicast*.

8.3. Resultados obtenidos

En esta sección se muestran los resultados obtenidos de las pruebas que se realizaron al sistema distribuido, los cuales se representan a través de tablas y gráficas. En todas las pruebas realizadas al sistema distribuido, se efectuaron 20 ejecuciones por cada combinación de las variables en los casos de pruebas, variando la cantidad de nodos de acceso, nodos expertos, archivos de reglas y archivos de hechos.

Las pruebas que se realizaron al sistema distribuido, se hicieron utilizando una cantidad variable de nodos de accesos y nodos expertos; los cuales se distribuyeron entre las máquinas virtuales disponibles para las pruebas, esto con el objetivo de balancear la carga de trabajo.

Inicialización del sistema distribuido

Pruebas en las que se verifica la correcta iniciación del sistema distribuido. Todos los nodos que componen el sistema distribuido se inician al mismo tiempo y se mide el tiempo necesario para que el sistema complete su inicio. En la tabla 8.1 se muestra el tiempo promedio en segundos que necesita el sistema distribuido para iniciar con distintas cantidades de nodos de acceso y nodos expertos.

	1 nodo de acceso	2 nodos de acceso	3 nodos de acceso
1 Nodo experto	10.686 s	11.442 s	12.048 s
3 Nodos expertos	11.234 s	12.029 s	12.676 s
5 Nodos expertos	12.200 s	15.797 s	16.128 s
7 Nodos expertos	19.706 s	20.047 s	20.911 s
9 Nodos expertos	20.628 s	21.087 s	22.177 s

Tabla 8.1: Tiempo de inicialización del sistema distribuido.

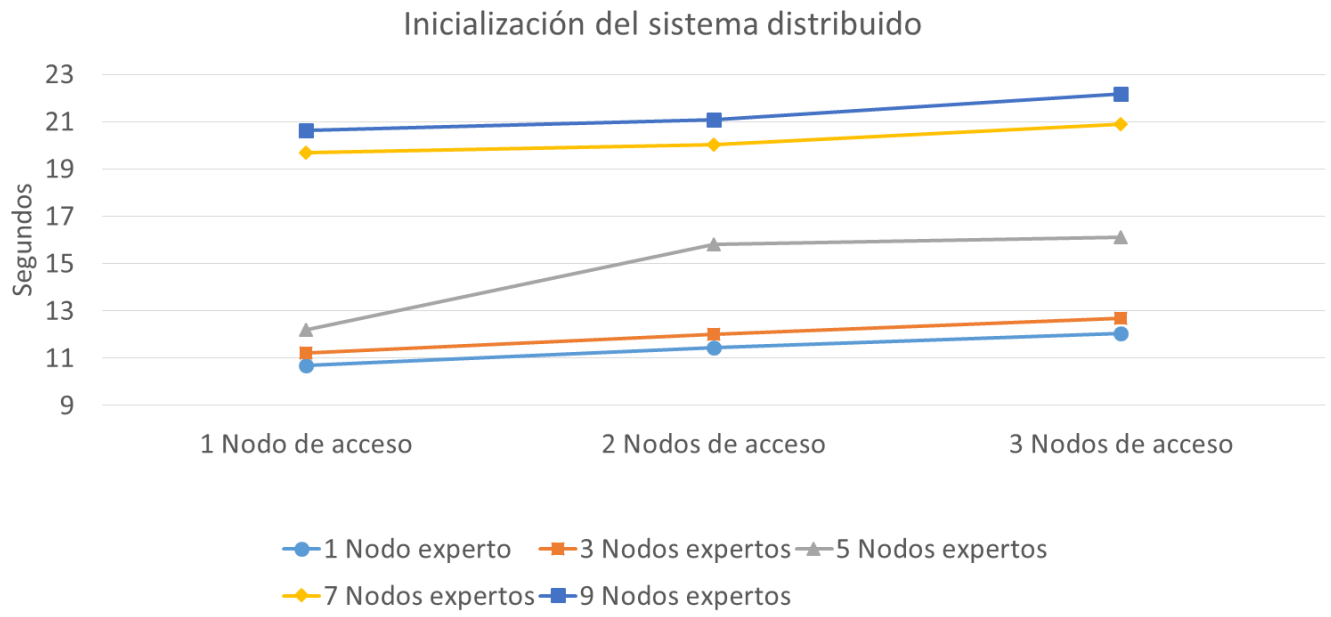

Figura 8.1: Tiempo de inicialización del sistema distribuido.

En los resultados para las pruebas de inicialización se observa que el sistema distribuido necesita más tiempo para iniciar conforme se agregan más nodos expertos y más nodos de acceso; esto ocurre debido a la necesidad de configurar más nodos, aunque ésto se realiza de manera concurrente, las pruebas se realizaron en máquinas virtuales. En la gráfica 8.1 se observa un aumento lineal del tiempo necesario para el inicio con uno, tres, siete y nueve nodos expertos, pero en el caso con cinco nodos expertos se comporta de una manera diferente; realizar pruebas con una mayor cantidad de nodos permitiría apreciar mejor el comportamiento del sistema distribuido

cuando inicia, pero se requieren pruebas en máquinas físicas y no virtuales.

Los resultados en este caso de pruebas mostraron que el sistema distribuido puede escalar la cantidad de nodos de acceso y expertos con los que se inicializa, pues aunque se agregan más nodos, el tiempo necesario para el inicio y configuración automática del sistema distribuido aumenta de una manera estable y gradual. Aunque el sistema distribuido muestra un comportamiento escalable en el proceso de inicialización, en algún punto el tiempo de inicio del sistema distribuido será muy alto y no será conveniente utilizar más nodos en el inicio del sistema.

Descubrimiento de nodos de acceso nuevos

Pruebas en las que se mide el tiempo promedio que necesitan uno o varios nodos de acceso que ya se encuentran funcionando en el sistema distribuido, para descubrir y configurar a uno o más de acceso que recién se integran al sistema distribuido. En la tabla 8.2 se muestra el tiempo promedio que necesitan desde 1 nodo de acceso hasta 3 nodos de acceso, para detectar desde 1 nodo de acceso hasta 3 nodos de acceso que fueron agregados al sistema distribuido en ejecución.

Estas pruebas están enfocadas en medir el tiempo necesario para agregar uno o varios nodos de acceso al sistema distribuido en funcionamiento, pues es importante que el sistema tenga la capacidad para agregar nuevos nodos de acceso, ya que ellos son los encargados de atender a los usuarios y administrar el funcionamiento del sistema; cuantos más nodos de acceso tenga el sistema distribuido, más usuarios y peticiones puede atender.

	1 nodo de acceso	2 nodos de acceso	3 nodos de acceso
1 nodo de acceso detectado	9.76 s	10.079 s	11.342 s
2 nodos de acceso detectados	11.45 s	11.493 s	12.712 s
3 nodos de acceso detectados	12.112 s	12.54 s	12.833 s

Tabla 8.2: Tiempo de detección de nodos de acceso nuevos.

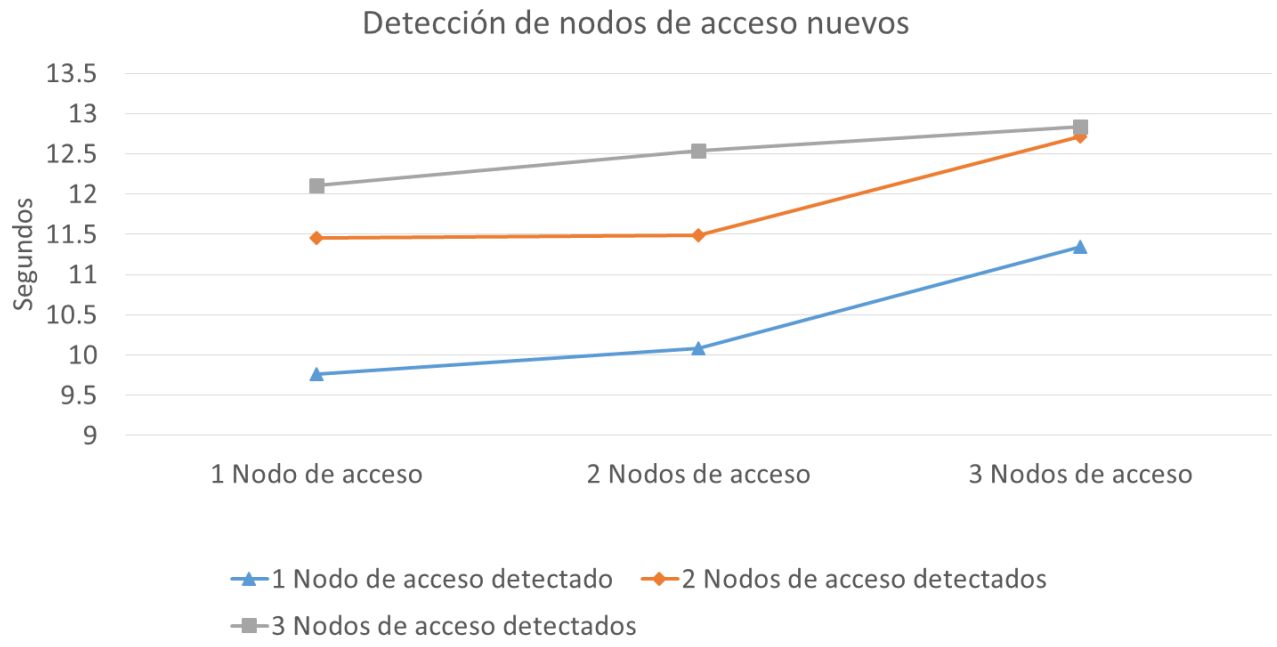

Figura 8.2: Tiempo de detección de nodos de acceso nuevos.

Los resultados obtenidos para este caso de prueba muestran que los nodos de acceso en el sistema distribuido requieren más tiempo conforme se requiere descubrir una mayor cantidad de nodos de acceso nuevo, también aumenta cuando la cantidad de nodos de acceso que ya se encuentran en funcionamiento aumentan. El tiempo necesario para detectar nodos puede incrementarse debido al aumento de la carga de trabajo del servicio de nombres de *CORBA*; también puede aumentar cuando un nodo debe configurar más nodos en sus registros, lo que agrega más carga de trabajo en la máquina anfitrión y en un ambiente virtual ésto puede el rendimiento del sistema distribuido. Los resultados muestran que el sistema distribuido tiene una capacidad de escalamiento con un límite.

Descubrimiento de nodos expertos nuevos

Pruebas en las que se mide el tiempo promedio que necesitan uno o varios nodos de acceso que se encuentran funcionando en el sistema distribuido, para descubrir y configurar a uno o más nodos expertos que recién se integran al sistema distribuido.

En la tabla 8.3 se muestra el tiempo promedio que necesitan desde 1 nodo de acceso hasta 3 nodos de acceso, para detectar desde 1 nodo experto hasta 3 nodos expertos que fueron agregados al sistema distribuido en ejecución.

	1 nodo de acceso	2 nodos de acceso	3 nodos de acceso
1 Nodo experto	1.601 s	1.722 s	1.857 s
2 Nodos expertos	1.715 s	1.897 s	2.004 s
3 Nodos expertos	2.006 s	2.058 s	2.152 s

Tabla 8.3: Tiempo de detección de nodos expertos nuevos.

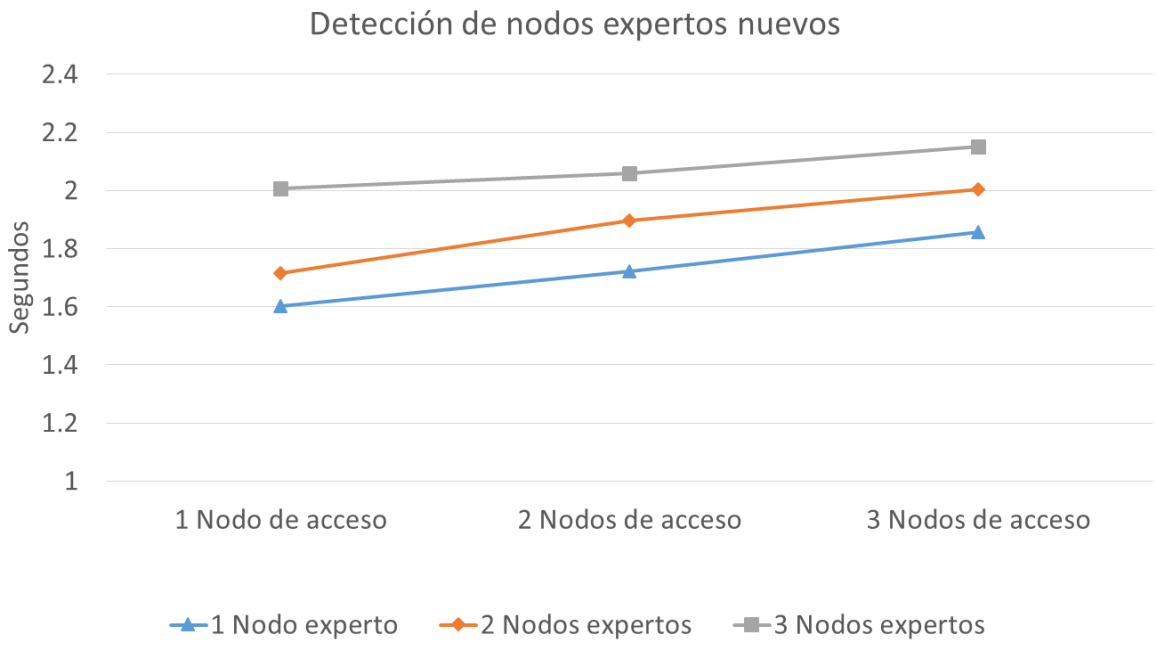

Figura 8.3: Tiempo de detección de nodos expertos nuevos.

En este caso de prueba los resultados mostraron el mismo comportamiento que en el caso de detección de nodos de acceso, pues conforme se aumenta la cantidad de nodos de expertos a detectar, aumenta el tiempo necesario. Una de las razones del aumento del tiempo necesario para la detección de los nuevos nodos es el servicio de nombres de *CORBA*, el cual es un servicio centralizado; por lo cual entre más peticiones recibe, puede requerir mayor tiempo para atenderlas.

Registro de archivos de reglas

Pruebas con las que se verifica que el sistema distribuido puede registrar los archivos de reglas y se distribuyen entre los nodos expertos presentes. El sistema distribuido posee tres políticas de asignación, con estas pruebas se registra la forma en la que se distribuyen los archivos de reglas y el tiempo necesario para registrar un archivo de reglas dependiendo de la cantidad de nodos expertos presentes en el sistema distribuido.

La política de distribución de archivos de reglas basada en la cantidad de archivos de reglas en los nodos expertos, determina los nodos expertos con menor carga de trabajo utilizando como métrica la cantidad de archivos de reglas que poseen. En la tabla 8.4 se muestra la manera en la que se distribuyen los archivos de reglas en las pruebas realizadas, en las cuales variaron la cantidad de nodos expertos presentes en el sistema y la cantidad de archivos de reglas a registrar.

	3 Expertos	5 Expertos	7 Expertos	9 Expertos
15 Archivos de reglas	5-5-5	3-3-3-3-3	2-2-2-2-2-2-3	1-1-1-2-2-2-2-2-2
30 Archivos de reglas	10-10-10	6-6-6-6-6	4-4-4-4-4-5-5	3-3-3-3-3-3-4-4-4
45 Archivos de reglas	15-15-15	9-9-9-9-9	6-6-6-6-7-7-7	5-5-5-5-5-5-5-5-5
60 Archivos de reglas	20-20-20	12-12-12-12-12	8-8-8-9-9-9-9	6-6-6-7-7-7-7-7-7

Tabla 8.4: Distribución de archivos basada en la cantidad de archivos de reglas en los nodos expertos.

En la tabla 8.5 se muestra el tiempo promedio de cada una de las etapas del registro un archivo de reglas, cuando se utiliza la política de distribución de archivos de reglas basada en la cantidad de archivos de reglas en los nodos expertos. Las etapas del procedimiento son las siguientes: procesamiento de las peticiones *REST*, selección del nodo experto, transferencia al nodo experto y creación del proceso experto en el nodo experto destino, también se muestra el tiempo total del procesamiento del archivo de reglas en el sistema distribuido. Las pruebas se realizaron variando la cantidad de nodos expertos presentes en el sistema distribuido y con una cantidad fija de 3 nodos de acceso. Los resultados que se muestra en la tabla 8.5, también se encuentran en la figura 8.4.

	3 Expertos	5 Expertos	7 Expertos	9 Expertos
Procesamiento *REST*	1.454 s	1.519 s	1.366 s	1.499 s
Selección nodo experto	0.001 s	0.001 s	0.001 s	0.001 s
Transferencia nodo experto	0.987 s	0.931 s	0.848 s	0.973 s
Creación proceso experto	1.809 s	1.795 s	1.798 s	1.792 s
Procesamiento completo	5.498 s	5.916 s	4.619 s	4.986 s

Tabla 8.5: Tiempo del registro un archivo de reglas utilizando la política de distribución basada en la cantidad de archivos de reglas en los nodos expertos.

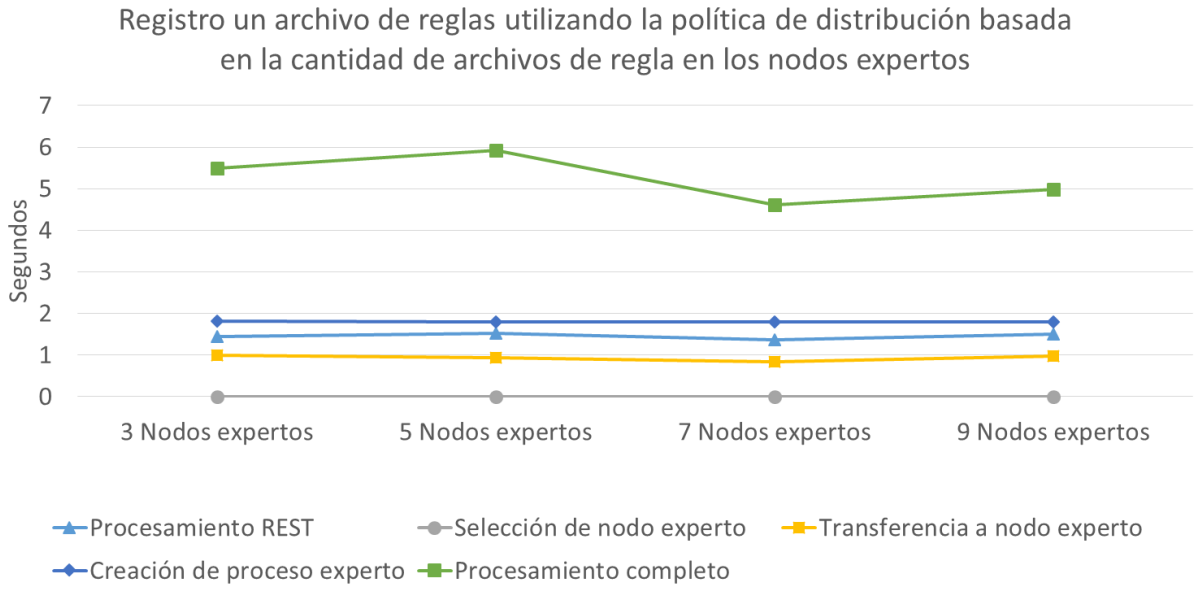

Figura 8.4: Tiempo del registro un archivo de reglas utilizando la política de distribución basada en la cantidad de archivos de reglas en los nodos expertos.

Los resultados en la tabla 8.5 y la figura 8.4 muestran un comportamiento estable en el proceso de distribución de reglas cuando se utiliza la política de distribución de archivos de reglas basada en la cantidad de archivos de reglas en los nodos expertos. Los procedimientos que componen el proceso de registro requieren un tiempo muy similar, aun cuando aumenta la cantidad de nodos expertos presentes en el sistema distribuido; el tiempo total muestra un patrón de oscilación con tendencia a disminuir, pero se requieren pruebas con más nodos expertos para comprobarlo.

La política de distribución de archivos de reglas basada en la cantidad de reglas en los nodos expertos, determina los nodos expertos con menor carga de trabajo utilizando como métrica la de reglas que poseen. En la tabla 8.6 se muestra la manera con mayor cantidad de ocurrencias, en la que se distribuyeron los archivos de reglas en las pruebas que se realizaron, en las cuales variaron la cantidad de nodos expertos presentes en el sistema y la cantidad de archivos de reglas a registrar.

	3 Expertos	5 Expertos	7 Expertos	9 Expertos
15 Archivos de reglas	4-5-6	2-3-3-3-4	1-2-2-2-2-3-3	1-1-1-1-2-2-2-2-3
30 Archivos de reglas	9-10-11	4-5-5-7-9	3-3-4-5-5-5-5	2-2-3-3-3-4-4-4-5
45 Archivos de reglas	14-15-16	8-8-9-10-10	5-5-6-6-7-7-9	2-3-3-4-4-7-7-7-8
60 Archivos de reglas	14-21-25	7-12-12-12-17	5-8-9-9-9-10-10	3-5-6-6-7-7-8-9-9

Tabla 8.6: Distribución de archivos basada en la cantidad de reglas en los nodos expertos.

La política de distribución de archivos de reglas basada en la cantidad de reglas en los nodos expertos, no asigna de la misma forma los archivos de reglas comparada con las otras políticas, pues se basa en la cantidad de reglas que contienen los nodos expertos; utilizar este valor como métrica ocasiona una distribución muy variable de los archivos de reglas. En la tabla 8.6 se muestran las distribuciones de reglas que más repeticiones ocurrieron durante las pruebas, pero no son las únicas distribuciones que ocurrieron.

En todas las pruebas del registro de archivos de reglas se utilizaron los mismos archivos, los cuales no poseen la misma cantidad de reglas. Durante las pruebas de cada política de asignación, los archivos de reglas se registraron en el mismo orden

En la tabla 8.5 se muestra el tiempo promedio de cada una de las etapas del registro un archivo de reglas, cuando se utiliza la política de distribución de archivos de reglas basada en la cantidad de reglas en los nodos expertos. Las etapas del procedimiento son las siguientes: procesamiento de las peticiones REST, selección del nodo experto, transferencia al nodo experto y creación del proceso experto en el nodo experto destino, también se muestra el tiempo total del procesamiento del archivo de reglas en el sistema distribuido. Las pruebas se realizaron variando la cantidad de nodos expertos presentes en el sistema distribuido y con una cantidad

fija de 3 nodos de acceso.

	3 Expertos	5 Expertos	7 Expertos	9 Expertos
Procesamiento *REST*	1.635 s	1.441 s	1.552 s	1.414 s
Selección nodo experto	0.001 s	0.002 s	0.002 s	0.002 s
Transferencia nodo experto	1.134 s	1.071 s	0.974 s	1.144 s
Creación proceso experto	1.82 s	1.822 s	1.72 s	1.83 s
Procesamiento completo	5.601 s	5.142 s	5.131 s	5.463 s

Tabla 8.7: Tiempo del registro un archivo de reglas utilizando la política de distribución basada en la cantidad de reglas en los nodos expertos.

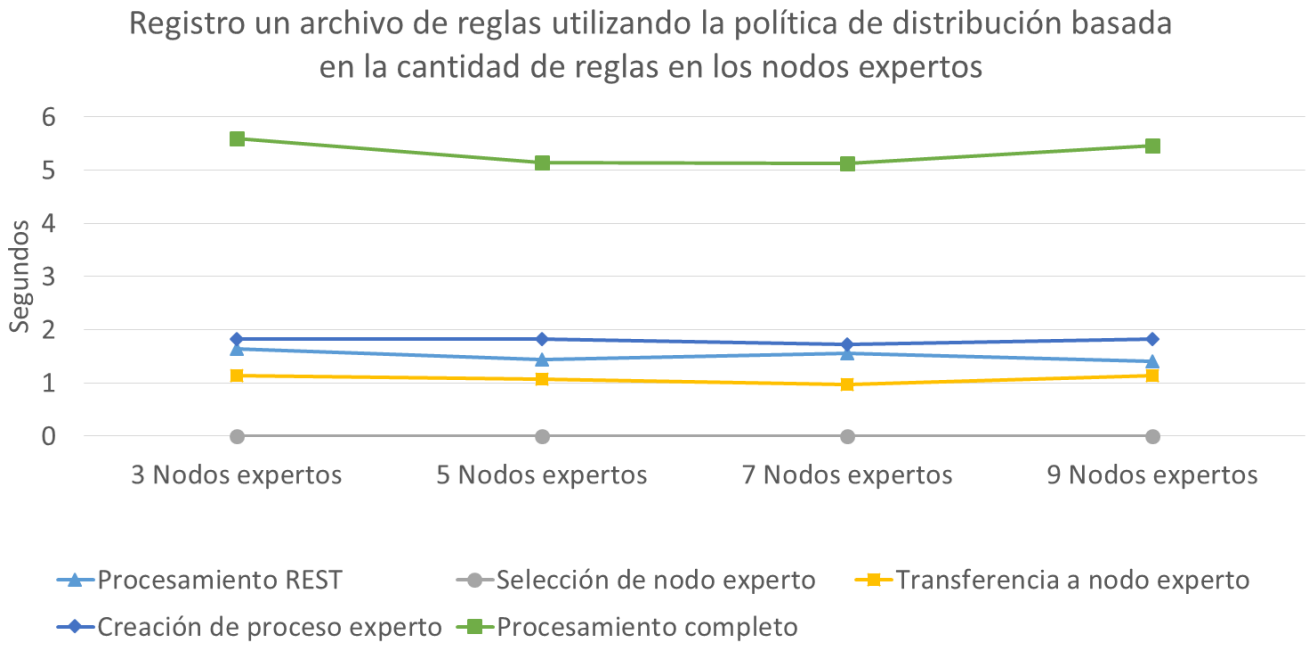

Figura 8.5: Tiempo del registro un archivo de reglas utilizando la política de distribución basada en la cantidad de reglas en los nodos expertos.

Los resultados que se obtuvieron cuando se realizaron las pruebas del registro de archivos de reglas cuando se utiliza la política de distribución de archivos de reglas basada en la cantidad de reglas en los nodos expertos, mostraron un comportamiento estable en el tiempo necesario para cada uno de los procedimientos; aunque el tiempo

necesario para seleccionar el nodo experto destino aumenta conforme aumenta la cantidad de ellos, para comprobar esto es necesario realizar pruebas con una mayor cantidad de nodos expertos.

La política de distribución de archivos de reglas basada en la iteración Round-Robin de una lista de nodos expertos, todos los nodos poseen la misma lista y apuntan a la misma posición; esta política se explica en la sección 4.3.1. En la tabla 8.8 se muestra la manera en la que se distribuyen los archivos de reglas en las pruebas realizadas, en las cuales variaron la cantidad de nodos expertos presentes en el sistema y la cantidad de archivos de reglas a registrar.

	3 Expertos	5 Expertos	7 Expertos	9 Expertos
15 Archivos de reglas	5-5-5	3-3-3-3-3	2-2-2-2-2-2-3	1-1-1-2-2-2-2-2-2
30 Archivos de reglas	10-10-10	6-6-6-6-6	4-4-4-4-4-5-5	3-3-3-3-3-3-4-4-4
45 Archivos de reglas	15-15-15	9-9-9-9-9	6-6-6-6-7-7-7	5-5-5-5-5-5-5-5-5
60 Archivos de reglas	20-20-20	12-12-12-12-12	8-8-8-9-9-9-9	6-6-6-7-7-7-7-7-7

Tabla 8.8: Distribución de archivos basada en la iteración Round-Robin de una lista de nodos expertos.

Las listas de nodos expertos utilizadas para las pruebas con la política de asignación Round-Robin, contienen una vez a todos los nodos expertos que se encuentran en ejecución en el sistema experto en cada uno de las configuraciones de las pruebas. Cuando se utilizan listas que incluyen a todos los nodos expertos del sistema distribuido, la distribución de archivos se comporta como en el caso de la distribución basada en la cantidad de archivos de reglas en los nodos expertos.

En las listas de nodos expertos se pueden omitir los nodos expertos a los cuales no se les quiere asignar archivos de reglas; también es posible agregar a un nodo experto más de una vez en una lista, de esta forma, se puede asignar mayor carga de trabajo a nodos con mejores especificaciones de *hardware*.

En la tabla 8.9 se muestra el tiempo promedio de cada una de las etapas del registro un archivo de reglas, cuando se utiliza la política de distribución de archivos de reglas basada en la iteración *Round-Robin* de una lista de nodos expertos. Las etapas del procedimiento son las siguientes: procesamiento de las peticiones REST, selección del nodo experto, transferencia al nodo experto y creación del proceso experto

en el nodo experto destino, también se muestra el tiempo total del procesamiento del archivo de reglas en el sistema distribuido. Las pruebas se realizaron variando la cantidad de nodos expertos presentes en el sistema distribuido y con una cantidad fija de 3 nodos de acceso.

	3 Expertos	5 Expertos	7 Expertos	9 Expertos
Procesamiento *REST*	1.502 s	1.584 s	1.531 s	1.428 s
Selección nodo experto	0.002 s	0.002 s	0.001 s	0.001 s
Transferencia nodo experto	0.966 s	1.077 s	0.831 s	0.979 s
Creación proceso experto	2.184 s	1.725 s	1.812 s	1.83 s
Procesamiento completo	5.639 s	5.369 s	4.874 s	5.007 s

Tabla 8.9: Tiempo del registro un archivo de reglas utilizando la política de distribución basada en la iteración *Round-Robin* de una lista de nodos expertos.

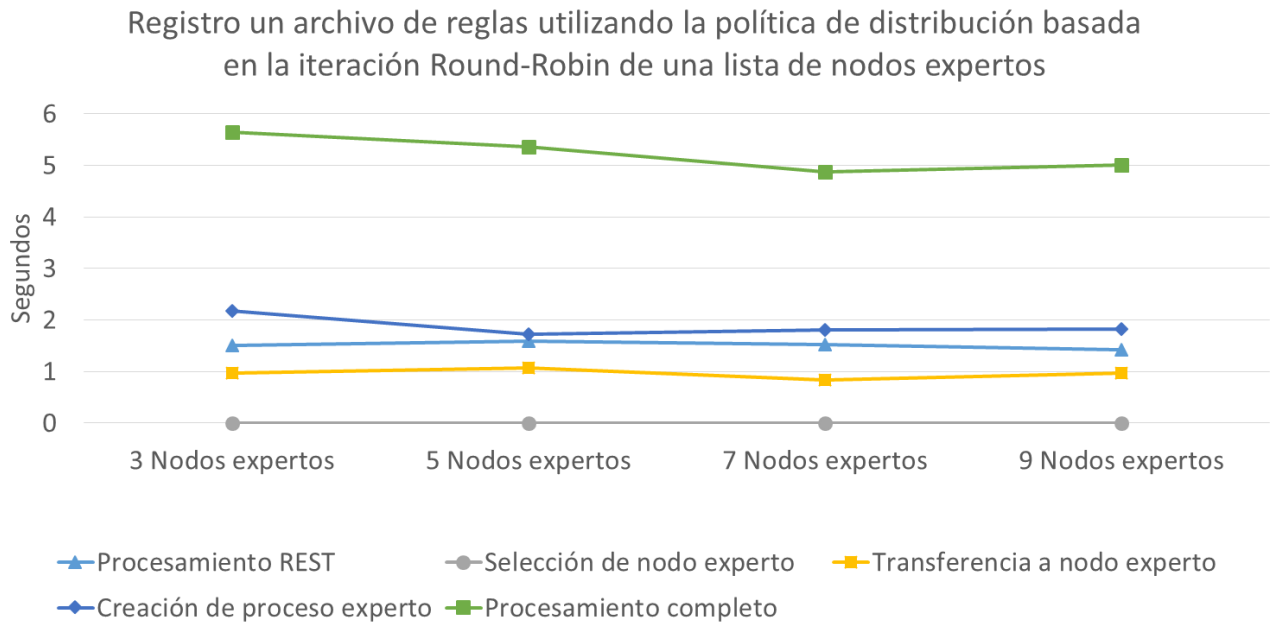

Figura 8.6: Tiempo del registro un archivo de reglas utilizando la política de distribución basada en la iteración *Round-Robin* de una lista de nodos expertos.

En el caso de las pruebas utilizando la política de distribución de archivos de

reglas basada en la iteración *Round-Robin* de una lista de nodos expertos, muestra que el tiempo necesario para algunos de los procedimientos que forman el proceso de registro completo, tienden a reducir el tiempo que necesitan conforme aumenta la cantidad de nodos expertos, así como el tiempo total; pero después se incrementan, aun así se mantienen en un rango muy cercano y estable. Es necesario tener en consideración que las pruebas se realizaron en sólo una máquina física utilizando varias máquinas virtuales, lo cual en algunas ocasiones puede repercutir en el desempeño y rendimiento del sistema distribuido en las pruebas de este tipo.

Comparando los resultados que se obtuvieron utilizando las distintas políticas de asignación que incorporadas el diseño propuesto, se puede notar que los tiempos que el sistema distribuido requiere se encuentran en el mismo rango y además mantienen cierto nivel de estabilidad aun cuando aumenta la cantidad de nodos expertos. Aunque los tiempo necesarios para los procedimientos de registro de los archivos de reglas no varía mucho entre las políticas, cada una muestra un patrón de variación particular, aunque lo único que cambia es el algoritmo de selección de los nodos expertos para asignar los archivos de reglas.

Detección de la falla de nodos de acceso

Pruebas que tienen como objetivo medir el tiempo necesario para que los nodos de acceso que se encuentran en ejecución en el sistema distribuido, detecten la falla de uno o más nodos de acceso que se encuentran en ejecución. En la tabla 8.10 se muestra el tiempo promedio que necesitan desde 1 nodo de acceso hasta 3 nodos de acceso, para detectar la falla de 1 nodo de acceso hasta 3 nodos de acceso.

	1 nodo de acceso	2 nodos de acceso	3 nodos de acceso
1 nodo de acceso fallido	4.035 s	4.096 s	4.569 s
2 nodos de acceso fallidos	6.032 s	6.556 s	7.147 s
3 nodos de acceso fallido	8.71 s	9.205 s	9.309 s

Tabla 8.10: Tiempo de detección de nodos de acceso fallidos.

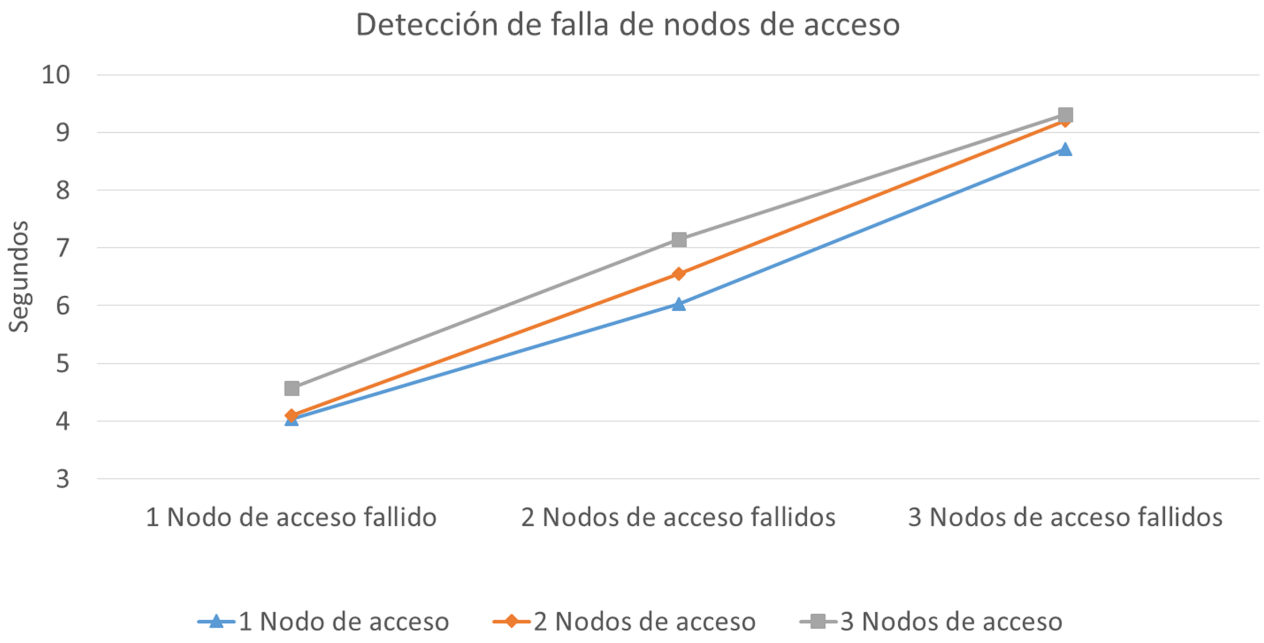

Figura 8.7: Tiempo de detección de nodos de acceso fallidos.

En el caso de la detección de la falla de nodos de acceso, se presenta la tendencia de aumentar el tiempo necesario para detectar la falla conforme aumenta la cantidad de nodos de acceso que fallan en el mismo momento; ésto puede puede ser ocasionado por el aumento de la carga de trabajo del servicio de nombres de *CORBA* conforme aumenta la cantidad de nodos en el sistema distribuido, lo cual puede ocasionar el aumento del tiempo para detectar la falla de los nodos en funcionamiento. Conforme aumenta la cantidad de nodos de acceso que fallan al mismo tiempo, aumenta la cantidad de tareas que debe realizar el nodo de acceso de mayor prioridad, pues es el único responsable de realizarlas; dichas tareas se ejecutan de forma concurrente, pero en ambientes virtuales, conforme aumenta la cantidad de nodos expertos puede que la máquina física anfitriona tenga problemas para mantener el mismo rendimiento. Aunque en entornos de producción el caso no sea tan probable, aun así es posible que el sistema distribuido requiere enfrentarse a casos de falla de la mayoría de sus nodos de acceso; tras los cuales deben agregarse más nodos de acceso, para asegurarse que el sistema funcione correctamente.

Detección de la falla de nodos expertos

Pruebas que tienen como objetivo medir el tiempo necesario para que los nodos de acceso que se encuentran en ejecución en el sistema distribuido, detecten uno o más nodos expertos del sistema que han fallado. En la tabla 8.11 se muestra el tiempo promedio que necesitan desde 1 nodo de acceso hasta 3 nodos de acceso, para detectar desde 1 nodo experto hasta 3 nodos expertos que han fallado.

	1 nodo de acceso	2 nodos de acceso	3 nodos de acceso
1 Nodo experto fallido	4.62 s	5.425 s	5.743 s
2 Nodos experto fallidos	5.771 s	6.08 s	7.147 s
3 Nodos experto fallido	6.014 s	6.524 s	7.917 s

Tabla 8.11: Tiempo de detección de nodos expertos fallidos.

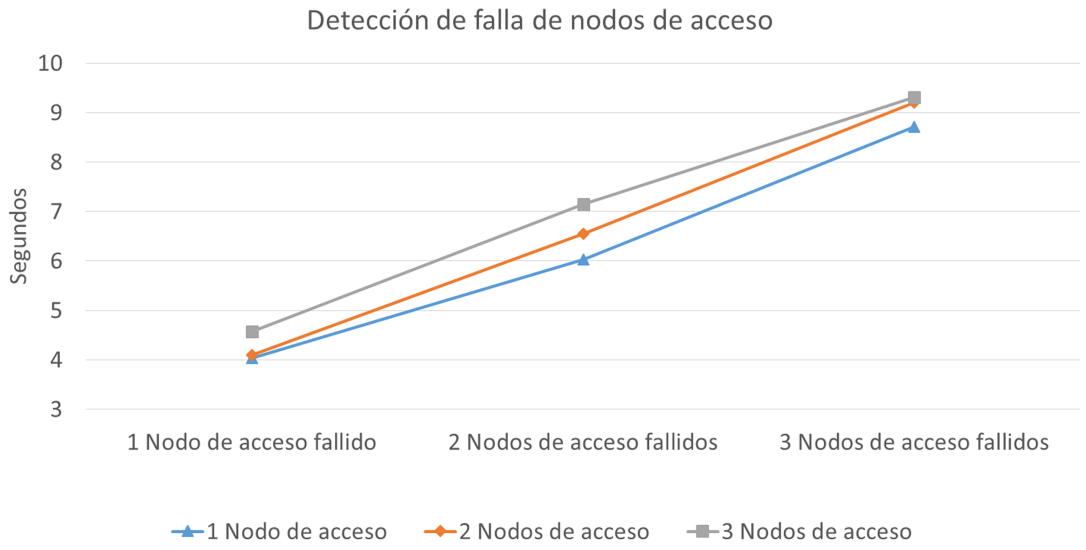

Figura 8.8: Tiempo de detección de nodos expertos fallidos.

Los procedimientos para la detección de nuevos nodos y fallas de nodos requieren más tiempo conforme aumenta la cantidad de nodos involucrados, esto se observa en el caso de prueba para la detección de nodos expertos que han fallado. Esta tendencia de incremento de tiempo necesario puede deberse al uso centralizado del

servicio de nombres de *CORBA*, entre otros aspectos, como los procedimientos que deben ejecutar los nodos para actualizar sus registros internos; cuantos más nodos fallidos deben ser detectados, más acciones se deben realizar concurrentemente.

Distribución de archivos en el sistema distribuido

Pruebas que se enfocan en verificar el funcionamiento correcto de la plataforma para la distribución de archivos, mediante la cual se pueden enviar un archivo a todos los nodos expertos presentes en el sistema distribuido, el funcionamiento de la plataforma se explica con mayor detalle en el capítulo 5. En la tabla 8.11 se muestra el tiempo promedio en las pruebas realizadas que tarda un archivo en ser distribuido a todos los nodos expertos; las pruebas se realizaron utilizando desde 3 nodos expertos hasta 9 nodos expertos. Durante el proceso de las pruebas, se comprobó que todos los archivos de hechos llegaron a todos los nodos expertos.

3 expertos	5 expertos	7 expertos	9 expertos
0.425	0.493	0.484	0.462

Tabla 8.12: Tiempo de distribución de archivos de hechos.

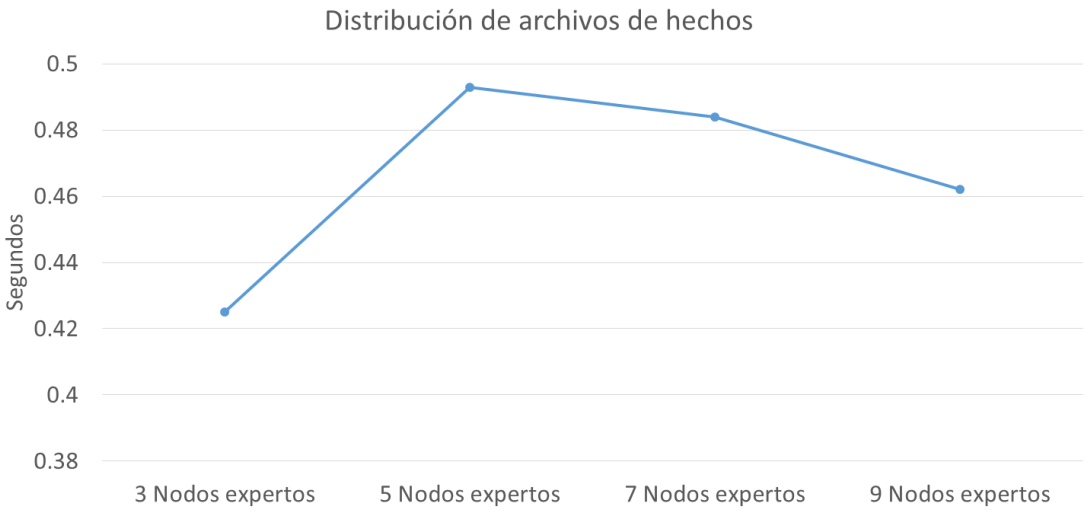

Figura 8.9: Tiempo de distribución de archivos de hechos.

Las pruebas que se realizaron a la plataforma para la distribución de archivos que se utiliza en el sistema distribuido, mostraron que el tiempo necesario para distribuir varios archivos de hechos a los nodos expertos al mismo tiempo, se mantiene en un rango estable. Los tiempos que se obtuvieron en las pruebas mostraron un aumento conforme se incrementó la cantidad de nodos expertos y posteriormente disminuye; este comportamiento puede ser ocasionado por el uso de máquinas virtuales para realizar las pruebas, por lo cual en el trabajo futuro del presente trabajo de tesis, se establece realizar pruebas con máquinas físicas.

Recuperación de archivos en nodos de acceso fallidos

Pruebas con las que se verifica la capacidad del sistema distribuido para recuperar los archivos que se encuentran en procesamiento en los nodos de acceso del sistema distribuido, los cuales han presentado una falla y no fueron capaces de volver a funcionar correctamente; el nodo de acceso de mayor prioridad en el sistema es el encargado de recuperar y procesar los archivos de los nodos de acceso fallidos, lo cual puede tener problemas cuando éste se ejecuta en una máquina virtual.

Un nodo de acceso requiere muy poco tiempo para procesar y asignar un archivo de reglas, por este motivo se agregaron métodos de registro de archivos de reglas que no terminan el registro y la transferencia del archivo de reglas a los nodos expertos; de esta forma fue posible detener nodos de acceso con archivos que no fueron procesados por completo y de esta forma realizar las pruebas.

En la tabla 8.13 se muestra el tiempo promedio que necesita el nodo de acceso de mayor prioridad después de detectar la falla de otros nodos de acceso, las pruebas se realizaron con la falla de 1 nodo de acceso hasta 3 nodos de acceso; también variando la cantidad de archivos de reglas presentes en los nodos, desde 15 archivos hasta 60 archivos en cada nodo de acceso. En las pruebas realizadas, siempre se recuperaron todos los archivos presentes en los nodos de acceso; el sistema es capaz de recuperar todo archivo que fue registrado en el sistema gestor de bases de datos.

	1 nodo de acceso	2 nodos de acceso	3 nodos de acceso
15 archivos de reglas	6.438 s	7.05 s	11.217 s
30 archivos de reglas	7.049 s	8.998 s	13.957 s
45 archivos de reglas	8.036 s	10.165 s	15.072 s
60 archivos de reglas	9.437 s	13.537 s	17.118 s

Tabla 8.13: Tiempo de recuperación de archivos en nodos de acceso fallidos.

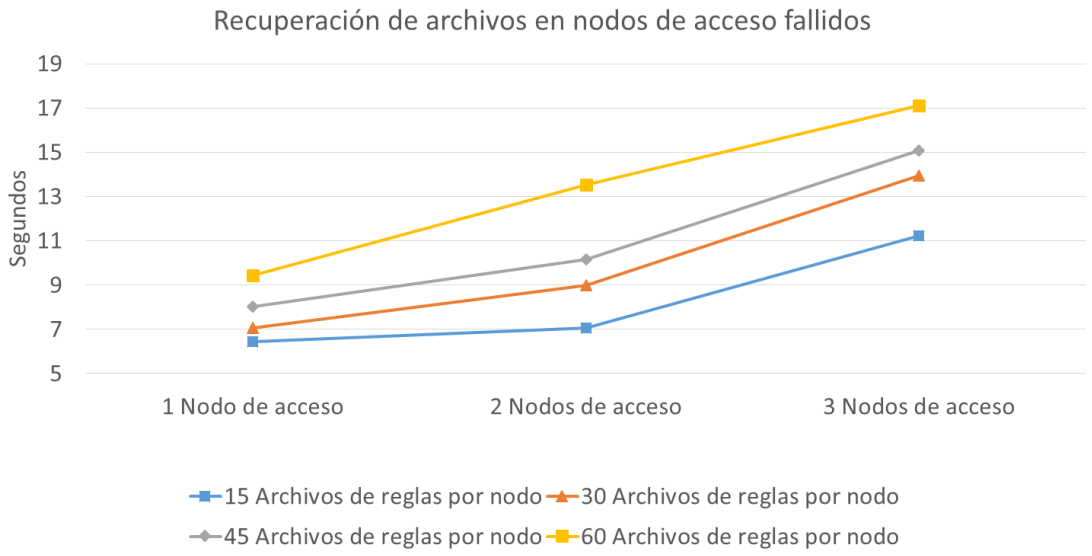

Figura 8.10: Tiempo de recuperación de archivos en nodos de acceso fallidos.

En el caso de la recuperación de archivos de nodos fallidos, el nodo de acceso debe detectar a uno o más nodos que han fallado y recuperar los archivos que no han completado su procesamiento y asignación. Lo anterior puede ocasionar que el nodo de acceso requiera transferir los archivos recuperados a los nodos expertos y estos deben crear los procesos del sistema experto, encargados de utilizar los archivos de reglas. En los resultados de estas pruebas se encontró que el tiempo que se necesita para realizar la recuperación que se describe anteriormente, se incrementa conforme aumenta la cantidad de nodos de acceso fallidos y la cantidad de archivos de reglas sin procesar que contenían.

El incremento en el tiempo de recuperación de archivos de nodos fallidos también

puede ser afectado por realizar pruebas utilizando máquinas virtuales, aunque todos los módulos dentro de los nodos del sistema distribuido realizan sus tareas de manera concurrente, al aumentar la cantidad de tareas que deben realizar, el rendimiento de todos los nodos puede verse afectado.

Un aspecto importante que debe tomar en cuenta es el acceso al sistema gestor de la base de datos, al ser servicio centralizado, su rendimiento se puede ver afectado bajo una alta carga de trabajo; por esta razón se plantea como trabajo futuro agregar el uso de un sistema gestor de bases de datos distribuidas.

Recuperación de archivos en nodos expertos fallidos

Pruebas encargadas de verificar la capacidad del sistema distribuido para recuperar los archivos de reglas que se han asignado a nodos expertos y los cuales han fallado; el nodo de acceso de mayor prioridad en el sistema es el encargado de recuperar, procesar y transferir los archivos de los nodos expertos fallidos, hacia los demás nodos expertos que continúan funcionando.

En la tabla 8.14 se muestra el tiempo promedio que necesita el nodo de acceso de mayor prioridad después de detectar la falla de nodos expertos, las pruebas se realizaron con la falla de 1 nodo experto hasta 3 nodos expertos; también variando la cantidad de archivos de reglas presentes en los nodos, desde 15 archivos hasta 60 archivos en cada nodo experto. En las pruebas realizadas, siempre se recuperaron todos los archivos presentes en los nodos expertos.

	1 experto	2 expertos	3 expertos
15 archivos de reglas	7.387 s	7.496 s	8.76 s
30 archivos de reglas	7.974 s	14.327 s	15.007 s
45 archivos de reglas	11.2 s	16.756 s	17.018 s
60 archivos de reglas	12.91 s	17.859 s	19.616 s

Tabla 8.14: Tiempo de recuperación de archivos en nodos expertos fallidos.

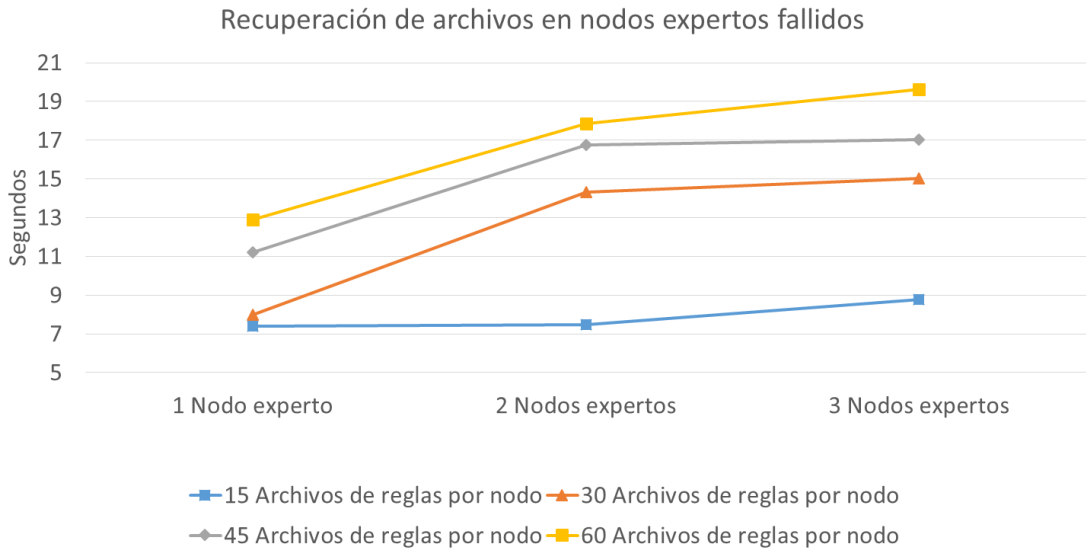

Figura 8.11: Tiempo de recuperación de archivos en nodos expertos fallidos.

El proceso de recuperación de archivos de reglas de nodos expertos fallidos es esencialmente el mismo que en el caso con nodos de acceso. Una vez que el nodo de mayor prioridad detecta la falla de uno o más nodos expertos, solicita al sistema gestor de la base de datos los archivos de reglas que encontraban asignados a los nodos expertos que fallaron; después necesita asignar los archivos a otros nodos expertos en funcionamiento. Los resultados muestran que también en este caso el tiempo aumenta conforme la cantidad de nodos expertos fallidos y la cantidad de archivos de reglas que tenían; se pueden presentar los mismos problemas de escalabilidad que en el caso con nodos de acceso.

8.4. Resumen

En este capítulo se muestran los resultados obtenidos de las pruebas que se realizaron al sistema distribuido implementado. las cuales tienen como objetivo comprobar que el sistema distribuido implementado tenga las características definidas como objetivo del presente trabajo de tesis: Escalabilidad, tolerancia a fallas y alta disponibilidad.

Las pruebas del sistema distribuido se realizaron en una máquina con un procesador de cuatro núcleos *Intel Core i7 4700MQ* con 2.4 GHz de frecuencia y 16179 MB

de memoria *RAM*; con Windows 7 64 bits como sistema operativo. En el proceso de pruebas se utilizaron 4 máquinas virtuales, las cuales se ejecutan en el sistema de virtualización *Virtualbox* [2]. Las máquinas virtuales en donde se ejecuta el sistema distribuido, utilizan *GNU/Linux Fedora* 20 de 64 bits con *Kernel* 3.16.2-200.fc20.x86_64. La máquina virtual que ejecuta el sistema gestor de bases de datos utiliza *Windows XP* de 32 bits como sistema operativo.

En este trabajo de tesis se efectuaron varias pruebas al sistema distribuido implementado, con el objetivo de verificar el correcto funcionamiento de cada una de las características y funciones del sistema; también se hicieron mediciones del tiempo promedio necesario para completar cada una las funciones que realiza el sistema. Algunas pruebas se enfocan en el proceso de inicialización del sistema distribuido, otras comprueban el funcionamiento de los métodos que registran y distribuyen los archivos en el sistema, también se realizaron pruebas de los procedimientos de recuperación de los archivos en nodos fallidos del sistema distribuido.

[2] `https://www.virtualbox.org/`

Capítulo 9

Conclusiones y trabajo futuro

Capítulo en el cual se discuten las conclusiones a las cuales se ha llegado acerca del objetivo del presente trabajo de tesis, el cual es determinar si es posible agregar a un sistema experto las siguientes características: Escalabilidad, tolerancia a fallas y alta disponibilidad; propias de un sistema distribuido, mediante la utilización de una arquitectura distribuida.

Las conclusiones permiten conocer las fortalezas y debilidades de la arquitectura distribuida propuesta, mediante las cuales se puede determinar en donde es necesario hacer cambios o correcciones al diseñó propuesto; lo conforma el trabajo futuro para la presente tesis.

9.1. Conclusiones

Un sistema distribuido posee varias características, en este trabajo de tesis nos enfocamos en las siguientes: Escalabilidad, tolerancia a fallas y alta disponibilidad; un sistema distribuido posee estas características mediante la distribución de funciones y tareas, entre todos los nodos que lo componen. La arquitectura distribuida que se propone en esta tesis, tiene como objetivo distribuir las responsabilidades del sistema en varias entidades, las cuales conforman el sistema distribuido; es importante que siempre existan varias instancias de las entidades, las cuales también se conocen como nodos.

En la implementación de la arquitectura distribuida propuesta, la cual se realizó pa-

ra comprobar el funcionamiento del diseño aquí plasmado, se utilizó el patrón *Threads Pool*; el cual permite utilizar gran cantidad de *Threads* sin tener un decremento del rendimiento al tener que crear y destruir cada vez que es necesario su utilización. En todas las entidades que componen el sistema distribuido se utiliza el patrón *Threads Pool*, pues es necesario ejecutar todas las peticiones y funciones en otros *Threads* diferentes al *Thread* principal del proceso en el sistema. Es necesario que el *Thread* principal se encuentre libre, pues es el encargado de recibir las peticiones remotas; estas pueden ser de objetos remotos *CORBA* o de otros procesos que se encuentran en el mismo nodo, los cuales se comunican mediante *Sockets* del *Middleware ZeroMQ*.

El arquitectura distribuida propuesta está diseñada para permitir que los usuarios puedan utilizar sus servicios sin importar la plataforma o sistema operativo que utilicen, por este motivo utiliza un protocolo de comunicación que es independiente de la plataforma que utilice el cliente. El sistema distribuido responde las peticiones de los usuarios a través de peticiones *REST*, las cuales son atendidas por un servidor dentro de cada entidad de acceso. La alta disponibilidad del sistema está basada en su capacidad para tener varios nodos de acceso, los cuales pueden atender las peticiones de los usuarios indistintamente; además es posible agregar al sistema distribuido más nodos de acceso en cualquier momento.

El sistema distribuido utiliza los servicios de plataforma para la distribución de archivos, la cual le permite enviar archivos a todos los nodos expertos que se encuentran en ejecución, el diseño de la plataforma se explica en el capítulo 6. La plataforma de distribución permite que un nodo de acceso pueda enviar un archivo a todos los nodos expertos sin tener que realizar el envío a cada uno por separado, esto permite que el sistema no presente problemas de escalabilidad en este ámbito. Los servicios de la plataforma de distribución se basan en clúster *RabbitMQ*, lo cual asegura que la plataforma de distribución también puede escalar conforme el sistema distribuido la haga.

Una característica importante del diseño de la arquitectura distribuida que aquí se propone, es la capacidad de utilizar una amplia variedad de sistemas expertos, los cuales requieren cumplir algunos requisitos para que sea posible su correcto funcionamiento con el sistema distribuido. Es posible utilizar con el sistema distribuido gran variedad de sistemas expertos, los cuales deben aceptar comandos a través de

un *Socket* del *Middleware ZeroMQ*. En el capítulo 7 se habla con mayor detalle del diseño del mecanismo de comunicación entre los procesos locales en las instancias del sistema distribuido y los comandos que deben entender los programas de los sistemas expertos que se deseen utilizar con el sistema distribuido.

En los resultados de las pruebas que se realizaron al sistema distribuido implementado, se observa que el sistema requiere más tiempo conforme la carga de trabajo en el sistema se incrementa; los resultados también se muestran a través de gráficas, las cuales permiten apreciar más fácilmente el comportamiento del sistema en las pruebas. En algunas ocasiones el incremento no fue lineal, esto ocurrió cuando se incrementó la cantidad de instancias entre las que se pueden distribuir las peticiones de los usuarios.

Otro aspecto que se puede apreciar en las pruebas es el incremento del tiempo necesario conforme aumenta la cantidad de nodos de acceso, los cuales se encargan de la administración de la arquitectura distribuida; cuantos más nodos de acceso se encuentren en funcionamiento, más nodos intervienen en algunos procesos como la política de selección *Round-Robin*, además las decisiones que toma un nodo de acceso se deben notificar a más nodos. Lo anterior puede repercutir en el rendimiento del sistema distribuido, pero también es necesario garantizar que siempre existan al menos 2 nodos de acceso en ejecución para el correcto funcionamiento; por eso es recomendable utilizar una mayor cantidad de nodos de acceso, pues esto incrementa la alta disponibilidad de los servicios.

El sistema distribuido es tolerable a fallas gracias a su capacidad de recuperación de los archivos presentes en los nodos que han fallado, los archivos se recuperan y se procesan para continuar su procesamiento. Es de gran importancia puntualizar que el sistema distribuido no es capaz de acceder a la memoria de conocimiento del sistema, cuando un proceso del sistema experto distribuido falla, todo su conocimiento debe estar disponible en los otros procesos del sistema experto distribuido; esto se hace para aumentar la compatibilidad con los sistemas expertos. La recuperación de los archivos de trabajo la realiza el nodo de acceso de mayor prioridad, el cual es el primero en recibir todas las notificaciones de las acciones de los demás nodos de acceso. Cuando gran cantidad de nodos fallan, el nodo de mayor prioridad puede presentar una gran cantidad de trabajo pues es el único que se encarga del proceso

de recuperación; pero al ser primer nodo en recibir las notificaciones de las acciones de los otros nodos, puede tomar decisiones más acertadas en el proceso de selección de nodos expertos.

Es importante señalar que las pruebas fueron realizadas utilizando máquinas virtuales en una máquina huésped, esto puede ocasionar que las pruebas tengan mejores tiempo en las etapas de comunicación y transmisión de información, pues éstas funcionan a través de mecanismos del sistema operativo anfitrión. Tomando en cuenta lo anterior, también puede ocurrir un incremento en el tiempo de procesamiento de las tareas en los nodos del sistema, pues en las pruebas se hacen varias peticiones a los distintos nodos de acceso al mismo tiempo; aunque sean distintas máquinas virtuales, éstas trabajan en una sola máquina física anfitriona.

Los servicios que se utilizan en los sistemas distribuidos deben ser provistos por otros sistemas distribuidos, pues en caso de la necesidad de escalamiento por parte del sistema distribuido consumidor, el sistema distribuido servidor también debería poder escalar. Por esta razón es necesario evitar utilizar servicios de sistema centralizados que no pueden escalar y que pueden dejar de funcionar, perdiendo el acceso al servicio. En la implementación del sistema distribuido se utilizaron 2 servicios centralizados, el sistema gestor de bases de datos y el servicio de nombres de *CORBA*; para la cantidad de nodos utilizados en las pruebas no presentan problemas, pero en el caso de escalar el sistema con una gran cantidad de nodos, estos servicios pueden presentar problemas.

Como conclusión final se puede decir que es posible agregar las características: Escalabilidad, tolerancia a fallas y alta disponibilidad, a un sistema experto distribuido a través de una arquitectura distribuida. Las pruebas realizadas al sistema distribuido implementado mostraron que el aumento en la carga de trabajo requiere mayor cantidad de tiempo en el procesamiento de dichas peticiones, pero no en gran medida, de forma que no sea posible el funcionamiento del sistema. La arquitectura distribuida propuesta es un primer paso para sistemas expertos distribuidos más generales, que no sean específicos de un problema o campo; aunque la arquitectura distribuida funciona correctamente para las pruebas realizadas, aun es necesario hacer pruebas con sistemas expertos distribuidos funcionales y además es necesario revisar los mecanismos de coordinación con el fin de optimizarlos y permitir que

permitan una mayor escalabilidad.

9.2. Trabajo Futuro

En esta sección se define el trabajo futuro a realizar relacionado con el presente trabajo de tesis, el cual se obtiene de las deficiencias encontradas en la arquitectura distribuida que se propone, así como de algunas actividades que no se pudieron realizar. A continuación se listan las posibles actividades futuras:

- Realizar las mismas pruebas previamente aplicadas, en un ambiente con máquinas físicas.
- Mejorar los algoritmos de coordinación en las entidades de acceso, de tal forma que no presenten problemas de escalabilidad.
- Realizar pruebas utilizando sistemas expertos y sistemas expertos distribuidos funcionales, en ambientes de pruebas reales, con gran cantidad de nodos y clientes.
- Eliminar el uso de los servicios centralizados del sistema gestor de bases de datos y el servicio de nombres, cambiando por un sistema gestor de bases de datos distribuidas y servicio de nombres de *CORBA* que funcione como un clúster.
- Investigar algunas estrategias para permitir que la arquitectura distribuida sea capaz de respaldar y recuperar la memoria de conocimiento de los sistemas expertos y sistemas expertos distribuidos que se utilicen, pero sin reducir la compatibilidad.

Bibliografía

[1] M. Sasikumar, S. Ramani, S. M. Raman, K. Anjaneyulu, and R. Chandrasekar, *A Practical Introduction to Rule Based Expert Systems*. Narosa Publishing House, 2007.

[2] I. Vlahavas, N. Bassiliades, I. Sakellariou, M. Molina, S. Ossowski, I. Futo, Z. Pasztor, J. Szeredi, I. Velbitskiy, S. Yershov, S. Golub, and I. Netesin, "System architecture of a distributed expert system for the management of a national data network," in *Artificial Intelligence: Methodology, Systems, and Applications* (F. Giunchiglia, ed.), vol. 1480 of *Lecture Notes in Computer Science*, pp. 438 – 451, Springer Berlin Heidelberg, 1998.

[3] E. Cardozo and S. Talukdar, "A distributed expert system for fault diagnosis," *Power Systems, IEEE Transactions on*, vol. 3, no. 2, pp. 641 – 646, 1988.

[4] S. Matsuda, H. Ogi, K. Nishimura, Y. Okataku, and S. Tamura, "Power system voltage control by distributed expert systems," in *Artificial Intelligence for Industrial Applications, 1988. IEEE AI '88., Proceedings of the International Workshop on*, pp. 303 – 308, 1988.

[5] J. Capella, A. Bonastre, and R. Ors, "A bridge crane advanced control system implemented by means of a distributed expert system," in *Emerging Technologies and Factory Automation, 2003. Proceedings. ETFA '03. IEEE Conference*, vol. 2, pp. 347 – 353, 2003.

[6] Y. Lirov and B. Melamed, "Distributed expert systems for network performance optimization," in *Global Telecommunications Conference, 1990, and Exhibition.*

'Communications: Connecting the Future', GLOBECOM '90., IEEE, vol. 2, pp. 1338 – 1343, 1990.

[7] S. Maria-Magdalena, L. Tiberiu, and C.Octavian, "Path finder algorithm used by distributed expert system for traffic control," in *Automation Quality and Testing Robotics (AQTR), 2012 IEEE International Conference on*, pp. 608 – 611, 2012.

[8] S. Liao, "Expert system methodologies and applications — a decade review from 1995 to 2004," *Expert Systems with Applications*, vol. 28, no. 1, pp. 93 – 103, 2005.

[9] A. Tanenbaum and M. Steen, *Distributed systems: principles and paradigms*. Pearson Education, 2nd edition ed., 2006. ISBN: 0-13-239227-5.

[10] S. Hong, Y. Kim, M. Kweon, D. Min, and S. Han, "Object-oriented real-time corba naming service an distributed environment," in *Information Networking, 1998. (ICOIN-12) Proceedings., Twelfth International Conference on*, pp. 637 – 640, 1998.

[11] C. McHale, *Corba Explained Simply*. Autopublicación, 2007.

[12] A. Campbell, G. Coulson, and M. Kounavis, "Managing complexity: Middleware explained," *IT Professional*, vol. 1, pp. 22 – 28, Sep 1999.

[13] G. Riley, *CLIPS Architecture Manual*. CLIPS Expert System Group, Jan. 1992.

[14] R. Esteller-Curto, E. Cervera, A. del Pobil, and R. Marin, "Proposal of a rest-based architecture server to control a robot," in *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on*, pp. 708 – 710, July 2012.

[15] C. Gill, J. Loyall, R. Schantz, M. Atighetchi, J. Gossett, D. Corman, and D. Schmidt, "Integrated adaptive qos management in middleware: A case study.," in *IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 276 – 285, IEEE Computer Society, 2004.

[16] H. Zhu and M. Zhou, "Methodology first and language second: A way to teach object-oriented programming," in *Companion of the 18th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, OOPSLA '03, (New York, NY, USA), pp. 140 – 147, ACM, 2003.

[17] A. Polukhin, *Boost C++ Application Development Cookbook*. Packt Publishing Ltd., 2013.

[18] T. Winstanley and P. Courvalin, "Expert systems in clinical microbiology," *Clin. Microbiol. Rev.*, vol. 24, pp. 515 – 556, July 2011.

[19] A. Kusiak, "Expert systems and optimization," *Software Engineering, IEEE Transactions on*, vol. 15, no. 8, pp. 1017 – 1020, 1989.

[20] C. Chang, T. T. Chan, and K. K. Lee, "Network switching and voltage evaluation using an expert system in ac railway systems," *Electric Power Applications, IEEE Proceedings B*, vol. 139, no. 1, pp. 1 – 12, 1992.

[21] R. J. Allwood, C. N. Cooper, and A. Taylor, "Diagnosing faults in a telecommunications network by an expert system," *Communications, Speech and Vision, IEEE Proceedings I*, vol. 137, no. 5, pp. 273 – 280, 1990.

[22] J. Bardina and R. Thirumalainambi, "Distributed web-based expert system for launch operations," in *Simulation Conference, 2005 Proceedings of the Winter*, pp. 1291 – 1297, 2005.

[23] H. Tzeng, C. Hong, C. Liao, W. Hao, and J. Chen, "A distributed expert system for stability analysis," in *Aerospace Control Systems, 1993. Proceedings. The First IEEE Regional Conference on*, pp. 672 – 676, 1993.

[24] X. Laisheng and W. Zhengxia, "Teaching resource grid system model based on multi-agent distributed expert system," in *Computational Intelligence and Software Engineering, 2009. CiSE 2009. International Conference on*, pp. 1 – 6, 2009.

[25] K. Bilal and S. Mohsin, "Muhadith: A cloud based distributed expert system for classification of ahadith," in *Frontiers of Information Technology (FIT), 2012 10th International Conference on*, pp. 73 – 78, 2012.

[26] S. Duggal and P. Popovich, "An example of management training in expert systems: Sba loan evaluation system," in *Proceedings of the 1990 ACM SIGBDP Conference on Trends and Directions in Expert Systems*, SIGBDP '90, (New York, NY, USA), pp. 588 – 618, ACM, 1990.

[27] D. Hardaway and R. Willi, "A review of barriers to expert system diffusion," in *Proceedings of the 1990 ACM SIGBDP Conference on Trends and Directions in Expert Systems*, SIGBDP '90, (New York, NY, USA), pp. 619 – 639, ACM, 1990.

[28] G. Coulouris, J. Dolimore, T. Kindberg, and G. Blair, *Distributed systems: concepts and design*. Pearson Education, 5th edition ed., 2001. ISBN: 84-7829-049-4.

[29] G. R. Andrews and F. B. Schneider, "Concepts and notations for concurrent programming," *ACM Computing Surveys*, vol. 15, pp. 3 – 43, Mar. 1983.

[30] B. Lewis and D. J. Berg, *Threads Primer: A Guide to Multithreaded Programming*. Sun BluePrints Program, 1996.

[31] S. Murugesan, "Intelligent agents on the internet and web," in *TENCON '98. 1998 IEEE Region 10 International Conference on Global Connectivity in Energy, Computer, Communication and Control*, vol. 1, pp. 97 – 102, 1998.

[32] C. Zhang, "Cooperation under uncertainty in distributed expert systems," *Artificial Intelligence*, vol. 56, no. 1, pp. 21 – 69, 1992.

[33] X. Li and M. Freedman, "Scaling ip multicast on datacenter topologies," in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '13, (New York, NY, USA), pp. 61 – 72, ACM, 2013.

[34] A. Brown and G. Wilson, "The architecture of open source applications, volume ii." Ebook, May 2012.

[35] Y. Ling, T. Mullen, and X. Lin, "Analysis of optimal thread pool size," *SIGOPS Oper. Syst. Rev.*, vol. 34, pp. 42 – 55, Apr. 2000.

[36] Pivotal Software, Inc, *AMQP Advanced Message Queuing Protocol*, 2008.

[37] M. Isenburg and J. Snoeyink, "Binary compression rates for ascii formats," in *Proceedings of the Eighth International Conference on 3D Web Technology*, Web3D '03, (New York, NY, USA), ACM, 2003.

[38] M. McCool and A. R. J. Reinders, *Structured Parallel Programming: Patterns for Efcient Computation*. Elsevier, 2012. ISBN: 978-0-12-415993-8.

[39] R. Pressman, *Software engineering: a practitioner's approach*. McGraw-Hill, seventh edition ed., 2010.

[40] L. Masinter, *Returning Values from Forms: multipart/form-data*. Xerox Corporation, 08 1998.

[41] P. Eugster, P. Felber, R. Guerraoui, and A. Kermarrec, "The many faces of publish/subscribe," *ACM Comput. Surv.*, vol. 35, pp. 114 – 131, June 2003.