UNIVERSIDAD TÉCNICA DE AMBATO

FACULTAD DE INGENIERÍA EN SISTEMAS ELECTRÓNICA E INDUSTRIAL

CARRERA DE INGENIERÍA EN SISTEMAS COMPUTACIONALES E INFORMÁTICOS

TEMA:

ANÁLISIS DE LOS PROCESOS DE DECISIÓN COMUN EN SISTEMAS DISTRIBUIDOS TOLERANTES A FALLOS A TRAVÉS DE ZOOKEEPER.

Trabajo de Graduación. Modalidad: TEMI. Trabajo Estructurado de Manera Independiente, presentado previo la obtención del título de Ingeniero en Sistemas Computacionales e Informáticos

SUBLÍNEA DE INVESTIGACIÓN:

Software distribuido Inteligente de Análisis y Control .

AUTOR: Toasa Guachi Renato Mauricio.
TUTOR: Ing. Clay Fernando Aldás Flores, Mg.

Ambato - Ecuador
Junio, 2015

APROBACIÓN DEL TUTOR

En mi calidad de Tutor del Trabajo de Investigación sobre el Tema:

"ANÁLISIS DE LOS PROCESOS DE DECISIÓN COMÚN EN SISTEMAS DISTRIBUIDOS TOLERANTES A FALLOS A TRAVÉS DE ZOOKEEPER.", del señor Renato Mauricio Toasa Guachi, estudiante de la Carrera de Ingeniería en Sistemas Computacionales e Informáticos, de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial, de la Universidad Técnica de Ambato, considero que el informe investigativo reúne los requisitos suficientes para que continúe con los trámites y consiguiente aprobación de conformidad con el Art. 16 del Capítulo II, del Reglamento de Graduación para Obtener el Título Terminal de Tercer Nivel de la Universidad Técnica de Ambato

Ambato, Junio de 2015

EL TUTOR

Ing. Clay Fernando Aldás Flores, Mg

AUTORÍA

El presente trabajo de investigación titulado: Análisis de los procesos de decisión común en sistemas distribuidos tolerantes a fallos a través de Zookeeper. Es absolutamente original, auténtico y personal, en tal virtud, el contenido, efectos legales y académicos que se desprenden del mismo son de exclusiva responsabilidad del autor.

Ambato, Junio de 2015

Renato Mauricio Toasa Guachi

CC:1804724167

APROBACIÓN COMISIÓN CALIFICADORES

La Comisión Calificadora del presente trabajo conformada por los señores docentes Ing. David Guevara A., Mg, e Ing. Renato Urvina B., Mg. revisó y aprobó el Informe Final del trabajo de graduación titulado "Análisis de los procesos de decisión común en sistemas distribuidos tolerantes a fallos a través de Zookeeper", presentado por el señor Renato Mauricio Toasa Guachi de acuerdo al Art. 17 del Reglamento de Graduación para obtener el título Terminal de tercer nivel de la Universidad Técnica de Ambato.

Ing. Vicente Morales L., Mg.

PRESIDENTE DEL TRIBUNAL

Ing. David Guevara A., Mg.	Ing. Renato Urvina B., Mg.
DOCENTE CALIFICADOR	DOCENTE CALIFICADOR

DEDICATORIA

A mis padres José Luis y Norma que han sabido formarme con valores y buenos sentimientos, los cuales me han ayudado a salir adelante en los momentos difíciles que me a tocado vivir.

A mis abuelitos que siempre han estado junto a mí brindandome su apoyo y consejos con lo que e logrado afrontar los retos que se me han presentado a lo largo de mi vida. Y a los que no estan presentes pero se que siempre me cuidan desde el cielo.

A mis tíos, a mis padrinos Hilda y Carmelo que gracias a su ayuda desinteresada y sabios consejos logre culminar con mi meta trazada.

Renato Mauricio Toasa Guachi.

AGRADECIMIENTO

En primer lugar a Dios, luego a los Ingenieros David, Clay, Ruben, a mi gran amigo Cristian que con su ayuda, conocimientos, exigencias, consejos y experiencia me permitieron seguir adelante para de esta forma lograr culminar con este difícil escalón hacia la vida profesional

Renato Mauricio Toasa Guachi

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Resumen

Apache ZooKeeper es un proyecto de software de la Apache Software Foundation, que provee un servicio de configuración centralizada y registro de nombres de código abierto para grandes sistemas distribuidos.

Mediante el análisis de Apache ZooKeeper se comprendió los servicios que ofrece zookeeper como: Consenso, Elección de Líder, los mismos que se realizan con el fin de administrar los procesos que generen los Sistemas Distribuidos que utilizan Zookeeper.

En cuanto a la configuración de los archivos necesarios para iniciar Zookeeper se demostró que no es muy compleja debido a que se detalla los puertos a utilizar, las direcciones Ip, la ubicación donde se creó los servidores virtuales, una vez configurados estos archivos se puede iniciar con Zookeeper y ver la información que muestran sus archivos de salida, para verificar que se inicio el servicio correctamente.

La programación con Zookeeper puede realizarse en dos lenguajes de programación muy conocidos y populares actualmente como son lenguaje C y java, en la presente investigación se utizó java para realizar la programación de un Monitor de Zookeeper que permitió ver las notificaciones de lo que sucede en los servidores en los cuales se esta ejecutando el servicio Zookeeper.

Una vez ejecutada la aplicación se logró apreciar el consumo de recursos que utiliza Zookeeper, los cuales fueron mínimos para Red, Memoria, Disco Duro, CPU, demostrando que Zookeeper es una herramienta de gran ayuda para los sistemas distribuidos que decidan utilizarlo.

Abstract

Apache Zookeeper is a software project of the Apache Software Foundation, which provides centralized configuration service name registration and open source for large distributed systems.

By analyzing Apache ZooKeeper could understand the services offered zookeeper as: Consensus, Leader Election, the same as is done in order to manage the processes that generate the Distributed Systems using Zookeeper.

As for the configuration files needed to start Zookeeper it proved to be not very complex because the ports to use detailed, the IP address, the location where the virtual server is created, once configured these files can be Zookeeper start with and see the information showing their output files, thus verifying that the service was started successfully.

Programming with Zookeeper likewise is not very complex, can be performed in two very popular programming languages and popular today such as C and Java language in this investigation has utilized java for programming a Monitor Zookeeper allowed to see notifications of what happens on servers which are running the Zookeeper service.

After running the application will be able to appreciate the consumption of resources used Zookeeper, which were minimal for Red, Memory, Disk, CPU Disco, showing that Zookeeper is a helpful tool for distributed systems that decide to use it.

Glosario de términos

Abstracción

La abstracción consiste en aislar un elemento de su contexto o del resto de los elementos que lo acompañan.

ACL

Una lista de control de acceso (ACL) es un concepto de seguridad informática usado para fomentar la separación de privilegios. Es una forma de determinar los permisos de acceso apropiados a un determinado objeto, dependiendo de ciertos aspectos del proceso que hace el pedido.

Algoritmo

Es un conjunto prescrito de instrucciones o reglas bien definidas, ordenadas y finitas que permite realizar una actividad mediante pasos sucesivos que no generen dudas a quien deba realizar dicha actividad. Dados un estado inicial y una entrada, siguiendo los pasos sucesivos se llega a un estado final y se obtiene una solución.

API

La interfaz de programación de aplicaciones (API) es el conjunto de subrutinas, funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción. Son usadas generalmente en las bibliotecas.

Asíncrona

Hace referencia al suceso que no tiene lugar en total correspondencia temporal con otro suceso.

Consenso

Es un servicio que permite acordar un mismo valor entre todos los equipos de la nube.

Concurrencia

Es la propiedad de los sistemas que permiten que múltiples procesos sean ejecutados al mismo tiempo, y que potencialmente puedan interactuar entre sí.

Computación en la nube

La computación en nube es un sistema informático basado en Internet y centros de datos remotos para gestionar servicios de información y aplicaciones. La computación en nube permite que los consumidores y las empresas gestionen archivos y utilicen aplicaciones sin necesidad de instalarlas en cualquier computadora con acceso a Internet.

Cuello de botella

Expresión utilizada para denominar la situación en la que la producción total se ve limitada por una de sus actividades, la de menor capacidad.

Detector de Fallos

Es una herramienta distribuida, que cada proceso puede invocar para obtener información acerca de los fallos de los procesos. Hay muchas clases de detectores de fallos en función de la calidad y el tipo de la información devuelta

Eficacia

Es la capacidad de alcanzar el efecto que espera o se desea tras la realización de una acción.

Elección de Líder

Consiste en elegir un proceso de entre varios, que se ejecutan en diferentes máquinas, en un sistema distribuido, para que actúe a modo de coordinador de los procesos y permita tomar decisiones globales.

Enumeración

Una enumeración es la expresión sucesiva de las partes de que consta un todo, ya sea total o parcialmente. También se la llama lista, es especial cuando está dispuesta verticalmente, a modo de tabla o cuadro, y relación. No debe confundirse con la numeración que es la acción de dar números a algo. A cada elemento de las enumeraciones también se lo llama ítem o apartado.

Explícito

Es aquello que expresa una cosa con claridad y determinación. Cuando algo es explícito, puede ser apreciado o advertido de manera evidente, sin lugar a dudas.

Instancia

En el paradigma de la orientación a objetos, una instancia se refiere a una realización específica de una clase o prototipo determinados.

Jerárquico

Es el criterio que permite establecer un orden de superioridad o de subordinación entre personas, instituciones o conceptos; es decir, la organización o clasificación de categorías o poderes, siguiendo un orden de importancia.

Librerías

Es un conjunto de implementaciones funcionales, codificadas en un lenguaje de programación, que ofrece una interfaz bien definida para la funcionalidad que se invoca.

Petición

Es un requerimiento o solicitud que le hace un cliente a un servidor - See more at: http://www.alegsa.com.ar/Dic/peticion.php#sthash.QtG22nZQ.dpuf

Primitivas

Un servicio está definido por un conjunto de operaciones más sencillas, llamadas primitivas. En general, se utilizan para realizar alguna acción o para informar de un

suceso ocurrido en una entidad par.

Pseudocódigo

Es una descripción de alto nivel compacta e informal del principio operativo de un programa informático u otro algoritmo.

Serializado

Consiste en un proceso de codificación de un objeto en un medio de almacenamiento (como puede ser un archivo, o un buffer de memoria) con el fin de transmitirlo a través de una conexión en red como una serie de bytes o en un formato humanamente más legible como XML o JSON, entre otros

Sesión

Es la duración de una conexión hecha desde el cliente hacie el servidor

Síncrona

Que se produce o se hace al mismo tiempo que otro hecho, fenómeno o circunstancia, en perfecta correspondencia temporal con él, o con los mismos intervalos, velocidad o período que otro hecho, fenómeno, movimiento, mecanismo.

Sistema de Archivos

Es el componente del sistema operativo encargado de administrar y facilitar el uso de las memorias periféricas, ya sean secundarias o terciarias. Sus principales funciones son la asignación de espacio a los archivos, la administración del espacio libre y del acceso a los datos resguardados.

Virtualización

Es la creación a través de software de una versión virtual de algún recurso tecnológico, como puede ser una plataforma de hardware, un sistema operativo, un dispositivo de almacenamiento u otros recursos de red.

INTRODUCCIÓN

El presente Trabajo Estructurado de Manera Independiente denominado: "ANÁLISIS DE LOS PROCESOS DE DECISIÓN COMÚN EN SISTEMAS DISTRIBUIDOS TOLERANTES A FALLOS A TRAVÉS DE ZOOKEEPER"; para el entendimiento del mismo, se lo ha dividido en los siguientes capítulos:

CAPÍTULO I denominado "EL PROBLEMA", identifíca el problema que actualmente tienen los sistemas distribuidos y como interviene Zookeeper para dar una solución acertada y fácil de implementar. Además se detalla la justificación y los objetivos que se van a cumplir en la presente investigación.

CAPÍTULO II denominado "MARCO TEÓRICO", muestra las investigaciones previas que sirven de soporte para el desarrollo de la investigación, además la información de estudios similares anteriormente realizados, asi como los aspectos conceptuales que sustentan el tema en general, y el conjunto de conceptos y fundamentos teóricos que han sido analizados en base al problema establecido.

CAPÍTULO III denominado "METODOLOGÍA", define el tipo de investigación que ha sido desarrollada, el tratamiento de los procesos que señala la modalidad de investigación, asimismo se presenta el tipo de análisis de los datos según el tipo de investigación.

CAPÍTULO IV denominado "DESARROLLO DE LA PROPUESTA", determina una extensa información sobre el tema investigado en donde se detalla conceptos, métodos y la forma de programar, además se detalla paso a paso como hacer funcionar el Servicio Zookeeper y se muesta información de los recursos utilizados por el mismo.

CAPÍTULO V denominado “CONCLUSIONES Y RECOMENDACIONES”, expone de forma clara y concisa las consideraciones mas relevantes que se han obtenido al finalizar el proyecto, además se indican recomendaciones que servirán de apoyo para el desarrollo del mismo.

CAPÍTULO 1

El problema

1.1. Tema de Investigación

Análisis de los procesos de decisión común en sistemas distribuidos tolerantes a fallos a través de Zookeeper.

1.2. Planteamiento del Problema

La coordinación distribuida en la nube es un aspecto muy importante en los servicios actuales que se ofertan a través de internet. Los principales avances y retos a resolver dentro de la coordinación distribuida en la computación en nube son: El consenso y detectores de fallos .

El Consenso: Es un servicio que permite acordar un mismo valor entre todos los equipos de la nube.

Los detectores de fallos: Estos son una herramienta distribuida, que cada proceso puede invocar para obtener información acerca de los fallos de los procesos. Hay muchas clases de detectores de fallos en función de la calidad y el tipo de la información devuelta [1].

En la actualidad ya existen varios proyectos a nivel internacional que están trabajando en el estudio e implementación de consenso y detectores de fallos en los sistemas distribuidos en la nube tradicionales. De entre ellas cabe resaltar dos:

ZooKeeper: Es un proyecto para ofrecer servicios de coordinación en sistemas distribuidos de la Apache Software Foundation. Entre estos servicios están los de consenso y detectores de fallos [2].

Megastore: Es un servicio de Google para ofrecer espacio en disco con alta disponibilidad y escalabilidad en la nube. Los accesos a este espacio en disco son coordinados de forma que se ofrezca una determinada coherencia en las lecturas y escrituras a los datos [3].

En el Ecuador la mayoría de empresas utilizan sistemas distribuidos tolerantes a fallos sin embargo hay un desconocimiento de los procesos, servicios y algoritmos que intervienen en estos sistemas ya que al ser muy complejos no existe mucha información sobre el tema.

En la Unidad Operativa de Investigaciones de la FISEI se requiere analizar como ZooKeeper realiza los procesos de detección de fallos, y como se genera consenso dentro de un entorno de red distribuida controlado y virtualizado, esta información servira de base para la realización del Proyecto de Investigación denominado: " Servicios de Coordinación en la Nube cuando los Elementos Intervinientes son Anónimos.".

Los algoritmos que intervienen en los procesos mencionados son complejos de entender y ejecutar, por tanto se debe recurrir a una herramienta que facilite el trabajo, es aquí donde interviene Zookeeper ya que facilitará el análisis de estos algoritmos.

Zookeeper permite el acceso al código fuente por tanto facilita su estudio y permitirá analizar cómo se aplican los algoritmos de consenso. Al estudiar acerca de los procesos y mecanismos de detección de fallos y consenso permitirá resolver el problema objeto de este estudio.

1.3. Delimitación

Área Académica: Software

Línea de Investigación: Arquitectura y Tecnología de redes.

Sublínea de Investigación: Software distribuido Inteligente de Análisis y Control.

Delimitación Espacial: Unidad Operativa de Investigación y Desarrollo de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial.

Delimitación Temporal: El desarrollo de este trabajo durará 6 meses a partir de la fecha de aprobación por parte de Consejo Directivo.

1.4. Justificación

La realización de este proyecto se da por la necesidad de analizar el funcionamiento de la herramienta zookeeper y entender cómo se producen los diferentes procesos de control de fallos y consenso, para que sea un punto de partida en la intención de generar nuevos algoritmos para agregar nuevas funcionalidades, las mismas que serán realizadas en estudios y proyectos de investigaciones posteriores.

Además este proyecto permite conocer los sistemas distribuidos, como estos interactúan entre sí.

La Unidad Operativa de Ia Dirección de Investigación y Desarrollo de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial prestará las facilidades necesarias para la realización del presente proyecto, el mismo que aportará con una base de teórica para el desarrollo del Proyecto de Investigación llamado: "Servicios de Coordinación en la Nube cuando los Elementos Intervinientes son Anónimos."

1.5. Objetivos

1.5.1. General

- Analizar los procesos de decisión común en un sistema distribuido tolerante a fallos a través del sistema de código abierto Zookeeper.

1.5.2. Específicos

- Describir las funcionalidades y características técnicas de la aplicación en el sistema de código abierto Zookeeper.
- Determinar los métodos fundamentales para controlar los procesos de decisión común en Zookeeper.
- Realizar una simulación de funcionamiento de Zookeeper en un entorno distribuido controlado.

CAPÍTULO 2

Marco Teórico

2.1. Antecendentes Investigativos

En la actualidad existen grandes empresas que utilizan Zookeeper para administrar parte de sus procesos críticos como yahoo que utiliza utiliza ZooKeeper para tareas como: maestro elección, detección de colisión, y el almacenamiento de metadatos, facebook que utiliza Zookeeper para: el almacenamiento de registro de datos a travez de múltiples máquinas, para conmutación por error y para detección de servicios[4], además existen varias Universidades de prestigio a nivel mundial que están trabajando en el estudio e implementación de algoritmos de consenso y detectores de fallos en los sistemas distribuidos en la nube tradicionales[5].

Revisados los repositorios de las Universidades más importantes del país no se encontró trabajos realizados que se relacionen con el tema de Cloud Computing y Zookeeper.

A nivel mundial se esta trabajando en la Universidad Politécnica de Madrid en un proyecto en el cuál se programa un algoritmo de consenso y detección de fallos, que son los servicios que ofrece Zookeeper a los Sistemas Distribuidos[5].

Además de esto se ha encontrado artículos científicos a nivel internacional que tienen relación al trabajo planteado, tales como:

ZooKeeper Wait-free coordination for Internet-scale systems.

Este trabajo describe a Zookeeper como un servicio de coordinación de procesos en aplicaciones distribuidas. Además se menciona algunas primitivas de configuración y la estructura de Zookeeper [6].

The Zookeeper's Problem.

Aquí se detalla algunos algoritmos de Zookeeper para la resolución de problemas en aplicaciones distribuidas [7].

BookKeeper Getting Started Guide.

Menciona la parte de la configuración de Bookeeper y Zookeeper, bookeeper es

una distribución antigua de Zookeeper [8].

Building an Impenetrable ZooKeeper.

Muestra algunas formas de configuración segura de Zookeeper, tomando en cuenta los parámetros para trabajar con Zookeeper [9].

Hay que recalcar que no existe mucha información acerca del trabajo propuesto debido a que se trata de un tema nuevo que se está investigando actualmente a nivel nacional e internacional.

2.2. Fundamentación Teórica

2.2.1. Sistemas Distribuidos

Un sistema distribuido es un programa que ejecuta sobre un conjunto de ordenadores conectados por líneas de comunicación. Una característica importante de estos sistemas es que pueden tolerar fallos de alguno de los ordenadores y seguir cumpliendo la función que tenían encomendada, todo ello manteniendo un estado consistente [10].

Este tipo de arquitectura permite obtener sistemas con mayor capacidad de procesamiento, con una tecnología determinada. Esta tecnología tendría que ser mucho más cara, si nos decidiéramos a utilizar un sistema monoprocesador con la misma capacidad de procesamiento. Por tanto, la razón del auge de estos sistemas es básicamente económica [10].

Aparte de las razones económicas, estos sistemas tienen otras ventajas frente a los sistemas monoprocesador, como son su versatilidad para crecer y su tolerancia a fallos. La primera ventaja se refiere a la capacidad de crecer sin cambiar de hardware y la segunda ventaja permite a estos sistemas seguir ejecutando en un estado consistente, a pesar del fallo de alguno de los procesadores [10].

El inconveniente que tienen estos sistemas es la complejidad de su software. Este no resulta tan fácil de definir. En cualquier caso debe resolver problemas como: ¿Qué código va a ejecutar en cada nodo de la red?, ¿Cómo se van a comunicar y sincronizar dichas ejecuciones?, ¿Cómo se va a reconfigurar el sistema?, ¿Cómo se va a conservar la consistencia del estado del proceso en caso de fallo? .

Los diseñadores de sistemas operativos distribuidos y de lenguajes de programación distribuidos deben responder a estas preguntas, ofreciendo al usuario del sistema operativo o al programador de aplicaciones distribuidas las herramientas de software necesarias que permitan abstraer los problemas de implementación[10].

2.2.2. Tolerancia a Fallos

Los fallos de comunicaciones se refieren a los fallos que se pueden producir en las líneas de comunicación que unen los diferentes procesadores, y pueden ser temporales o permanentes. Ha sido ampliamente estudiado cómo tolerar estos fallos. Para resolver los fallos temporales se han propuesto protocolos normalmente basados en reintentos y para resolver los fallos permanentes se usan protocolos de cambio de encaminamiento de los mensajes por la red. En ambos casos, estos protocolos tratan de solucionar el problema de forma transparente al usuario y, si no pueden solucionarlo, se encargan de avisar al usuario del mismo [10].

Los fallos de diseño de software no son privativos de los sistemas distribuidos, ya que se refieren a fallos introducidos por el programador, al no coincidir lo programado con lo especificado. Para resolver estos fallos se han estudiado varias aproximaciones. Una es la verificación de programas mediante aserciones lógicas. Otra consiste en hacer n versiones de un mismo módulo software que cumplan la misma especificación, pero con código diferente, y durante la ejecución de los mismos se comparan y votan los resultados de estos módulos, considerándose como buen resultado el de la mayoría. Esto permite tratar tanto fallos permanentes como transitorios. La tercera y última aproximación se lleva a cabo mediante bloques de recuperación y consiste en insertar un test de aceptación del resultado, después de la ejecución de un módulo software, y en caso de no pasar el test volver a realizar la ejecución, pero con otro módulo alternativo que cumpla la misma especificación, volviendo a repetir el proceso de test y recuperación [10].

Los fallos de procesador son los que se presentan por algún fallo de hardware, bien en el procesador, bien en su bus, o bien en su memoria. Normalmente, suelen provocar la parada del procesador, con la consecuente pérdida, tanto de los datos almacenados en memoria volátil, como del estado del procesamiento[10].

2.2.3. Zookeeper

Apache ZooKeeper es un proyecto de software de la Apache Software Foundation, que provee un servicio de configuración centralizada y registro de nombres de código abierto para grandes sistemas distribuidos. ZooKeeper es un subproyecto de Hadoop[2].

La arquitectura de ZooKeeper soporta alta disponibilidad a través de servicios redundantes. Los clientes pueden así preguntar a otro maestro ZooKeeper si el primero falla al responder. Los nodos ZooKeeper guardan sus datos en un espacio de nombres jerárquico, como hace un sistema de archivos o una estructura de datos. Los clientes pueden leer y escribir desde y hacia los nodos y de esta forma tienen

un servicio de configuración compartido. ZooKeeper es usado por varias compañías, incluyendo Rackspace y Yahoo, así como sistemas de búsqueda empresarial open source como Solr[2].

Entre los servicios que ofreece Zookeeper están los de consenso y eleccíon de lider [2].

ZooKeeper tiene como objetivo proporcionar un núcleo de rendimiento simple y alto para la construcción de primitivas de coordinación más complejas en el cliente[6].

La interfaz de ZooKeeper permite un alto rendimiento de la implementación del servicio. ZooKeeper ofrece una garantía por cliente ya que las solicitudes que recibe se ejecutan de forma FIFO primero en entrar, primero en Salir. Esto permite la implementación de un alto rendimiento ya que las solicitudes se procesan en servidores locales[6] .

Las Aplicaciones distribuidas a gran escala requieren diferentes formas de coordinación. La configuración es una de las más formas básicas de coordinación. En su forma más simple, la configuración es sólo una lista de parámetros operativos para los procesos del sistema, mientras que los sistemas más sofisticados tienen parámetros de configuración dinámicos. Pertenencia a grupos y elección de líder también son comunes en distribución de sistemas, a menudo los procesos necesitan saber qué otros procesos están vivos y de lo que estos procesos están a cargo [6].

Un enfoque de la coordinación es el desarrollo de los servicios para cada una de las diferentes necesidades de coordinación. Por ejemplo Amazon Simple Queue Service [11] se centra específicamente en las colas. Otros servicios han sido desarrollados específicamente para elección de líder [12] y la configuración [13].

El diseño del servicio de coordinación, se aleja de la implementación de las primitivas específicas sobre el lado del servidor, y en su lugar se opta por exponer una API que permite a los desarrolladores de aplicaciones implementar sus propias primitivas. Esta elección condujo a la aplicación de un núcleo de coordinación que permite nuevas primitivas sin necesidad de cambios en el núcleo de servicios. Este enfoque permite múltiples formas de coordinación adaptados a los requisitos de las aplicaciones, en lugar de restringir a los desarrolladores a un conjunto fijo de primitivas[6].

El diseño de la API de ZooKeeper, se aleja de las primitivas de bloqueo, las primitivas de bloqueo para un servicio de coordinación pueden causar, entre otros problemas, que los clientes defectuosos afecten negativamente en el rendimiento de los clientes más rápidos. La implementación del servicio en sí se vuelve más complicada si el procesamiento de las solicitudes depende de las respuestas y la detección de fallos de otros clientes. Zookeeper, por lo tanto, implementa una API

que manipula los objetos de datos simples organizados jerárquicamente como en sistemas de archivos. De hecho, la API de ZooKeeper se asemeja a la de cualquier otro sistema de archivos [6].

2.2.4. Servicios que Ofrece Zookeeper

Entre los servicios que utiliza Zookeeper para realizar sus procesos de alto rendimiento estan consenso y elección de líder. Estos servicios se utilizan para poder administrar los procesos de una forma ordenada, además permiten que se muestren notificaciones acerca de lo que sucede en los servidores[2].

2.2.5. Consenso

El consenso permite a los procesos llegar a una decisión común a partir de valores iniciales y a pesar de fallos, es un problema fundamental en computación distribuida tolerante a fallos,

En un entorno distribuido, los procesos tienen dificultades para ponerse de acuerdo en un valor cuando uno o más procesos han propuesto cuál debería ser:

- **Protocolos de acuerdo específicos**
  - Exclusión mutua: consenso en quién entra en la SC.
  - Elección: consenso en quién es el nuevo coordinador.
  - Multidifusión ordenada: consenso en el orden de entrega.
- **Protocolos de acuerdo genéricos**
  - Problemática.
  - Condiciones que deben cumplir.
  - Algoritmos .
- **La comunicación es fiable pero los procesos pueden fallar.**
  - Por caída n.
  - Por fallos arbitrarios (bizantinos).
- **Hasta f de los N procesos pueden fallar**
  - Se debe llegar a un consenso incluso en este caso
- **Los procesos no firman sus mensajes**
  - Esto puede ser relevante en algunos casos.

Condiciones para el consenso:

- **Terminación.**
  - Cada proceso correcto ha de fijar su variable de decisión
- **Acuerdo.**
  - El valor de decisión de los procesos correctos es el mismo.
- **Integridad.**
  - Si todos los procesos correctos han propuesto el mismo valor, entonces cualquier proceso correcto en el estado decidido ha elegido dicho valor.

2.2.6. Elección de Líder

Se trata de un proceso en el cual se va a elegir un proceso único para que tome un determinado rol[14].

- **Consideraciones**
  - Un proceso convoca elecciones cuando lleva a cabo una acción que inicia el algoritmo de elección.
  - Puede haber N elecciones concurrentes.
  - Un proceso siempre tiene uno de estos dos roles:
    - Participante : comprometido en una ejecución del algoritmo .
    - No participante : no comprometido en ninguna ejecución .
  - El proceso elegido debe ser único, incluso en elecciones concurrentes .
  - Identificadores.
    - Todos los procesos tienen un identificador .
    - El proceso elegido es aquél de mayor identificador.
  - Variable elegido .
    - Cada proceso p i mantiene una variable e i que contiene el identificador del proceso elegido .
    - Cuando el proceso se convierta en participante, fija la variable al valor especial $\perp$, indicando que no hay consenso todavía .

- **Rendimiento**
  - Ancho de banda: proporcional al número de mensajes enviados.
  - Tiempo de ronda (turnaround): tiempo pasado desde que se convocan elecciones hasta que se elige un proceso .

2.2.7. Descripción General del Servicio

ZooKeeper ofrece a sus clientes la abstracción de un conjunto de nodos de datos llamados znodes, organizados de acuerdo a un espacio de nombres jerárquico. Los znodes en esta jerarquía son los datos que manipulan los clientes a través del Api de ZooKeeper. Los espacios de nombres jerárquicos se utilizan comúnmente en sistemas de archivos. Es una forma de organización de los datos, ya que los usuarios están acostumbrados a esta abstracción y permite una mejor organización de la aplicación de metadatos [6].

Para referirse a un znode dado, usamos la notación estándar de UNIX para archivos de rutas del sistema. Por ejemplo, se utiliza /A /B /C para saber la ruta del sistema en el que se encuentra C [6].

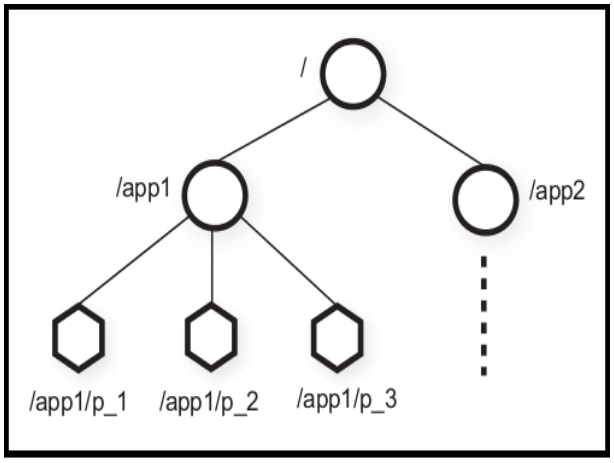


Figura 2.1: Espacio de nombres Jerárquicos.
Fuente: “Apache Zookeeper”, http://zookeeper.apache.org/

Hay dos tipos de znodes que un cliente puede crear:

Regulares: Los clientes manipulan znodes regulares por creación y eliminación de forma explícita.

Efímeras: Los clientes crean los znodes o bien eliminarlos de forma explícita, o dejar que el sistema los elimine automáticamente cuando la sesión iniciada se termine deliberadamente o debido a un fallo .

ZooKeeper implementa vistas para permitir a los clientes recibir notificaciones oportunas de los cambios .

2.2.8. Garantías de Zookeeper

Escritura consistente: todos los pedidos de actualización del estado de ZooKeeper son serializados y respetan el orden de precedencia[6] .

Orden de cliente FIFO: todos los pedidos de un cliente son ejecutados en el orden en el que fueron enviados .

Disponibilidad: el servicio está disponible mientras haya una mayoría de servidores activa y comunicada .

Durabilidad: los cambios persisten, a pesar de fallas, si en algún momento se recupera la mayoría de servidores.

2.2.9. API Zookeeper

EL API de Zookeeper ofrece varios métodos para poder trabajar, entre los mas importantes estan[6]:

- Create: Permite la creción de los nodos de Zookeeper.
- Delete: Elimina los nodos que no se utilicen
- Exists: Verifica si el nodo existe para evitar nombres duplicados.
- GedData: Obtiene información de los nodos de Zookeeper.
- SetData: Envia la información de los nodos de Zookeeper.

Mas adelante se muestrá más métodos que ofrece el API.

2.2.10. Programación con Zookeeper

Apache ZooKeeper está implementado en Java y su API nativa es también Java. ZooKeeper también proporciona una API de lenguaje C, y la distribución ofrece módulos contrib para Perl, Python y clientes REST. Zookeeper se puede representar de dos formas[15]:

1. **Síncrona**: Puede realizar peticiones constantes hacia el servidor, lo que no es aconsejable ya que puede ocacionar colisiones.

2. **Asíncrona**: Accede como un servicio remoto para evitar peticiones constantes hacia el servidor por eso se a optado en utilizar observadores y notificaciones.

La que se utilice depende de la situación. Por ejemplo, se puede optar por la API de Java asíncrona si se implementa una aplicación Java para procesar un gran número de znodes independientemente uno de otro; en este caso, se podría hacer un buen uso de la API asíncrona para lanzar simultáneamente todas las tareas independientes en paralelo. Por otro lado, si está implementando tareas sencillas que realizan operaciones secuenciales en ZooKeeper, la API síncrona es más fácil de usar y puede ser una mejor opción en estos casos[15].

A continuación se muestra como programar con Zookeeper:[15]

- **Conexión con el servidor**

El siguiente método muestra los que parámetros se necesita para poder conectarse hacia el servidor.

```
1  public ZooKeeper connect(String hosts, int sessionTimeout)
2                                     throws IOException,
                                          InterruptedException {
3          final CountDownLatch connectedSignal = new CountDownLatch(1);
4          ZooKeeper zk = new ZooKeeper(hosts, sessionTimeout, new Watcher
               () {
5                  @Override
6                  public void process(WatchedEvent event) {
7                          if (event.getState() == Watcher.Event.
                               KeeperState.SyncConnected) {
8                  connectedSignal.countDown();
9            }
10         }
11         });
12         connectedSignal.await();
13         return zk;
14 }
```

- **Crear un grupo de znodes.**

Se realiza la creación de grupos de znodes para poder identificar que acción van a realizar en la aplicación.

```
1  public void createGroup(String groupName)
2                  throws KeeperException, InterruptedException {
3                  String path = "/" + groupName;
4                  zk.create(path,
```

```
5                           null /* data */,
6                           ZooDefs.Ids.OPEN_ACL_UNSAFE,
7                           CreateMode.PERSISTENT);
8   }
```

- **Crear znodes miembros de grupo**

Es la asignación de los nodos a un grupo para realizar acciones definidas.

```
1   public String joinGroup(String groupName, String memberName, byte[]
        data)
2                   throws KeeperException, InterruptedException {
3                   String path = "/" + groupName +
4                                                   "/" + memberName + "-";
5                   String createdPath = zk.create(path,data,ZooDefs.Ids.
                        OPEN_ACL_UNSAFE,CreateMode.EPHEMERAL_SEQUENTIAL);
6                   return createdPath;
7   }
```

- **Listar miembros de un grupo de forma indefinida**

Muestra los los nodos y el grupo al que pertenecen.

```
1   public class ListGroupForever {
2     private ZooKeeper zooKeeper;
3     private Semaphore semaphore = new Semaphore(1);
4
5     public ListGroupForever(ZooKeeper zooKeeper) {
6       this.zooKeeper = zooKeeper;
7     }
8
9     public static void main(String[] args) throws Exception {
10      ZooKeeper zk = new ConnectionHelper().connect(args[0]);
11      new ListGroupForever(zk).listForever(args[1]);
12    }
13
14    public void listForever(String groupName)
15            throws KeeperException, InterruptedException {
16      semaphore.acquire();
17      while (true) {
18        list(groupName);
19        semaphore.acquire();
20      }
21    }
22
23    private void list(String groupName)
```

```
24            throws KeeperException, InterruptedException {
25      String path = "/" + groupName;
26      List<String> children = zooKeeper.getChildren(path, new Watcher() {
27        @Override
28        public void process(WatchedEvent event) {
29          if (event.getType() == Event.EventType.NodeChildrenChanged) {
30            semaphore.release();
31          }
32        }
33      });
34      if (children.isEmpty()) {
35        System.out.printf("No members in group %s\n", groupName);
36        return;
37      }
38      Collections.sort(children);
39      System.out.println(children);
40      System.out.println("———————————————————");
41  }
```

- **Eliminar un Grupo de znodes**

Permite eliminar los nodos que ya no se utilicen.

```
1  public void delete(String groupName)
2          throws KeeperException, InterruptedException {
3    String path = "/" + groupName;
4    try {
5      List<String> children = zk.getChildren(path, false);
6      for (String child : children) {
7        zk.delete(path + "/" + child, -1);
8      }
9      zk.delete(path, -1);
10   }
11   catch (KeeperException.NoNodeException e) {
12     System.out.printf("Group %s does not exist\n", groupName);
13   }
14 }
```

2.2.11. Rendimiento del Ordenador

El rendimiento de un equipo de cómputo es a menudo la clave para medir la eficacia de un sistema de hardware y software. Aunque el problema de evaluar el rendimiento global de un equipo de computo es aún un poco intratable, se puede ganar mucho con la obtenión de datos en la medición y evaluación de diversos aspectos de las actividades internas que realiza un equipo de cómputo[16].

El rendimiento expresa la manera o la eficiencia con que un sistema de computación cumple sus metas, también se podría decir que es una cantidad relativa mas no absoluta pero suele hablarse de medidas absolutas de rendimientos como por ejemplo el número de trabajos atendidos por unidad de tiempo[16].

2.2.12. Herramientas de Medición de Recursos

Existen varias herramientas para poder medir el consumo de recursos del ordenador, tales como:

- **SiSoft Sandra**: Herramienta utilizada para medir el rendimiento de la memoria RAM(Random Access Memory)[17].
- **OverClock Checking Tool:** Permite medir el consumo de CPU(Central Processing Unit) y memoria RAM[17].
- **HD-Tune:** Mide el consumo de disco utilizado[17].
- **SAR(System Activity Report**): Es una herramienta completa que permite medir el consumo de Red, CPU, Disco, Memoria y obtener datos para su posterior análisis[18].

Herramienta de Medición	RED	CPU	DISCO	MEMORIA
SiSoft Sandra				X
OverClock Checking Toll		X		X
HD-Tune			X	
SAR	X	X	X	X

Tabla 2.1: Comparación de Herramientas de Medición de Recursos

Elaborado por: Renato Toasa

La herramienta seleccionada es SAR, debido a que permite una medición completa de los recursos del computador de una forma rápida.

2.2.13. Herramienta SAR

El nombre SAR proviene de las siglas de "System Activity Report" (informe de la actividad del sistema). En Linux se encuentra normalmente en el paquete sysstat. El paquete sysstat incluye programas y scripts para recopilar y mostrar información sobre el rendimiento del sistema, generando informes detallados. Este conjunto de

programas puede resultar de mucha utilidad a la hora de detectar cuellos de botella y para hacernos una idea de cómo se utiliza el sistema a lo largo del día[18].

La utilización de la CPU, intercambio de memoria, y las estadísticas de transferencia de red son parte de la generación de datos que permite sysstat, asimismo los informes originados en base al rendimiento pueden ser utilizados para localizar cuellos de botella y múltiples estadísticas de los servidores. Sar recopila, anuncia, y guarda la información de la actividad del sistema para que pueda ser visualizada[18].

Sar permite establecer mecanismos de monitorización definidos por el administrador, obteniendo valores de utilización de las áreas de CPU (porcentaje de uso), I/0 RED (transmisiones por segundo de bits), RAM (Gigabytes utilizados), en intervalos de tiempo definidos según sean requeridos. Sar presenta la posibilidad de recoger periódicamente datos y guardarlos en ficheros de administración /etc/tmp/-sar.data.txt, mostrando dentro de un archivo de texto plano los datos de rendimiento actuales de todas la áreas definidas. Con el objetivo de hacer aún más útiles los datos recogidos, Sar nos permite exportar los datos a un formato legible como un editor de texto (con puntos y comas como separadores en la generación de datos)[19].

2.2.14. Herramientas para Gráficar

Una vez obtenidos los datos del consmo de recursos del computador es necesario realizar gráficos estadísticos para poder interpretar estos datos, existen varias herramientas para realizar graficar, tales como:

Chart Chooser: Es un servicio online gratis que permite utilizar 17 diferentes tipos de gráficos para Excel y Powerpoint.

Matplotlib: Permite realizar gráficos estadisticos mediante la programación de scripts en Phyton[20].

Gnuplot: Herramienta que permite graficar funciones en 2 y 3 dimensiones, utilizando archivos de texto, tablas de datos o programando fórmulas en líneas de comandos[21].

Herramienta para Graficar	Forma de Graficar
Chart Chosser	Datos de exel o powerpoint
Matplotlib	Sripts en Phyton
Gnuplot	Archivos de texto

Tabla 2.2: Comparación de Herramientas para Graficar

Elaborado por: Renato Toasa

Para realizar las gráficas se selecciono Gnuplot, ya que los datos obtenidos son en archivos de texto y gnuplot procesa este tipo de archivos.

2.2.15. Herramienta GNUPLOT

Gnuplot es un programa en línea de comandos que permite dibujar gráficas de funciones en 2 y 3 dimensiones a través de las fórmulas que las definen. También puede dibujar gráficos usando una tabla de coordenadas (en formato sólo texto) creadas con cualquier programa.

El software tiene copyright pero se distribuye libremente y está disponible para UNIX, Linux, IBM OS/2, MS Windows, MSDOS, Macintosh, VMS, Atari y muchas otras plataformas.

Originalmente estaba destinado a científicos y estudiantes para permitirles visualizar gráficamente funciones matemáticas o tablas de datos. Hace su trabajo muy bien y es utilizado por otras herramientas, entre las que se encuentran Maxima y Octave, para dibujar gráficas evitandoles tener que desarrollar su propio motor de dibujo (un ejemplo paradigmático de software libre y cooperativo)[21].

Además de dibujar la gráfica en pantalla puede guardarla en multitud de formatos entre los que se encuentran los usuales, como jpg, png, pdf, svg; y otros, menos usuales, pero muy interesantes para los usuarios LaTeX como metafont, eps, pstricks, picture[21].

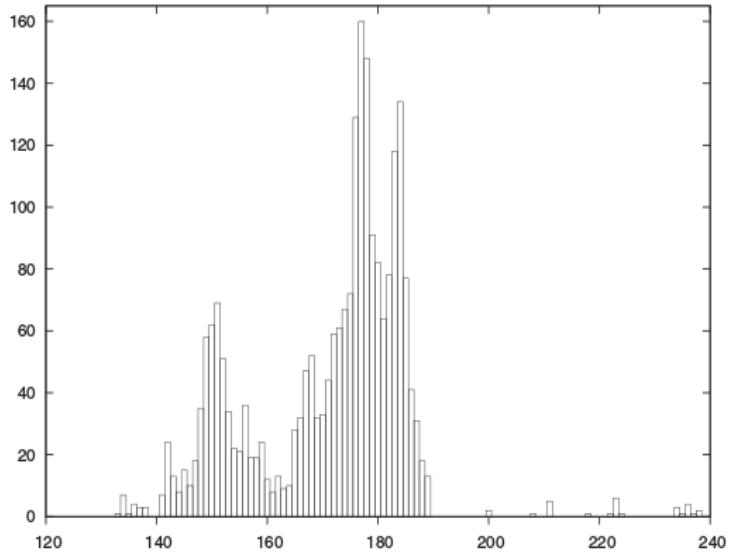


Figura 2.2: Gráfico generado con GNUPLOT
Fuente: Philipp K Janert, "Gnuplot in Action Understanding Data with Graphs", Manning Publications Co. (2010), 398.

2.3. Propuesta de Solución

Con la realización de este proyecto se va a establecer y determinar los procesos de decisión común en un sistema distribuido tolerante a fallos a través del sistema de código abierto Zookeeper de esta manera agregar nuevas funcionalidades que beneficiaran a los sistemas distribuidos tolerantes a fallos.

CAPÍTULO 3

Metodología

El presente trabajo tendrá un enfoque de Investigación Aplicada(I) porque se orientará a la obtención de nuevos conocimientos y su aplicación para la solución a problemas o interrogantes de carácter científico con el tema relacionado a Sistemas Distribuidos Tolerantes a Fallos y los servicios que actúan en este complejo proceso.

3.1. Modalidad Básica de la investigación

Investigación aplicada

Se realizará una investigación aplicada ya que se empleará lo aprendido en todas las áreas de la carrera para lograr el desarrollo correcto de la aplicación.

Investigación Bibliográfica - Documental

Se realizará una investigación bibliográfica - documental para poder obtener información más profunda con respecto a problemas similares, de esta manera se recopilará información valiosa que servirá como sustento científico para el desarrollo de la aplicación de consulta de información.

Investigación de campo

Se realizará una investigación de campo para obtener información en el lugar de los hechos y estudiar la situación del problema con sus causas y efectos.

3.2. Recolección de información

En el siguiente proyecto se realizó la recolección de información mediante la búsqueda en revistas, artículos y libros científicos sobre los siguientes temas: Sistemas distribuidos, Tolerancia a fallos, Consenso, Elección de Líder, Zookeeper, Programación con Zookeeper.

3.3. Procesamiento y análisis de datos

Lo primero que se realizará al recopilar la información, será analizar los resultados obtenidos con relación al problema ya planteado y seleccionar los que se requiere para el desarrollo del proyecto y así poder cumplir con los objetivos planteados anteriormente.

Para efectos de medición de resultados una vez terminada la simulación de Zookeeper se realizó lo siguiente:

1. Instalar 2 Sistemas operativos en un entorno virtual con las mismas características 512 RAM, 10 GB en disco.
   a) Fedora 20
   b) Centos 7
2. Montar Zookeeper en los 2 servidores.
3. En los 2 equipos iniciar los servidores de Zookeeper uno a continuación de otro verificando el correcto funcionamiento de los mismos.
4. Iniciar el monitoreo de los servidores mediante la aplicación programada.
5. Ejecutar el archivo en el cual se especifíca que parámetros se va a medir y se obtiene la media de los datos obtenidos.
6. El tiempo de toma de datos es de 1 minuto por cada toma.
7. Realizar 50 tomas cuando los servidores funcionaban correctamente, y otras 50 tomas cuando sufrían fallas.
8. Se obtiene 200 archivos que contenian el uso de recursos y la media de estos datos.
9. Con todos los valores de media se calcula nuevamente la media de estos valores determinando una media general para cada recurso medido.
10. Se obtiene 5 valores por cada sistema operativo utilizado.
11. Con los valores obtenidos se realiza las gráficas en donde se muestra el consumo de cada recurso en cada sistema operativo utilizado.

Para cada métrica (memoria, CPU, red y disco) se generan archivos con los rendimientos de los cuales para la memoria se toma % de memoria usada, para la CPU se toma % de utilización, para la red se toma la tasa de KB recibidos y la tasa de KB transmitidos, y por último para el disco el % de utilización.

3.4. Desarrollo del Proyecto

- Describir las funcionalidades y características técnicas de la aplicación en el sistema de código abierto Zookeeper.
- Determinar los métodos fundamentales encargados en controlar los procesos de decisión común en Zookeeper.
- Realizar una simulación de funcionamiento de Zookeeper en un entorno distribuido controlado.

CAPÍTULO 4

Desarrollo de la propuesta

Describir las funcionalidades y características técnicas de la aplicación en el sistema de código abierto Zookeeper.

4.1. Fundamentos de Zookeeper

Varias primitivas utilizadas para la coordinación son comúnmente compartidos a través de muchas aplicaciones.

ZooKeeper no expone directamente primitivas. En su lugar, se expone una API similar a los sistemas de archivos integrado por un pequeño conjunto de llamadas que permite a las aplicaciones implementar sus propias primitivas. Por lo general utilizan términos para denotar estas implementaciones de primitivas. Estos términos incluyen operaciones ZooKeeper que manipulan los nodos de datos pequeños, llamados znodes, que se organizan jerárquicamente en forma de árbol, al igual que en un sistema de archivos. En la figura 4.1 se puede ver El árbol de datos de Zookeeper, en donde el nodo raíz contiene más nodos y tres de estos nodos contienen nodos abajo de ellos, estos nodos hijos son los datos [4].

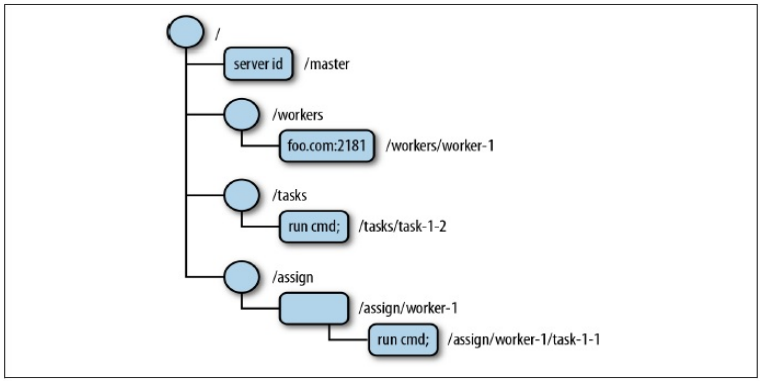


Figura 4.1: Árbol de datos de Zookeeper
Fuente: Flavio Junqueira and Benjamin Reed, "Zookeeper Distributed Process Coordination", Oreilly (2013).

La ausencia de datos en los nodos contienen informacíon importante acerca de un znode. Por ejemplo en una arquitectura Maestro-Esclavo la ausensia de un nodo maestro significa que ningun maestro a sido elegido actualmente. En la figura 4.1 incluye otros znodes importantes para la configuración de la arquitectura Maestro-Esclavo[4]. Como:

- **Nodo Trabajador (Worker znode):** Es el nodo padre de todos los nodos que representan un trabajador que este disponible en el sistema, si un trabajador no esta disponible debe ser removido[4].
- **Nodo Tareas (Task znode):** Es el nodo padre de todas las tareas creadas y que esperan ser ejecutadas por los trabajadores[4].
- **Nodo Asignación (Assing znode):** Es el padre de todos los nodos que representan una asignación de una tarea a un trabajador, Cuando un maestro asigna una tarea a un trabajador, se añade un nodo asignación[4].

4.1.1. Znodes y Diferentes Modos para Znodes

Los nodos de datos que utiliza Zookeeper para realizar sus procesos son denominados znodes.

Al crear un nuevo znode, también es necesario especificar un modo. Los diferentes modos determinan cómo se comporta el znode.

4.1.2. Znodes persistentes y efímeros

Un znode puede ser persistente o efímera. **Un znode persistente** se puede eliminar sólo a través de una llamada a eliminar. **Un znode efímero**, por el contrario, se elimina si el cliente que lo ha creado se bloquea o simplemente cierra su conexión con ZooKeeper[4].

Los Znodes persistentes son útiles cuando el znode almacena datos de una aplicación y estos datos deben ser conservados, incluso después que su creador ya no es parte del sistema. En el ejemplo principal de trabajo, tenemos que mantener la asignación de tareas a los trabajadores, incluso cuando el maestro que realizó la asignación no esta disponible.

Los Znodes efímeros transmiten información acerca de algún aspecto de la aplicación que debe existir sólo mientras la sesión de su creador es válida. El znode principal en el ejemplo principal de trabajo es efímero. Su presencia implica que hay un maestro y maestro esta en servicio. Si el znode maestro permanece mientras que el maestro se ha ido, entonces el sistema no será capaz de detectar que el maestro no esta disponible. Esto evitaría que el sistema pueda avanzar, por lo que el znode debe ir con el maestro. También utilizamos znodes efímeras para los trabajadores. Si un trabajador deja de estar disponible, su sesión expira y su znode trabajador desaparece automáticamente[4].

Un znode efímero puede ser borrado en dos situaciones:

1. Cuando la sesión del cliente creador termina, ya sea por vencimiento o porque es explícitamente cerrado.

2. Cuando un cliente, no necesariamente el creador, lo elimina.

4.1.3. Znodes Secuenciales

Un znode también se puede configurar para ser secuencial. A un znode secuencial se asigna un único, monótonamente creciente número entero. Este número de secuencia se agregará a la ruta utilizada para crear el znode. Por ejemplo, si un cliente crea un znode secuencial con la ruta / tareas / task, ZooKeeper asigna un número de secuencia, por ejemplo 1, y lo anexa a la ruta. El camino de la znode convierte / tareas / task-1. Znodes secuenciales proporcionan una manera fácil de crear znodes con nombres únicos. También proporcionan una manera de ver fácilmente el orden de creación de znodes. En resumen, hay cuatro opciones para el modo de un znode: persistentes, efímeros, persistenses_secuenciales y efímeros_secuenciales[6].

4.2. Versiones Zookeeper

Cada znode contiene un número asociado a él, que se incrementa cada vez que se realizan cambios en el znode. Un par de operaciones en el API se pueden ejecutar de forma condicional setData deleteData, ambas llamadas tienen una versión como parámetro de entrada y la operación de lleva a cabo solo si la versión aprobada por el cliente coincide con la versión actual del servidor. [9]

4.3. Arquitectura de Zookeeper

Las aplicaciones pueden hacer llamadas a Zookeeper a traves de librerías cliente. La librería cliente es responsable de la interacción con los servidores de Zookeeper. La figura 4.4 muestra la relación entre clientes y servidores, cada cliente importa la librería y puede comunicarse con cualquier nodo de Zookeeper.[7]

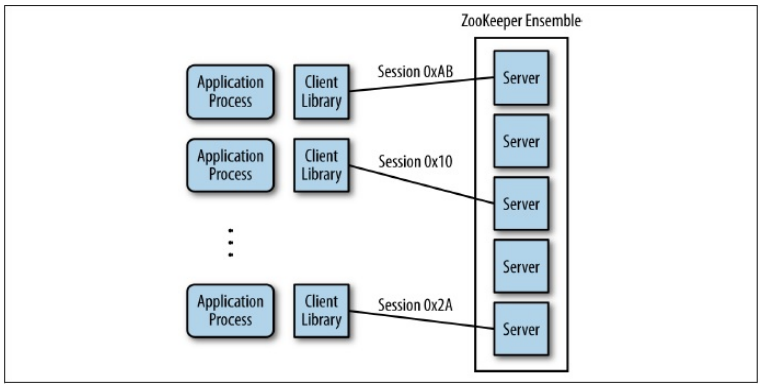


Figura 4.2: Arquitectura de Zookeeper.
Fuente:Flavio Junqueira and Benjamin Reed, "Zookeeper Distributed Process Coordination", Oreilly (2013).

Los servidores de Zookeeper se ejecutan de dos modos : Autónomo y Quorum, en el modo Autónomo hay solo un servidor y el estado de zookeepeer no se replica. En el modo Quorum hay un grupo de servidores que es conocido como conjunto Zookeeper, estos replican el estado de Zookeeper y reciben las peticiones de los clientes. Se utiliza el término “Conjunto Zookeeper” para refereirse a una instalación se servidores. Esta instalación podría contener un único servidor y funcionar en modo autónomo o contener un grupo de servidores y funcionar en modo quorum.[7]

4.3.1. Zookeeper Autónomo

En el modo autónomo Zookeeper no replica su árbol de datos debido a que hay un solo servidor. Este modo es utilizado con fines de aprendizaje o pruebas de funcionamiento.

4.3.2. Zookeeper Quorum

En el modo quorum Zookeeper replica su árbol de datos en todos los servidores en el conjunto. En la administración pública, el quórum es el número mínimo de diputados necesarios para estar presente en la votación. En ZooKeeper, es el número mínimo de servidores que tienen que estar en ejecución y disponible para que ZooKeeper funcione. Este número es también el número mínimo de servidores que tienen para almacenar los datos de un cliente antes de decirle al cliente que se almacena de forma segura. Por ejemplo, podríamos tener cinco servidores ZooKeeper en total, pero un quórum de tres. Mientras las tres servidores han almacenado los datos, el cliente puede continuar, y los otros dos servidores finalmente se pondrán al día y almacenar los datos[4].

Es importante elegir un tamaño adecuado para el quórum. Quórum deben garantizar que, independientemente de los retrasos y caídas en el sistema, cualquier solicitud de actualización del servicio persistirá hasta que otra petición lo reemplace.[4]

4.4. Observadores y Notificaciones

Debido a que Zookeeper accede como un servicio remoto, el acceso a un znode cada vez que un cliente necesita saber su contenido el proceso seria muy costoso, seria aumentar la latencia y las operaciones que se van a ejecutar en Zookeeper. En la figura 4.2 en la segunda llamada a getCHildren/task se obtiene el mismo valor de la primera llamada ocacionando una acción innecesaria[4].

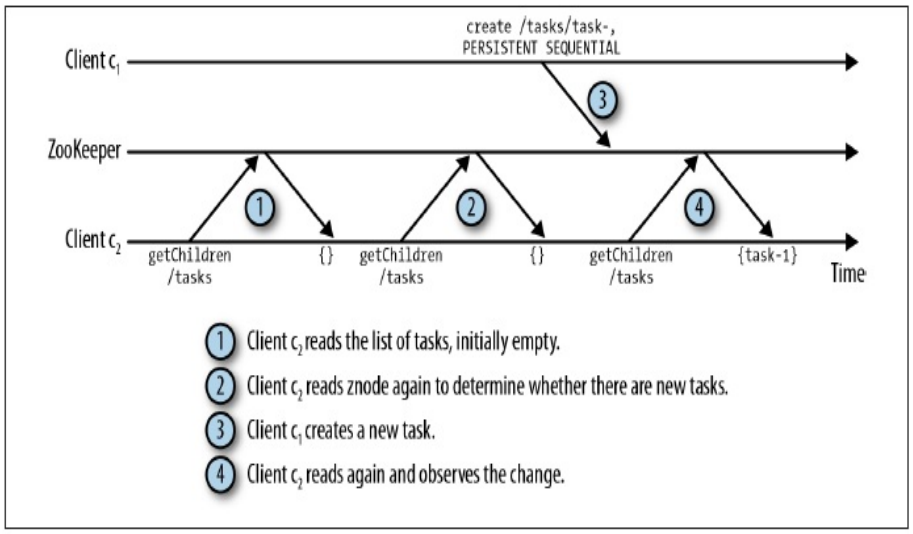


Figura 4.3: Múltiples Lecturas en el Mismo Znode
Fuente:Flavio Junqueira and Benjamin Reed, "Zookeeper Distributed Process Coordination", Oreilly (2013).

Este es un problema común que se da en sondeo. Para sustituir el sondeo se a optado por un mecanismo basado en notificaciones, los clientes se registran con Zookeeper para recibir notificaciones de los cambios de los znodes. EL registro para recibir notificaciones de un znode consiste en establecer un observador. Se debe establecer un observador para cada notificación. En la figura 4.3 se detalla como un cliente lee los valores de Zookeeper solo cuando se recibe una notificaión que indica que el valor de una tarea a cambiado.

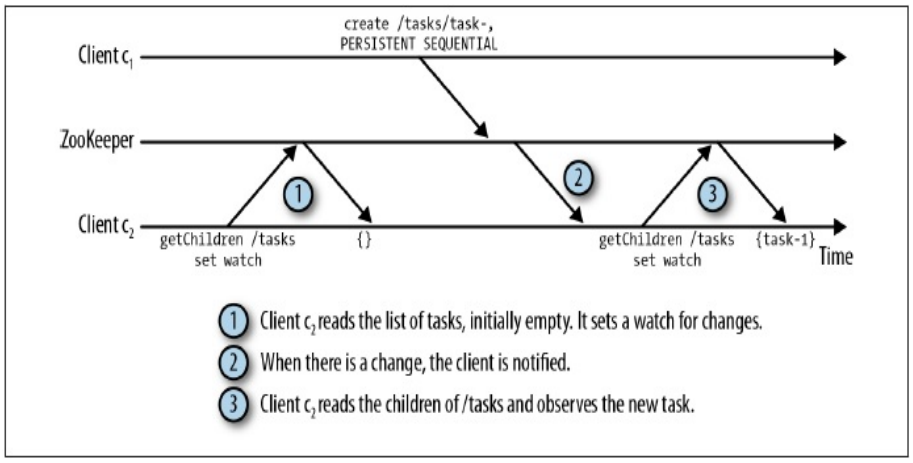


Figura 4.4: Uso de Notificaciones para recibir Información de los Cambios de un Znode
Fuente:Flavio Junqueira and Benjamin Reed, "Zookeeper Distributed Process Coordination", Oreilly (2013).

Determinar los métodos fundamentales encargados en controlar los procesos de decisión común en Zookeeper.

4.5. Descripción General del API

Para poder utilizar los servicios que ofrece Zookeeper las aplicaciones deben realizar una intancia de un objeto de la clase de Zookeeper. Una vez que se establece una conexión a un servidor, un ID de sesión se le asigna al cliente. El cliente enviará informacion al servidor al servidor periódicamente para mantener la validez de la sesión. La aplicación puede llamar a las API ZooKeeper través de un cliente, siempre y cuando el ID de sesión del cliente sigue siendo válida[22].

Si por alguna razón el cliente no puede enviar información hacia el servidor durante un largo periódo de tiempo, el servidor finalizará la sesión, y la aplicación deberá crear una nueva instacia de objeto de la clase Zookeeper para poder hacer llamadas al API.

A continuación se detalla las clases y métodos que ofrece Zookeeeper a los desarrolladores:

4.5.1. Clase Anidada Zookeeper.states

Clase	Descripción
static Zookeeper.states	Devuelve la constante de enumeración de este tipo.
static ZooKeeper.States[]	Devuelve una matriz que contiene las constantes de este tipo de enumeración, en el orden en que se declaran.

Tabla 4.1: Clase Anidada Zookeeper

Fuente: "Apache Zookeeper", http://zookeeper.apache.org/doc/r3.4.1/api/org/apache/zookeeper/ZooKeeper.States.html

4.5.1.1. Enumeraciones que ofrece Zookeeper.states

Enumeración	Descripción
public static final ZooKeeper.States CONNECTING	El cliente esta conectandose con los servidores de zookeeper,
public static final ZooKeeper.States CONNECTED	El cliente esta conectado.
public static final ZooKeeper.States CLOSED	Se cerró la conección con el servidor.
public static final ZooKeeper.States AUTH_FAILED	Muestra cuando el estado de conección falló.

Tabla 4.2: Enumeraciones de Zookeeper.states

Fuente: "Apache Zookeeper", http://zookeeper.apache.org/doc/r3.4.1/api/org/apache/zookeeper/ZooKeeper.States.html

4.5.1.2. Métodos que Ofrece Zookeeper.states

Método	Descripción
isAlive()	Muestra si el proceso esta activo o no
valueOf(String name)	Devuelve la constante de enumeración de este tipo con el nombre especificado.
values()	Devuelve una matriz que contiene las constantes de este tipo de enumeración, en el orden en que se declaran.

Tabla 4.3: Métodos de Zookeeper.states

Fuente: "Apache Zookeeper", http://zookeeper.apache.org/doc/r3.4.1/api/org/apache/zookeeper/ZooKeeper.States.html

4.5.2. Constructores de Zookeeper

Constructor	Descripción
ZooKeeper(String connectString, int sessionTimeout, Watcher watcher)	Datos iniciales para le crear un objeto cliente de Zookeeper
ZooKeeper(String connectString, int sessionTimeout, Watcher watcher, long sessionId, byte[] sessionPasswd)	Datos extras para le crear un objeto cliente de Zookeeper

Tabla 4.4: Constructores de Zookeeper

Fuente:https://zookeeper.apache.org/doc/r3.3.3/api/org/apache/zookeeper/ZooKeeper.html

4.5.3. Métodos de Zookeeper

Método	Descripción
void addAuthInfo(String scheme, byte[] auth)	Añade el esquema especificado, con información de autenticación para la conexión acrtual
void close()	Cierra el objeto cliente
String create(String path, byte[] data, List<ACL> acl, CreateMode createMode)	Crea un nodo con una ruta específica
void create(String path, byte[] data, List<ACL> acl, CreateMode createMode, AsyncCallback.StringCallback cb, Object ctx)	Es la versión Asíncrona del Método create
void delete(String path, int version)	Elimina el nodo con la ruta dada.
void delete(String path, int version, AsyncCallback.VoidCallback cb, Object ctx)	Es la versión Asíncrona del Método delete
Stat exists(String path, boolean watch)	Retorna el statdo del nodo con la ruta dada.
void exists(String path, boolean watch, AsyncCallback.StatCallback cb, Object ctx)	Es la versión asíncrona del Método exist
List<ACL> getACL(String path, Stat stat)	Retorna la ACL y el estado del nodo de la ruta dada

Tabla 4.5: Métodos de Zookeeper

Fuente:https://zookeeper.apache.org/doc/r3.3.3/api/org/apache/zookeeper/ZooKeeper.html

MÉTODO	DETALLE
long getSessionId()	El identificador de sesión para la instancia de cliente ZooKeeper.
byte[] getSessionPasswd()	La contraseña de sesión para la instancia de cliente ZooKeeper.
int getSessionTimeout()	El tiempo de espera de sesión para la instancia de cliente ZooKeeper.
void register(Watcher watcher)	Específica el observador especifico para la conexion
Stat setACL(String path, List<ACL> acl, int version)	Envia el ACL para el nodo de la ruta dada
Stat setData(String path, byte[] data, int version)	Envia datos para el nodo en la ruta dada
String toString()	Representación de la cadena del cliente Zookeeper
void getChildren(String path, boolean watch, AsyncCallback.Children2Callback cb, Object ctx)	La versión asíncrona de getChildren
byte[] getData(String path, boolean watch, Stat stat)	Retorna los datos y estadísticas del nodo de la ruta dada
void getACL(String path, Stat stat, AsyncCallback.ACLCallback cb, Object ctx)	La versión asíncrona de getACL
List<String> getChildren(String path, boolean watch)	Retorna la lista de los hijos del nodo de la ruta dada

Tabla 4.6: Métodos de Zookeeper-2

Fuente:https://zookeeper.apache.org/doc/r3.3.3/api/org/apache/zookeeper/ZooKeeper.html

4.5.4. Métodos Heredados de la Clase java.lang.Object

Método	Descripción
public boolean equals(Object obj)	Indica si algún otro objeto es "igual a" éste.
public final Class<?> getClass()	Devuelve la clase de este objeto en tiempo de ejecución. La clase de objeto devuelto es el objeto que está bloqueado por los métodos estáticos sincronizados de la clase representada.
public int hashCode()	Devuelve un valor de código para el objeto hash . Este método se apoya en beneficio de tablas hash como las que presenta java.util.Hashtable.
public final void notify()	Levanta un solo hilo que está esperando en el monitor de este objeto. Si todos los hilos están esperando en este objeto, uno de ellos es elegido para ser levantado.
public final void notifyAll()	Levanta a todos los hilos que están esperando en el monitor de este objeto.
public final void wait(long timeout) throws InterruptedExce	Hace que el subproceso actual espere hasta que notify() o notifyAll() levanten un hilo para este objeto.

Tabla 4.7: Métodos Heredados

Fuente:https://zookeeper.apache.org/doc/r3.3.3/api/org/apache/zookeeper/ZooKeeper.html

4.6. Sesiones

Antes de ejecutar cualquier petición hacia Zookeeper, el cliente debe establecer una sesión con el servicio. El término sesión es muy importante cuando se trabaja con Zookeeper ya que todas las operaciones que ejecuta el cliente hacia Zookeeper se asocian a una sesión. Las sesiones ofrecen una garantía de orden en las peticiones que realiza el cliente, lo que significa que las sesiones ejecutan las solicitudes en orden FIFO (Primero en Entrar, Primero en Salir). Normalmente un cliente tiene solo una sesión abierta por lo que sus peticiones se ejecutan en orden FIFO. [4]

4.6.1. Estados y Tiempo Real de una Sesión

El tiempo real de una sesión corresponde al período de tiempo entre la creación y la finalización de la sesión, si se cierra correctamente o por un tiempo de espera

indefinido. Para conocer que es lo que realmente sucede en una sesión se debe conocer los posibles estados en los que puede estar una sesión y los eventos que pueden cambiar este estado. Los estados en los que puede estar una sesión son Conectando (CONECTING), Conectado (CONECTED), Cerrado (CLOSED) y No Conectado (NOT-CONECTED), las transiciones entre los estados dependen de varios eventos que ocurren el cliente, y el servicio[4].

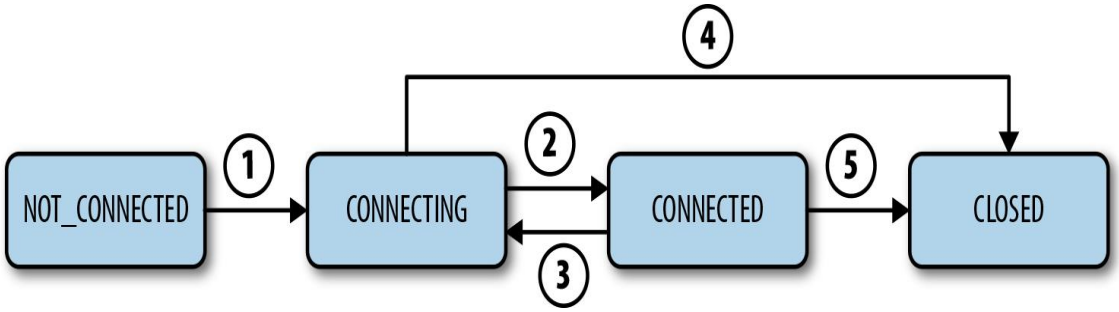


Figura 4.5: Estados de Sesión y Transisiones
Fuente:Flavio Junqueira and Benjamin Reed, "Zookeeper Distributed Process Coordination", Oreilly (2013).

4.6.2. Creación de una Sesión Zookeeper

El api de Zookeeper tiene la posibilidad de realizar llamadas directas al sistema, cada vez que se inicia Zookeeper se crea una sesión en el sistema, manteniendo un proceso activo hasta que se termine la ejecución de Zookeeper[4].

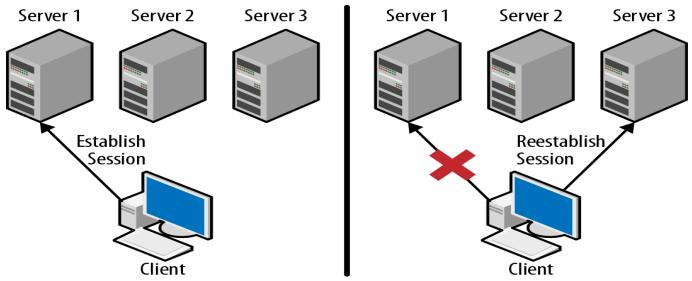


Figura 4.6: Sesiones en Zookeeper
Fuente:Flavio Junqueira and Benjamin Reed, "Zookeeper Distributed Process Coordination", Oreilly (2013).

La estructura del constructor que se crea es el siguiente.

```
1 ZooKeeper (
2 String connectString,
3 int sessionTimeout,
4 Watcher watcher
5 );
```

En donde:

connectString= Contiene el nombre de host y el puerto que va a utilizar Zookeeper.

sessionTimeout= El tiempo en milisegundos que Zookeeper espera un mensaje del cliente para luego declarar una sesión inválida.

watcher= Es objeto en el cual se recibe los eventos de la sesión. Debido a que watcher es una interfaz se debe implementar una clase y luego crear la instancia para el constructor de Zookeeper. Los clienztes utilizan la interfaz watcher para supervisar el estado de la sesión de Zookeeper. Se generara eventos cuando se establezca o se pierda la conexión al servidor de Zookeeper, también se puede utilizar para monitorear los cambios que sucedan con los datos que maneja Zookeeper.[4]

La forma de establecer una sesión es asíncrona. Este constructor iniciará la conexión con el servidor y retornará inmediatamente antes que la sesión este plenamente establecida. El argumento watcher especifica el Observador que sera notificado de cualquier cambio de estado en la sesión. Esta notificación puede llegar en cualquier momento antes o después de la llamada al costructor que retorna.

La instancia establecida de Objeto Cliente Zookeeper seleccionará arbitrariamente un servidor de los especificados en la cadena de conexión creada anteriormente y tratará de conectarse a el. Si la canexión con el servidor establecido fallá, otro servidor especificado en la cadena de conexión sera utilizado hasta que se establezca una conexión. El cliente seguirá intentando conectarse hasta que la sesión se cierre explícitamente o la sesión haya sido caducada por el servidor.[9]

4.7. Sesión Zookeeper modo Autónomo

Para empezar a trabajar se debe realizar la descarga de Zookeeper de su pagina oficial http://mirror.symnds.com/software/Apache/zookeeper/, una vez hecho esto se descomprime los archivos y podemos empezar a trabajar.

Para poder iniciar se utilizan los archivos zkServer y zkCli que se encuentran en la carpeta bin de Zookeeper.

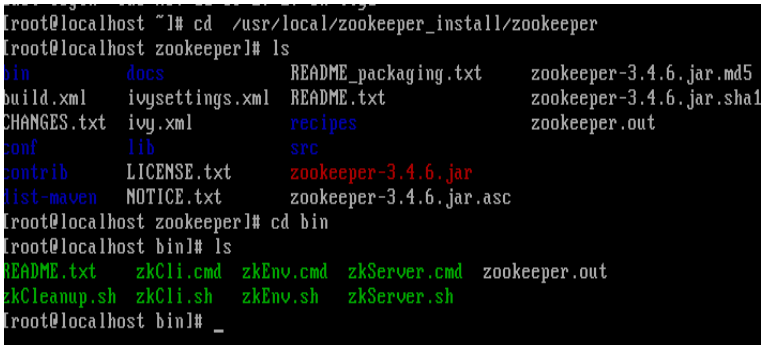


Figura 4.7: Ubicación de archivos de Configuración
Elaborado por: Renato Toasa

Pero antes es necesario mover el directorio de datos temporales hacia otra ubicación ya que al tenerlos en la raíz de Zookeeper puede ocurrir algunos problemas debido al espacio, esto para esto se edita el archivo zoo.cfg que se encuentra en la carpeta conf de Zookeeper y se edita la parde dataDir

Figura 4.8: Archivo zoo.cfg
Elaborado por: Renato Toasa

Una vez realizado esto ahora se puede iniciar Zookeeper, desde el terminal de linux se ubica en la carpeta Zookeeper y se ejecuta lo siguiente:

```
1 bin/zkServer.sh start
```

start permite iniciar el servidor de Zookeeper,

z

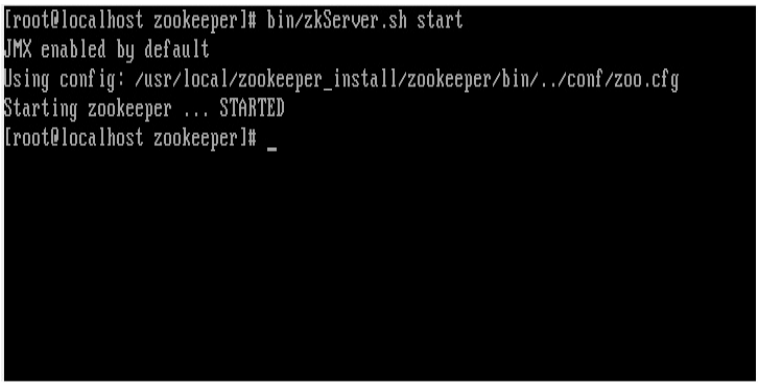


Figura 4.9: Servidor Zookeeper Iniciado
Elaborado por: Renato Toasa

Ahora se muestra el servicio iniciado.

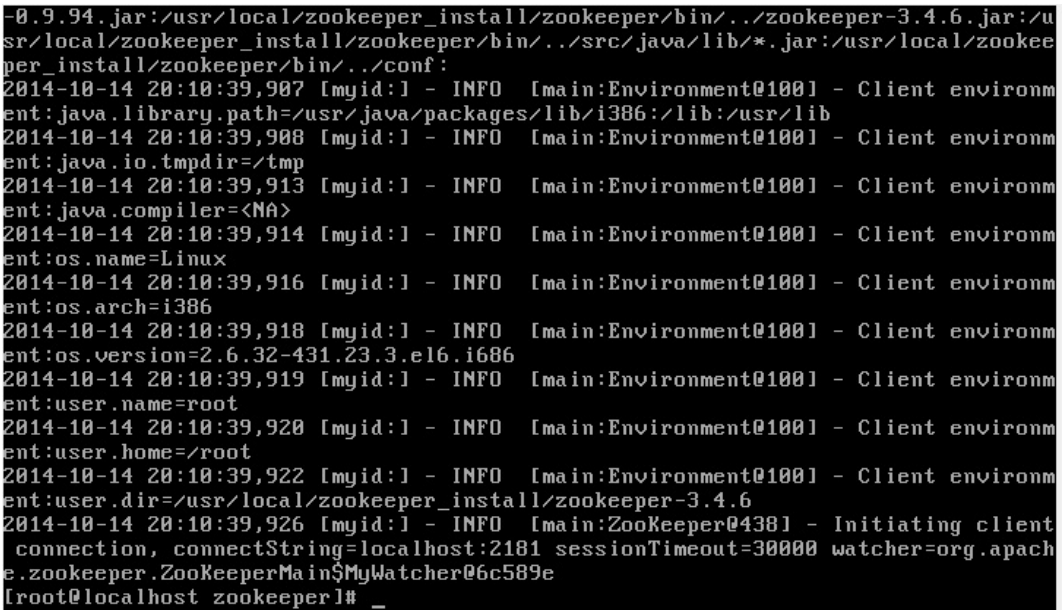


Figura 4.10: Cliente Zookeeper Iniciado
Elaborado por: Renato Toasa

4.8. Sesión Zookeeper modo Quorum

Para conseguir la fiabilidad del servicio se debe ejecutar varios servidores. Se puede ejecutar varios servidores incluso si se tiene una sola máquina, esto se realiza mediante la configuración de ciertos archivos cono se muestra a continuación.[4].

Para que los servidores esten comunicados entre es necesario establecer algunos datos para la configuración, para esto abrimos el archivo zoo.cfg que se encuentra en la carpeta conf de Zookeeper, y se edita los siguientes datos.

```
1 tickTime=2000
2 initLimit=10
3 syncLimit=5
4 dataDir=./data
5 clientPort=2181
6 server.1=127.0.0.1:2222:2223
7 server.2=127.0.0.1:3333:3334
8 server.3=127.0.0.1:4444:4445
```

En cada servidor hay tres campos separados con dos puntos, el primer campo es la dirección IP del servidor, el segundo y tercer campo son los números de puertos utilizados para la comunicación y la elección de líder de quorum. Debido a que estamos inicializando tres procesos de servidor en la misma máquina se utiliza diferentes números de puerto para cada entrada[4].

Una vez realizado esto se crea directorios de datos para cada servidor, mediante la terminal de linux se crea de la siguiente forma:

```
1 mkdir z1
2 mkdir z1/data
3 mkdir z2
4 mkdir z2/data
5 mkdir z3
6 mkdir z3/data
```

Cuando se inicia un servidor es necesario saber que servidor se acaba de iniciar. Un servidor se da cuenta de su identificación mediante la lectura de un archivo llamado myid en el directorio de datos. Podemos crear estos archivos con los siguientes comandos:

```
1 echo 1 > z1/data/myid
2 echo 2 > z2/data/myid
3 echo 3 > z3/data/myid
```

Ahora es necesario crear el archivo de configuración en cada servidor, para esto se copia el archivo zoo.cfg configurado anteriormente y se cambia unicamente el puerto por el que se conecta, esto se debe realizar en cada directorio creado para los servidores.

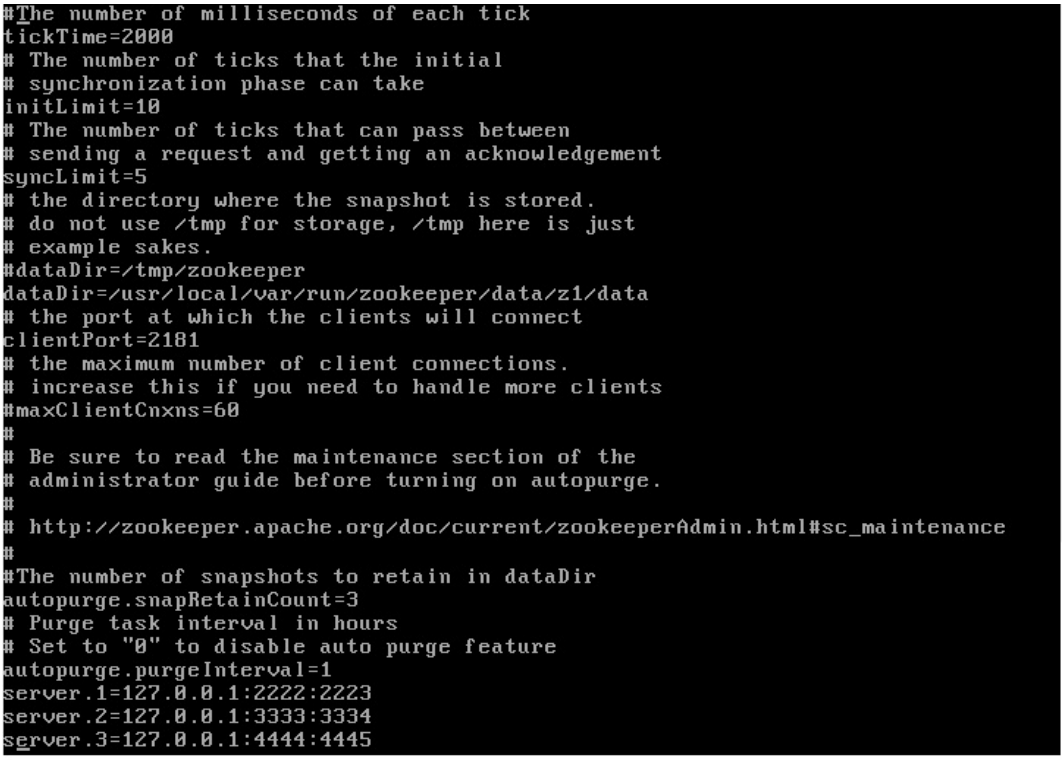


Figura 4.11: Configuración z1, z2,z3
Elaborado por: Renato Toasa

Ahora se procede a iniciar cada uno de los servidores virtuales creados anteriormente, para esto hay que ubicarse en cada directorio creado para el servidor y desde ahi ubicarse en la carpeta Zookeeper y ejecutar skServer que es el servidor de Zookeeper.

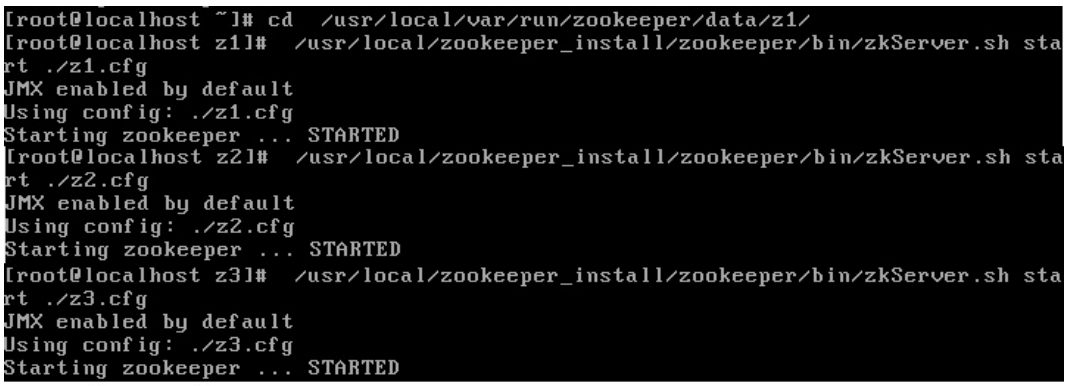


Figura 4.12: Iniciar Servidores
Elaborado por: Renato Toasa

A continuación se procede a abrir el archivo Zookeeper.out que se encuentra en cada uno de los servidores virtuales creados. Los resultados son

Figura 4.13: Zookeeper.out z1
Elaborado por: Renato Toasa

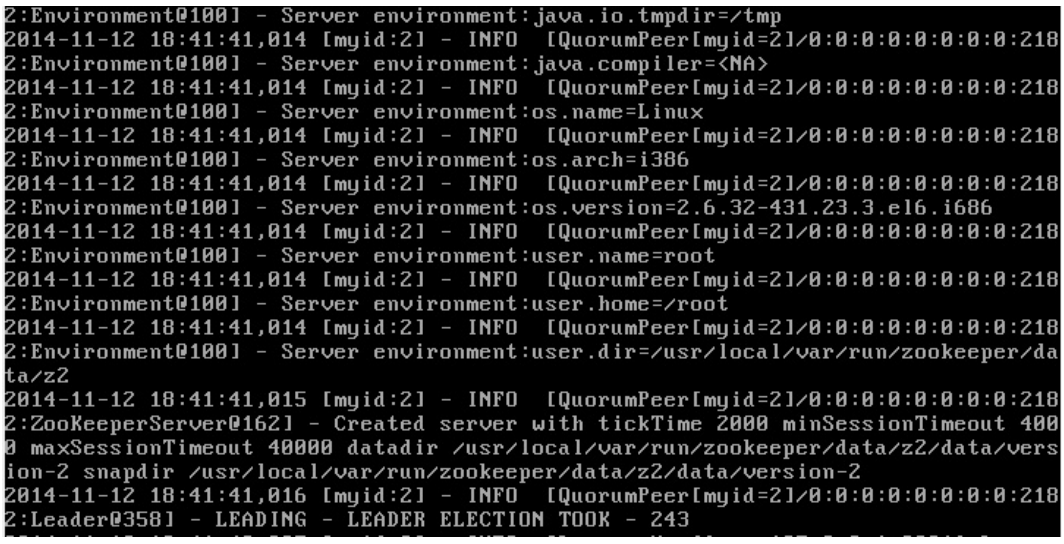


Figura 4.14: Zookeeper.out z2
Elaborado por: Renato Toasa

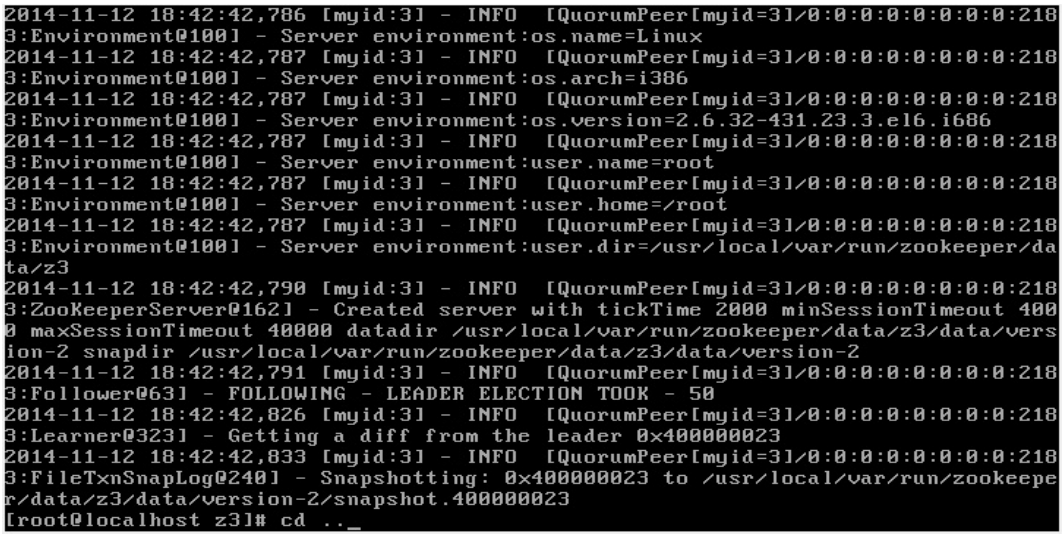


Figura 4.15: Zookeeper.out z3
Elaborado por: Renato Toasa

ES importante revisar la palabra que entecede a LEADER ELECTION TOOK, esta puede ser FOLLOWING cuando esta esperando instrucciones del líder, LEADING muestra que es el proceso que esta liderando a los demás.

Zookeeper es una API que se basa en Algoritmos Distribuidos para Ordenar Eventos, **Entrega Casual de Mensajes, Elección de un proceso coordinador(líder) y para Consenso Distribuido**, lamentablemente no se tiene un pseudocódigo propio disponible ya que Hadoop propietario de Zookeeper no lo a liberado, pero se obtuvo un pseudocódigo que realiza el consenso y elección de lider de otros autores ya que los principios de funcionamiento son los mismos. En la figura 4.16 se observa el pseudocódigo del algoritmo de elección de líder y en la Figura 4.17 se observa el pseudocódigo del algoritmo de consenso .[5]

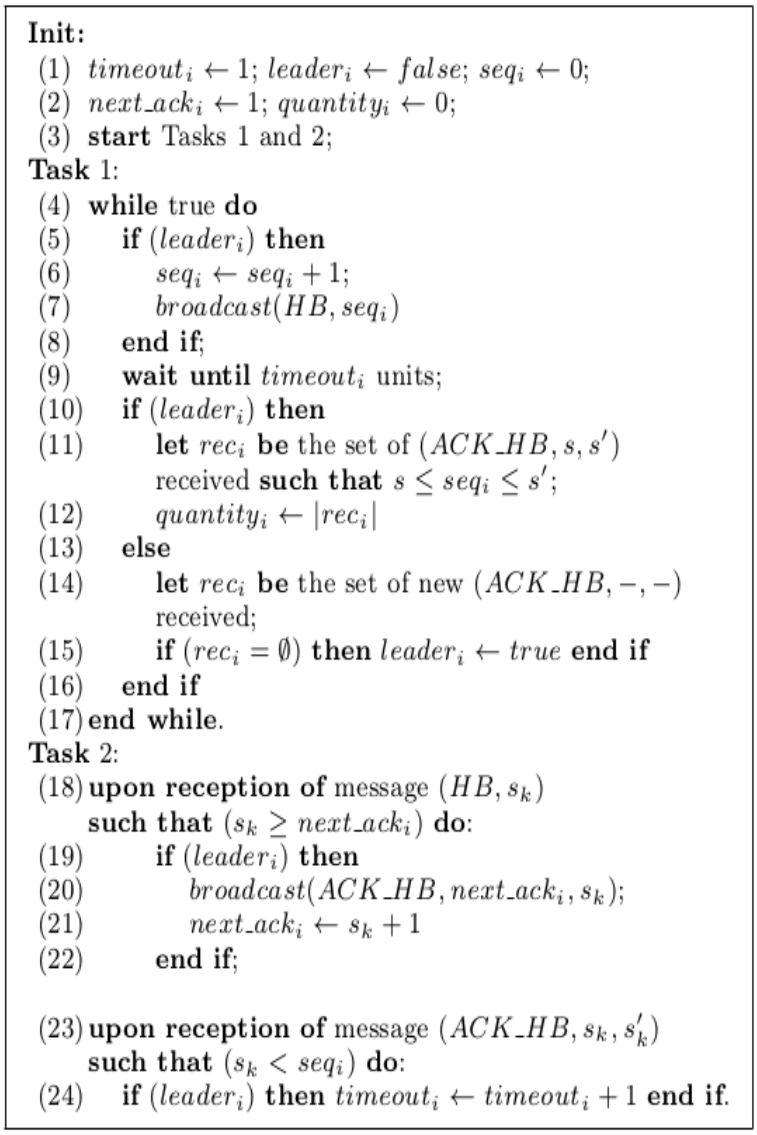


Figura 4.16: Algoritmo de Elección de líder
Fuente: Sergio Arevalo, Carlos Herrera, Ernesto Jimenez y Jian Tang, "Eventual Election of Multiple Leaders for Solving Consensus in Anonymous Systems", distributed systems (2014), 1-8.

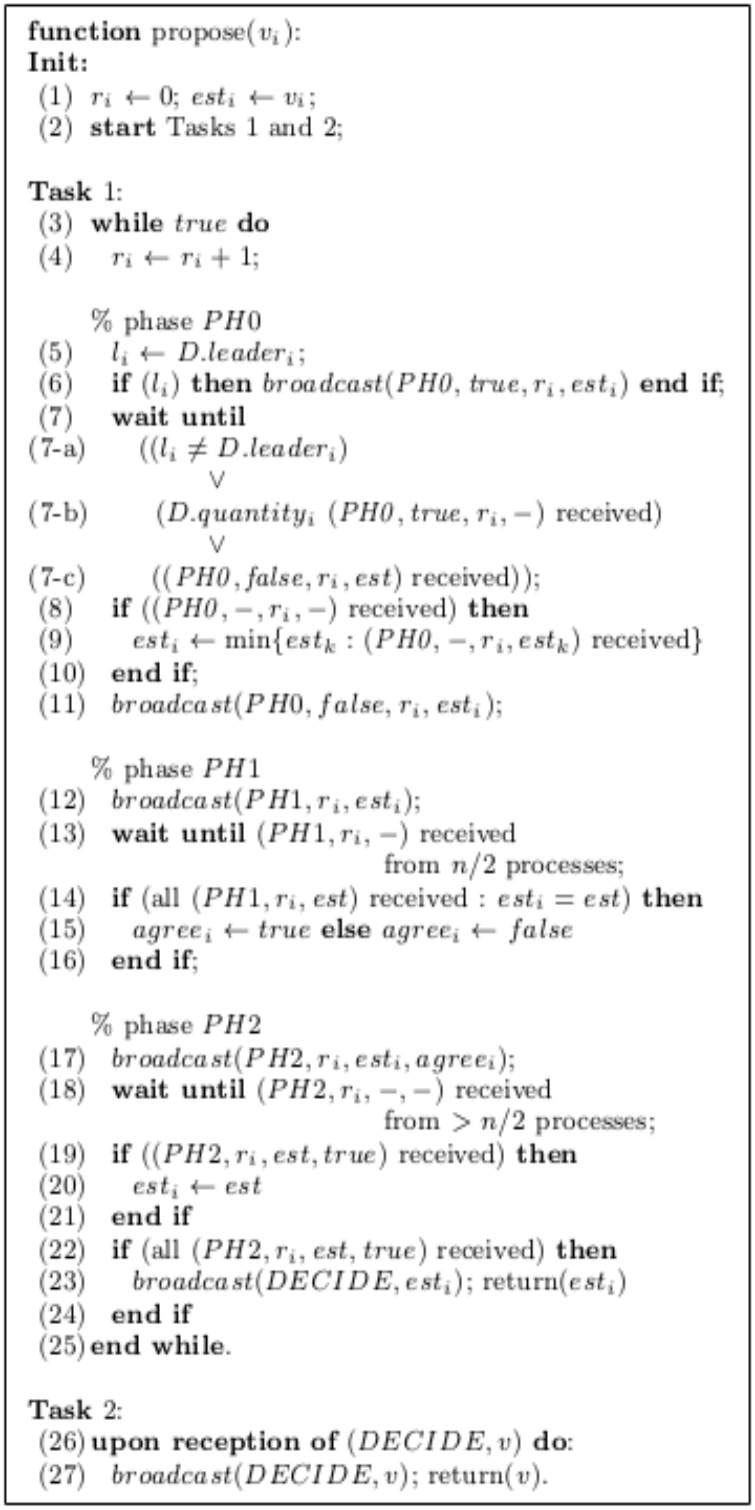


Figura 4.17: Algoritmo de Consenso
Fuente: Fuente: Sergio Arevalo, Carlos Herrera, Ernesto Jimenez y Jian Tang, "Eventual Election of Multiple Leaders for Solving Consensus in Anonymous Systems", distributed systems (2014), 1-8.

El algoritmo de Elección de Líder fue programado por La Escuela Politecnica Nacional, y el algoritmo de Consenso fue programado por la Universidad Técnica de Ambato, esto se realizó debido a que las 2 Universidades forman parte del proyecto "Servicios de Coordinación en la Nube cuando los Elementos Intervinientes son Anónimos".

Como resultado se puede verificar que hay eleccíon de un proceso líder y se realiza consenso cuando los servidores llegan a un comun acuerdo, la figura 4.18 muestra estos resultados.

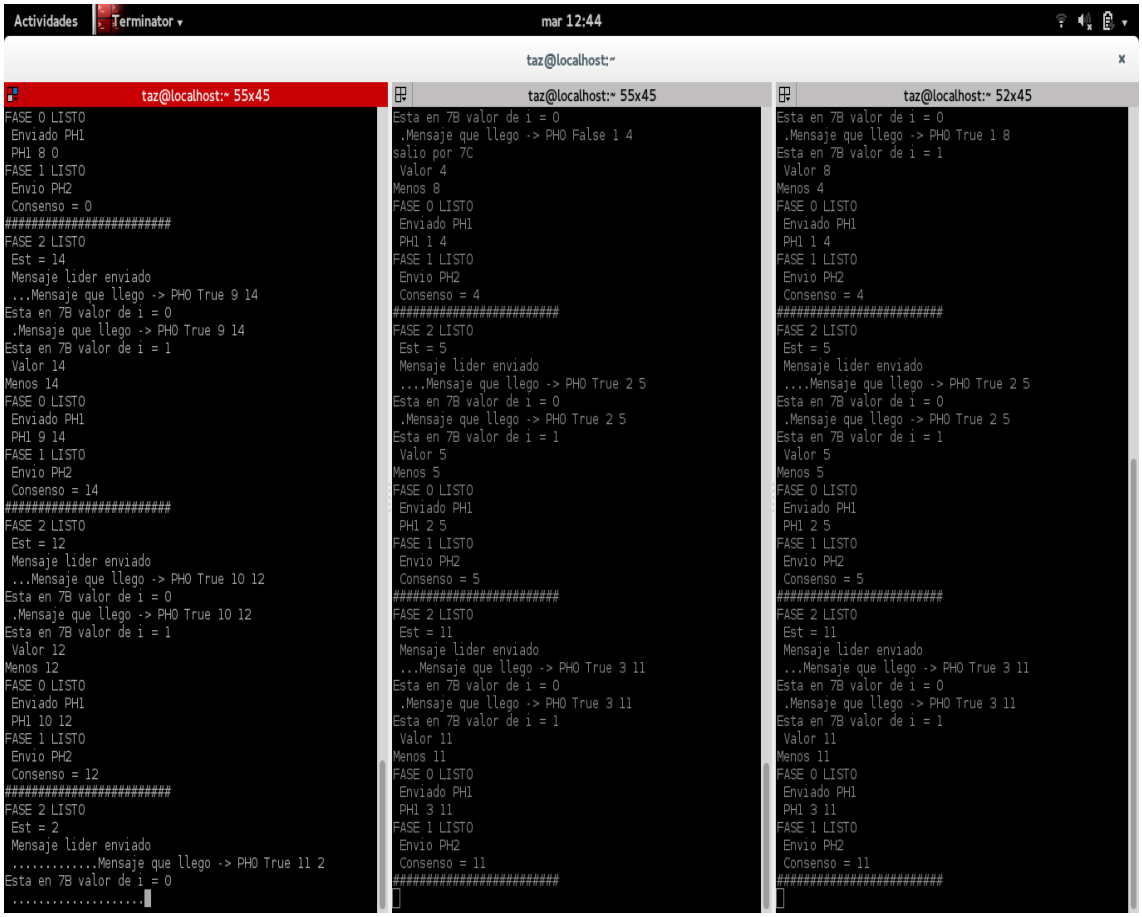


Figura 4.18: Resultado de Consenso y elección de líder
Elaborado por: EPN y UTA

Mediante estos algoritmos Zookeeper ofrece :

1. Consistencia secuencial: las actualizaciones de un cliente se aplican en el orden en el que fueron enviadas.

2. Atomicidad: las actualizaciones se realizan de forma atómica

3. Imagen única del sistema con independencia del servidor al que se conecta

4. Fiabilidad: cuando una actualización se completa, perdura.

Zookeeper consigue realizar esto mediante:

1. **Broadcast atómico**
   a) Entrega de mensajes fiable, si un mensaje se entrega a un servidor será entregado a todos.
2. **Orden total:** Si en un servidor un mensaje A se entrega antes que un mensaje B, entonces en todos los servidores se hace de igual forma
3. **Orden causal:**
   a) Si un mensaje B es enviado después de que un mensaje A ha sido entregado por el emisor de B, A debe ser ordenado antes que B.
   b) Si un proceso envía C después de enviar B, C debe ordenarse después de B.
4. **El orden total lo consigue con identificadores únicos en los mensajes (proceso líder) .**
5. **Elección de líder.**
6. **La consistencia se asegura utilizando Quorums.**

Realizar una simulación de funcionamiento de Zookeeper en un entorno distribuido controlado.

4.9. Simulación de Funcionamiento de Zookeeper

Una vez entendido todo lo que Zookeeper ofrece se va a realizar una simulacion basica para poder apreciar de una mejor manera lo que ofrece el servicio.

4.9.1. Funcionamiento de Zookeeper

Iniciar los 3 servidores creados anteriormente:

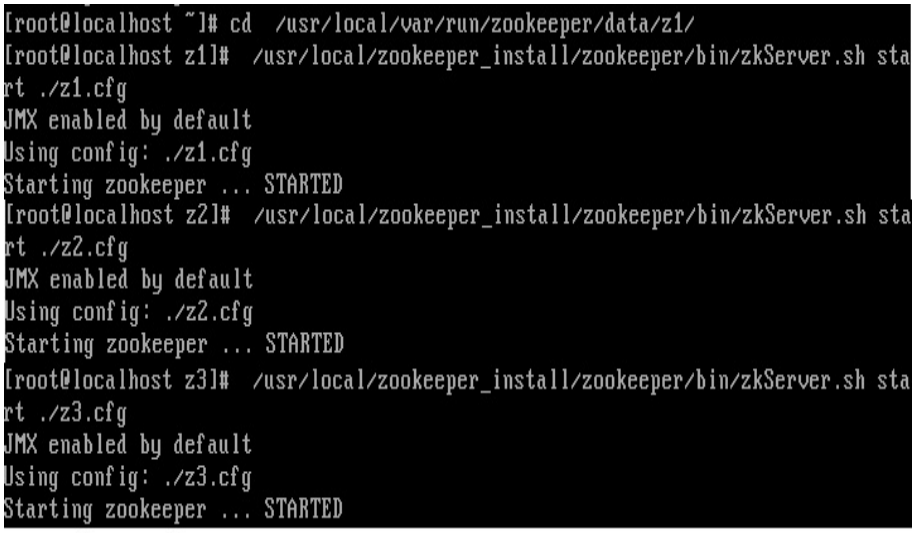


Figura 4.19: Servidores Iniciados
Elaborado por: Renato Toasa

Una vez hecho esto el servicio Zookeeper esta disponible, ahora se va a configurar el cliente que va a conectarse al servicio. La cadena de conexión enumera todas las máquinas que estan activas: puertos de las máquinas activas, en esta caso la cadena de conexión es: 127.0.0.1:2181, 127.0.0.1:2182, 127.0.0.1:2183 que es lo que se creo en los archivos de configuración de los servidores de Zookeeper.

Se utiliza zkCli.sh para conectar el cliente, de la siguiente forma :

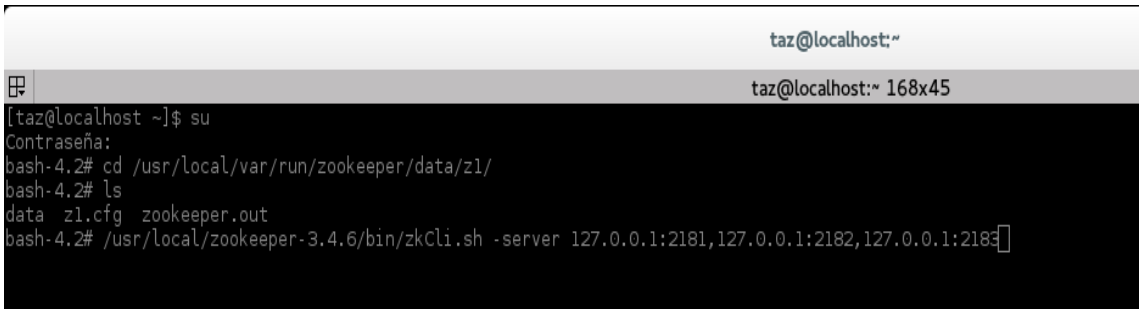


Figura 4.20: Conexión del Cliente Zookeeper
Elaborado por: Renato Toasa

Cuando el servidor se conecta, se ve un mensaje de la forma:

Session establishment complete on server 127.0.0.1/127.0.0.1:2183

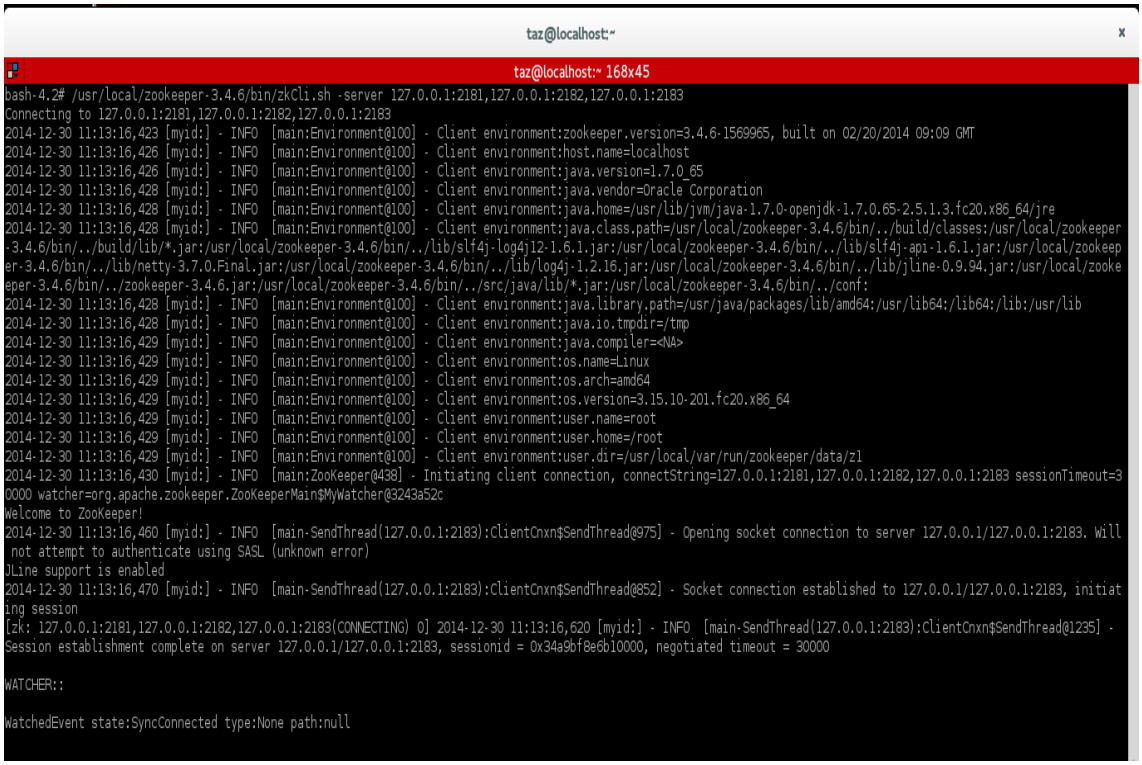


Figura 4.21: Conexión Correcta con el Servidor
Elaborado por: Renato Toasa

En este caso el registro muestra que el cliente se conectó al 127.0.0.1;2183, se puede detener la conexión presionando Ctrl+C y volver e ejecutar el comando y se observa en el registro la conexón hacia otro servidor como se muestra en la Figura 4.21

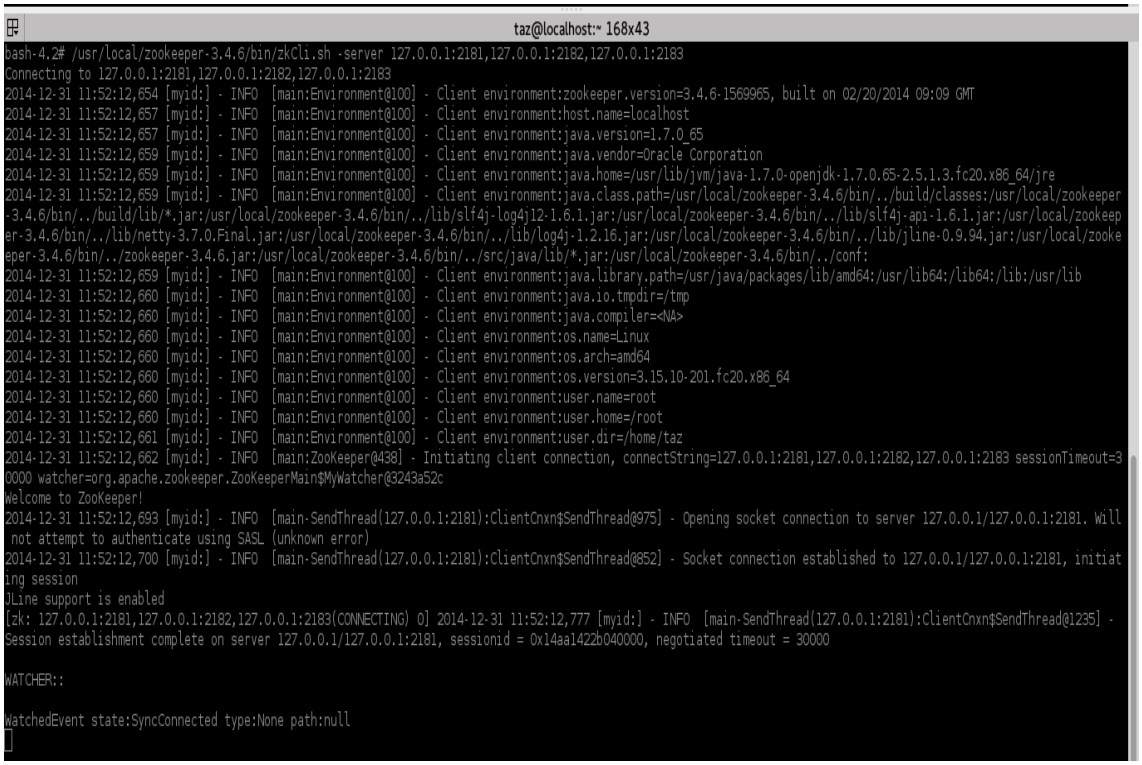


Figura 4.22: Conexión en otro Servidor
Elaborado por: Renato Toasa

Los clientes se conectan en un orden aleatorio a los servidores en la cadena de conexión. Esto permite a ZooKeeper lograr el equilibrio de carga simple. Sin embargo, no se permite a los clientes especificar una preferencia por un servidor para conectarse.

Ahora se crea una estructura maestro-exclavo con los Znodes de Zookeeper, El modelo Maestro-Esclavo implica 3 funciones :

1. **Master**: Los observadores master son para los nuevos monitorear mediante notificaciones a los trabajadores(workers) y tareas(tasks), y para asignar tareas a los trabajadores disponibles.

2. **Worker**: Los workers se registran en el sistema, para asegurarse de que el master verifique que están disponibles para ejecutar tareas, y luego para asignarlos con nuevas tareas.

3. **Client**: Los clientes crean nuevas tareas y esperan respuestas del sistema.

Ahora se va a detallar lo que cada rol debe realizar:

4.9.1.1. El Rol Master

Un solo proceso puede ser llamado master, para esto el proceso debe crear un nodo efímero llamado master [4].

La Figura 4.23 muestra el proceso de la creación de un nodo master, se utiliza -e para determinar que nodo que se creará es efímero.

Figura 4.23: Creación nodo Master
Elaborado por: Renato Toasa

Para verificar que se a creado correctamente se procede a listar la raíz del árbol ZooKeeper de la siguiente manera:

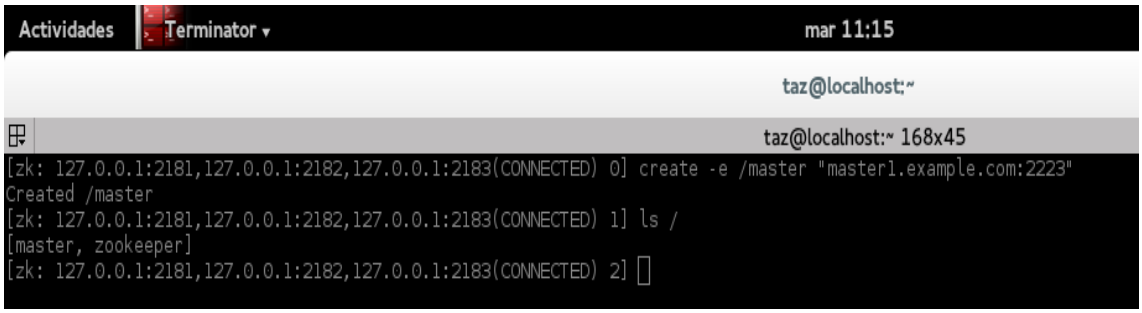


Figura 4.24: Listar árbol Zookeeper
Elaborado por: Renato Toasa

Una vez hecho esto se procede a obtener los metadatos y los datos del nodo master.

Figura 4.25: Obtener Metadatos y datos de Master
Elaborado por: Renato Toasa

Primero se creo un znode efímera con la ruta / master. Se añade la información del host al znode en caso de que otros necesitan comunicarse con el znode creado. No es estrictamente necesario agregar la información de la máquina, pero se lo hizo sólo para demostrar que se puede añadir datos si es necesario. Para hacer el znode efímero, se a añadido la opción -e. Por la razón que un nodo efímero se elimina automáticamente si la sesión en la que se ha creado se cierra o expire.

Pero si algun proceso intentea crear otro nodo master no se va a crear debido que solo un proceso puede ser master:

Figura 4.26: Nodo Existente
Elaborado por: Renato Toasa

ZooKeeper dice que ya existe un nodo / master. De esta manera, el segundo proceso sabe que ya hay un maestro. Sin embargo, es posible que el maestro activo puede bloquearse, y el maestro de copia de seguridad puede que tenga que asumir el papel de maestro activo. Para detectar esto, tenemos que establecer un observador sobre el nodo / master de la siguiente manera:

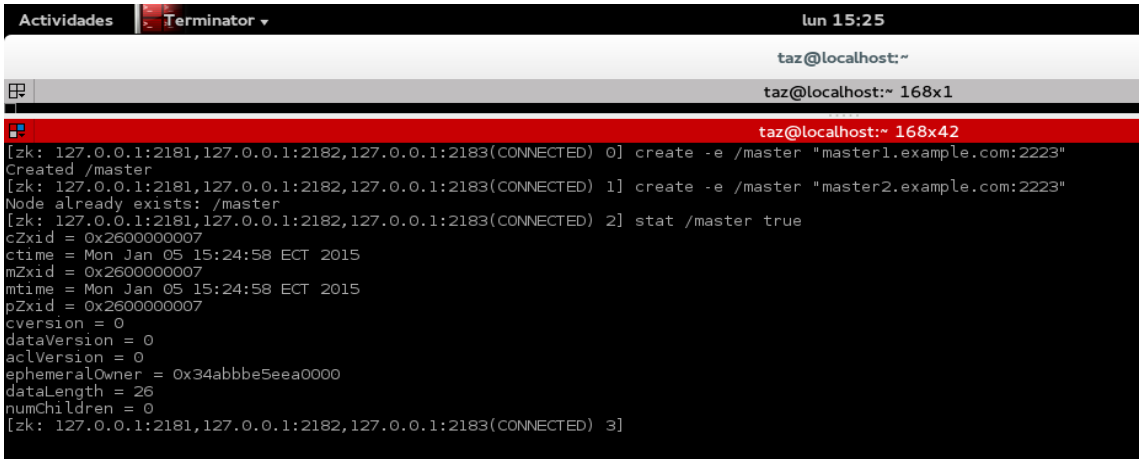


Figura 4.27: Observador en el Nodo
Elaborado por: Renato Toasa

El comando stat obtiene los atributos de un znode y permite establecer una observador sobre la existencia de la znode.

Para continuar con la explicación de los roles Worker y Client es necesario crear znodes padres para ver el funcionamiento de workers y clients.

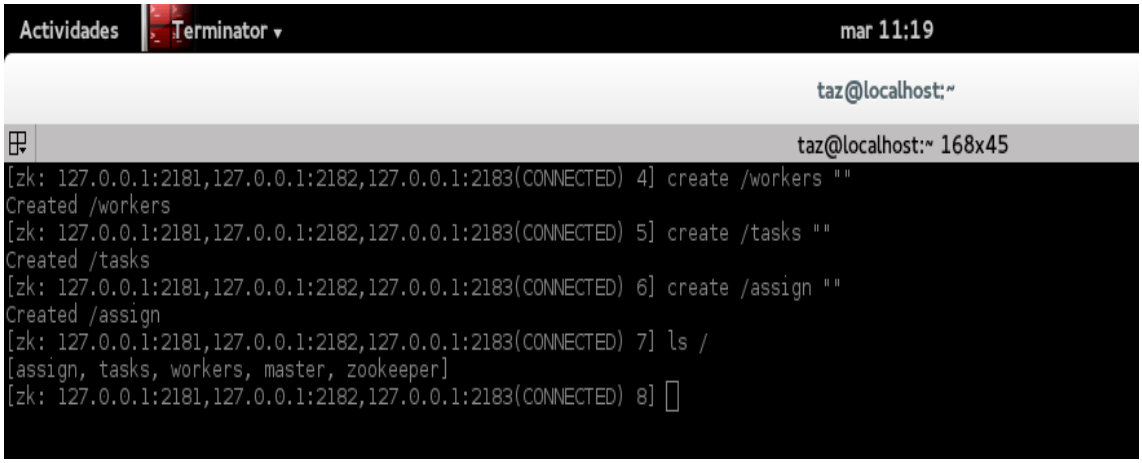


Figura 4.28: Creación workers, task, /assign
Elaborado por: Renato Toasa

Los tres nuevos znodes son znodes persistentes y no contienen datos. Se utiliza estos znodes para verfificar el funcionamiento del master para que los workers que están disponibles decirles cuando hay tasks para asignar y hacer asignaciones a los workers.

Hay que tomar en cuanta que en una aplicación real, estos znodes necesitan ser creados ya sea mediante un proceso primario antes de que comience la asignación de tareas o por algún procedimiento de arranque. Independientemente de cómo se crean, una vez que existen el maestro tiene que observar los cambios en los hijos de los workers y tasks.

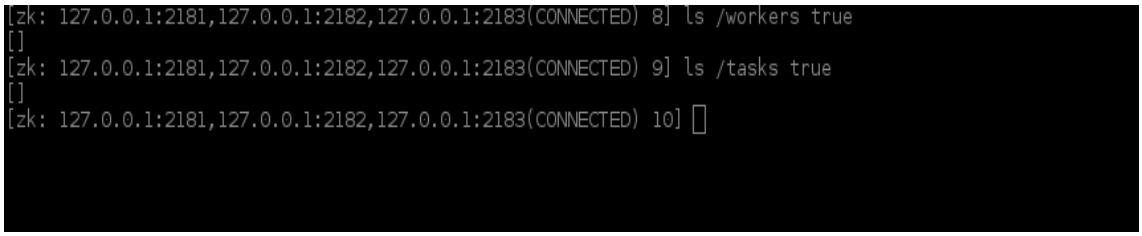


Figura 4.29: Cambios en Workers, tasks
Elaborado por: Renato Toasa

4.9.1.2. El rol Worker

El primer paso de un worker es notificar al maestro que está disponible para ejecutar tareas. Lo hace mediante la creación de un znode efímera. Los workers utilizan sus nombres de host para poder identificarse[4]:

Figura 4.30: Creación workers
Elaborado por: Renato Toasa

A continuación, el worker necesita para crear un znode padre, /assign/worker1.example.com, para recibir asignaciones.

Figura 4.31: Znode para Notificaciones
Elaborado por: Renato Toasa

Ahora si el worker esta listo para recibir asignaciones.

4.9.1.3. The Client Role

El Client añade tareas en el sistema. Para este ejemplo, no importalo que la tarea realiza realmente. Así que se supone que el cliente pide al sistema master-worker ejecutar un comando cmd con la tarea a realizar. Para añadir una tarea al sistema, un cliente realiza lo siguiente[4]:

Figura 4.32: Creación de una tarea
Elaborado por: Renato Toasa

El master siguiente comprueba la nueva tarea, obtiene la lista de todos los workers disponibles, y lo asigna a worker1.example.com que es el que se creo anteriormente.

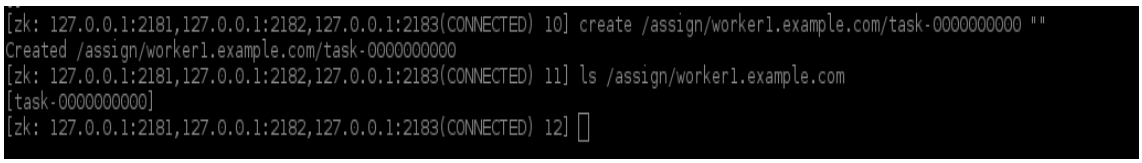


Figura 4.33: Asignación task - worker
Elaborado por: Renato Toasa

El worker recibe una notificación de que una nueva task se ha asignado.

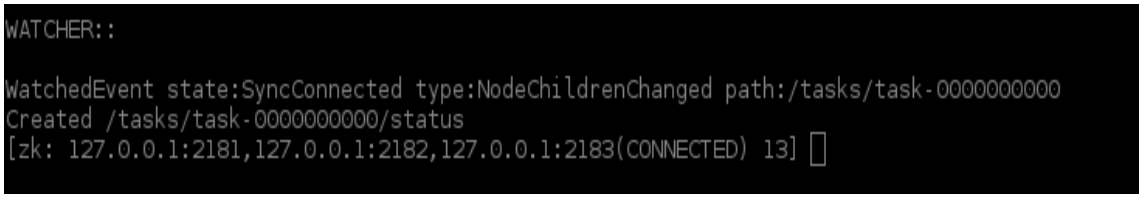


Figura 4.34: Notificación de task Asignada
Elaborado por: Renato Toasa

Una vez que el worker termina de ejecutar la tarea, se añade un estado “done” al znode task:

Figura 4.35: Estado done al znode task
Elaborado por: Renato Toasa

Se puede observar la información y estado de la tarea para verificar que se creo correctamente :

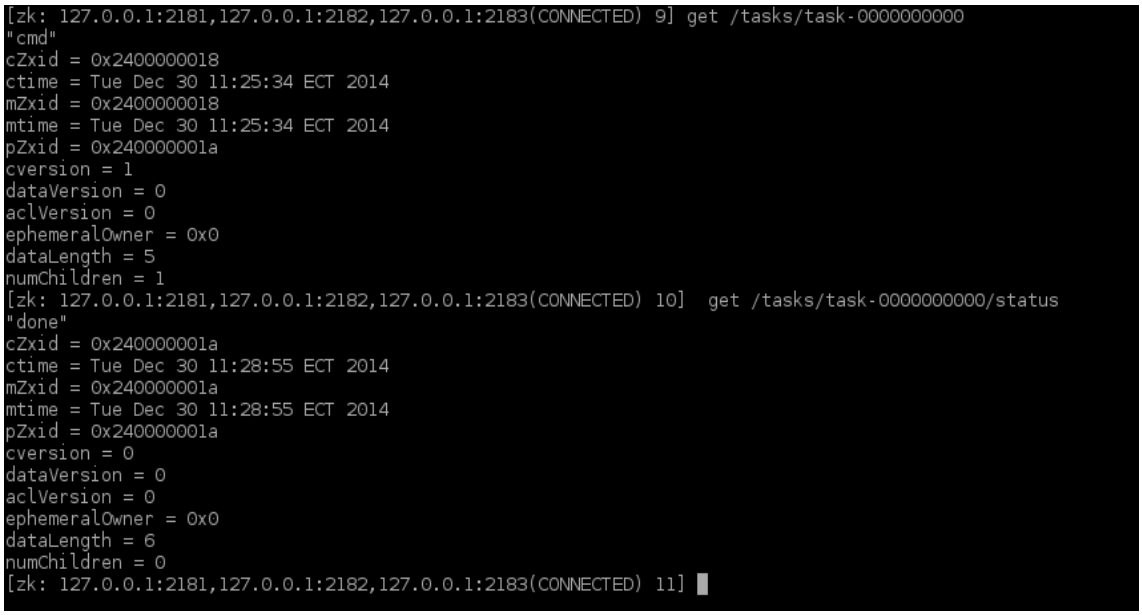


Figura 4.36: Verificación task
Elaborado por: Renato Toasa

El cliente comprueba el contenido de la znode estado para determinar qué ha sucedido a la tarea. En este caso, se ha ejecutado con éxito las tareas y el resultado es "done".

Las tareas pueden ser más sofisticados e incluso involucrar a otro sistema distribuido. Lo importante de esto es que, independientemente de lo que la tarea realice, el mecanismo para ejecutarlo y transmitir los resultados a través de ZooKeeper es en esencia el mismo.

Toda esta informacón que mostrará Zookeeper sera mostrada en un Monitor el mismo que fue programado y ejecutado, a continuación se muestra la información del Monitor de Zookeeper:

4.9.2. Monitor Zookeeper

4.9.2.1. Código Fuente

Clase Master Watcher para el monitoreo.

```
1  import java.io.IOException;
2  import org.apache.zookeeper.WatchedEvent;
3  import org.apache.zookeeper.Watcher;
4  import org.apache.zookeeper.ZooKeeper;
5  /**
6   *
7   * @author renato
8   */
9  public class Master implements Watcher{
10 ZooKeeper zk;
11 //Crea una variable en la cual se guarda el puerto que se utiliza.
12 String hostport;
13 Master (String hostport){
14     this.hostport= hostport;
15 }
16
17 //Construye el objeto zookeeper utilizando el objeto master creado
       anteriormente
18 void startZK(){
19         try{
20     zk = new ZooKeeper(hostport, 15000, this);
21     }catch(IOException ex){
22                 System.out.println("Error al instanciar zk");
23         }
24 }
25
26 //Imprime el evento que esta en activo en ese momento
```

```
27 public void process(WatchedEvent e){
28     System.out.println(e);
29 }
30 //Una vez que se ha conectado a ZooKeeper, habrá un subproceso de fondo
       que mantendrá la sesión ZooKeeper. Este hilo es un hilo de
       utilidad, que significa que el programa puede salir incluso si el
       hilo está todavía activo
31
32    public static void main(String[] args)throws Exception {
33         Master m = new Master(args[0]);
34         m.startZK();
35         Thread.sleep(60000);
36     }
37 }
```

- **Programación de un Hilo para una ejecución constante.**

```
1  import java.awt.TextArea;
2  import java.io.BufferedReader;
3  import java.io.File;
4  import java.io.FileReader;
5  import java.io.IOException;
6  import java.io.InputStreamReader;
7  import java.util.logging.Level;
8  import java.util.logging.Logger;
9  /**
10  *
11  * @author renato
12  */
13 //Creacion de un Hilo para guardar los datos en un archivo, de esta
       forma se mantiene una ejecucion constante.
14 public class Tree extends Thread{
15     public void run() {
16         Runtime r = Runtime.getRuntime();
17         Process p = null;
18         try {
19             p = r.exec("tail␣-f␣/home/taz/NetBeansProjects/Master/src/
                   master/Fisei");
20         } catch (IOException ex) {
21             Logger.getLogger(Principal.class.getName()).log(Level.
                   SEVERE, null, ex);
22         }
23         BufferedReader b = new BufferedReader(new InputStreamReader(p.
               getInputStream()));
24         String line = "";
```

```
25          try {
26              while ((line = b.readLine()) != null) {
27                  Principal.llenarTexto(line);
28              }
29          } catch (IOException ex) {
30              Logger.getLogger(Principal.class.getName()).log(Level.
                    SEVERE, null, ex);
31          }
32          try {
33              b.close();
34          } catch (IOException ex) {
35              Logger.getLogger(Principal.class.getName()).log(Level.
                    SEVERE, null, ex);
36          }
37      }
38       public javax.swing.JTextArea txtMensaje1;
39  }
```

- **Clase Principal y Diseño de la Interfaz.**

```
1  import java.io.BufferedReader;
2  import java.io.IOException;
3  import java.io.InputStreamReader;
4  import java.util.logging.Level;
5  import java.util.logging.Logger;
6  /**
7   * @author renato
8   */
9  public class Principal extends javax.swing.JFrame {
10      public Principal() {
11          initComponents();
12      }
13      @SuppressWarnings("unchecked")
14      // <editor-fold defaultstate="collapsed" desc="Generated Code">
15      private void initComponents() {
16      //Inicia el servidor 1
17      private void btnIniciarServer1ActionPerformed(java.awt.event.
            ActionEvent evt) {
18                  Runtime r = Runtime.getRuntime();
19          Process p = null;
20          try {
21              p = r.exec("/usr/local/zookeeper-3.4.6/bin/zkServer.sh␣
                    start␣/usr/local/var/run/zookeeper/data/z1/z1.cfg");
22          } catch (IOException ex) {
```

```
23              Logger.getLogger(Principal.class.getName()).log(Level.
                    SEVERE, null, ex);
24          }
25          try {
26              p.waitFor();
27          } catch (InterruptedException ex) {
28              Logger.getLogger(Principal.class.getName()).log(Level.
                    SEVERE, null, ex);
29          }
30          BufferedReader b = new BufferedReader(new InputStreamReader(p.
                getInputStream()));
31          String line = "";
32          try {
33              while ((line = b.readLine()) != null) {
34                  System.out.println(line);
35                  txtServer1.setText(line);
36              }
37          } catch (IOException ex) {
38              Logger.getLogger(Principal.class.getName()).log(Level.
                    SEVERE, null, ex);
39          }
40          try {
41              b.close();
42          } catch (IOException ex) {
43              Logger.getLogger(Principal.class.getName()).log(Level.
                    SEVERE, null, ex);
44          }
45      }
46          //Detiene el Servidor 1
47      private void btnDetenerServer1ActionPerformed(java.awt.event.
            ActionEvent evt) {
48                  Runtime r = Runtime.getRuntime();
49          Process p = null;
50          try {
51              p = r.exec("/usr/local/zookeeper-3.4.6/bin/zkServer.sh␣stop
                    ␣/usr/local/var/run/zookeeper/data/z1/z1.cfg");
52          } catch (IOException ex) {
53              Logger.getLogger(Principal.class.getName()).log(Level.
                    SEVERE, null, ex);
54          }
55          try {
56              p.waitFor();
57          } catch (InterruptedException ex) {
58              Logger.getLogger(Principal.class.getName()).log(Level.
                    SEVERE, null, ex);
```

```
59          }
60          BufferedReader b = new BufferedReader(new InputStreamReader(p.
                getInputStream()));
61          String line = "";
62          try {
63              while ((line = b.readLine()) != null) {
64                  System.out.println(line);
65                  txtServer1.setText(line);
66              }
67          } catch (IOException ex) {
68              Logger.getLogger(Principal.class.getName()).log(Level.
                    SEVERE, null, ex);
69          }
70          try {
71              b.close();
72          } catch (IOException ex) {
73              Logger.getLogger(Principal.class.getName()).log(Level.
                    SEVERE, null, ex);
74          }
75      }
76          //Inicia el Servidor 2
77      private void btnIniciarServer2ActionPerformed(java.awt.event.
            ActionEvent evt) {
78              Runtime r = Runtime.getRuntime();
79          Process p = null;
80          try {
81              p = r.exec("/usr/local/zookeeper-3.4.6/bin/zkServer.sh␣
                    start␣/usr/local/var/run/zookeeper/data/z2/z2.cfg");
82          } catch (IOException ex) {
83              Logger.getLogger(Principal.class.getName()).log(Level.
                    SEVERE, null, ex);
84          }
85          try {
86              p.waitFor();
87          } catch (InterruptedException ex) {
88              Logger.getLogger(Principal.class.getName()).log(Level.
                    SEVERE, null, ex);
89          }
90          BufferedReader b = new BufferedReader(new InputStreamReader(p.
                getInputStream()));
91          String line = "";
92
93          try {
94              while ((line = b.readLine()) != null) {
95                  System.out.println(line);
```

```
96                  txtServer2.setText(line);
97              }
98          } catch (IOException ex) {
99              Logger.getLogger(Principal.class.getName()).log(Level.
                    SEVERE, null, ex);
100         }
101         try {
102             b.close();
103         } catch (IOException ex) {
104             Logger.getLogger(Principal.class.getName()).log(Level.
                    SEVERE, null, ex);
105         }
106
107     }
108         //Detiene el Servidor 2
109     private void btnDetenerServer2ActionPerformed(java.awt.event.
            ActionEvent evt) {
110             Runtime r = Runtime.getRuntime();
111         Process p = null;
112         try {
113             p = r.exec("/usr/local/zookeeper-3.4.6/bin/zkServer.sh␣stop
                    ␣/usr/local/var/run/zookeeper/data/z2/z2.cfg");
114         } catch (IOException ex) {
115             Logger.getLogger(Principal.class.getName()).log(Level.
                    SEVERE, null, ex);
116         }
117         try {
118             p.waitFor();
119         } catch (InterruptedException ex) {
120             Logger.getLogger(Principal.class.getName()).log(Level.
                    SEVERE, null, ex);
121         }
122         BufferedReader b = new BufferedReader(new InputStreamReader(p.
                getInputStream()));
123         String line = "";
124         try {
125             while ((line = b.readLine()) != null) {
126                 System.out.println(line);
127                 txtServer2.setText(line);
128             }
129         } catch (IOException ex) {
130             Logger.getLogger(Principal.class.getName()).log(Level.
                    SEVERE, null, ex);
131         }
132         try {
```

```
133             b.close();
134         } catch (IOException ex) {
135             Logger.getLogger(Principal.class.getName()).log(Level.
                    SEVERE, null, ex);
136         }
137     }
138         //Inicia el Servidor 3
139     private void btnIniciarServer3ActionPerformed(java.awt.event.
            ActionEvent evt) {
140         Runtime r = Runtime.getRuntime();
141         Process p = null;
142         try {
143             p = r.exec("/usr/local/zookeeper-3.4.6/bin/zkServer.sh␣
                    start␣/usr/local/var/run/zookeeper/data/z3/z3.cfg");
144         } catch (IOException ex) {
145             Logger.getLogger(Principal.class.getName()).log(Level.
                    SEVERE, null, ex);
146         }
147         try {
148             p.waitFor();
149         } catch (InterruptedException ex) {
150             Logger.getLogger(Principal.class.getName()).log(Level.
                    SEVERE, null, ex);
151         }
152         BufferedReader b = new BufferedReader(new InputStreamReader(p.
                getInputStream()));
153         String line = "";
154         try {
155             while ((line = b.readLine()) != null) {
156                 System.out.println(line);
157                 txtServer3.setText(line);
158             }
159         } catch (IOException ex) {
160             Logger.getLogger(Principal.class.getName()).log(Level.
                    SEVERE, null, ex);
161         }
162         try {
163             b.close();
164         } catch (IOException ex) {
165             Logger.getLogger(Principal.class.getName()).log(Level.
                    SEVERE, null, ex);
166         }
167     }
168         //Detiene el servidor 3
```

```
169     private void btnDetenerServer3ActionPerformed(java.awt.event.
            ActionEvent evt) {
170         Runtime r = Runtime.getRuntime();
171         Process p = null;
172         try {
173             p = r.exec("/usr/local/zookeeper-3.4.6/bin/zkServer.sh stop
                    /usr/local/var/run/zookeeper/data/z3/z3.cfg");
174         } catch (IOException ex) {
175             Logger.getLogger(Principal.class.getName()).log(Level.
                    SEVERE, null, ex);
176         }
177         try {
178             p.waitFor();
179         } catch (InterruptedException ex) {
180             Logger.getLogger(Principal.class.getName()).log(Level.
                    SEVERE, null, ex);
181         }
182         BufferedReader b = new BufferedReader(new InputStreamReader(p.
                getInputStream()));
183         String line = "";
184 
185         try {
186             while ((line = b.readLine()) != null) {
187                 System.out.println(line);
188                 txtServer3.setText(line);
189             }
190         } catch (IOException ex) {
191             Logger.getLogger(Principal.class.getName()).log(Level.
                    SEVERE, null, ex);
192         }
193         try {
194             b.close();
195         } catch (IOException ex) {
196             Logger.getLogger(Principal.class.getName()).log(Level.
                    SEVERE, null, ex);
197         }
198     }
199         //Ejecuta Monitor
200     private void btnMonitoreoActionPerformed(java.awt.event.ActionEvent
            evt) {
201         Runtime r = Runtime.getRuntime();
202         Process p = null;
203         try {
204             p = r.exec("/home/taz/NetBeansProjects/Master/src/master/
                    master.sh" );
```

```
205         } catch (IOException ex) {
206             Logger.getLogger(Principal.class.getName()).log(Level.
                SEVERE, null, ex);
207         }
208         try {
209             p.waitFor();
210             Tree t = new Tree();
211             t.start();
212         } catch (InterruptedException ex) {
213             Logger.getLogger(Principal.class.getName()).log(Level.
                SEVERE, null, ex);
214         }
215         BufferedReader b = new BufferedReader(new InputStreamReader(p.
            getInputStream()));
216         String line = "";
217         try {
218             while ((line = b.readLine()) != null) {
219                 System.out.println(line);
220             }
221         } catch (IOException ex) {
222             Logger.getLogger(Principal.class.getName()).log(Level.
                SEVERE, null, ex);
223         }
224         try {
225             b.close();
226         } catch (IOException ex) {
227             Logger.getLogger(Principal.class.getName()).log(Level.
                SEVERE, null, ex);
228         }
229     }
230 }
```

Interface de Usuario

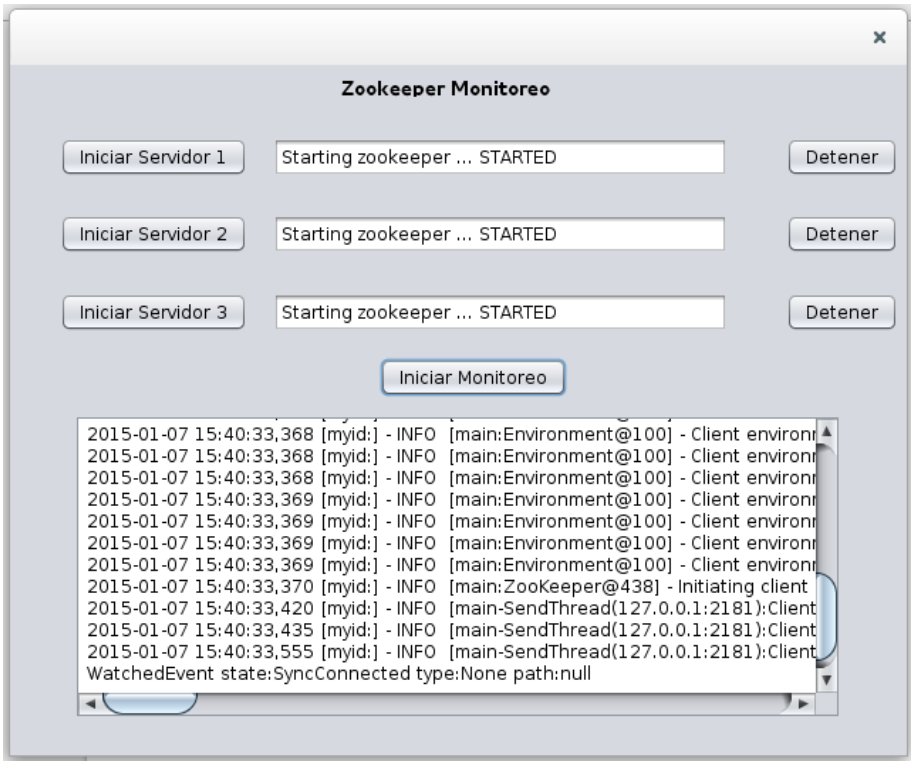


Figura 4.37: Interfaz Monitor Zookeeper
Elaborado por: Renato Toasa

Esta aplicación permite al usuario iniciar y detener los servidores de Zookeeper, además muestra datos de lo que secede en los servidores. Estos datos van a variar segun la actividad que se realiza en los servidores Zookeeper.

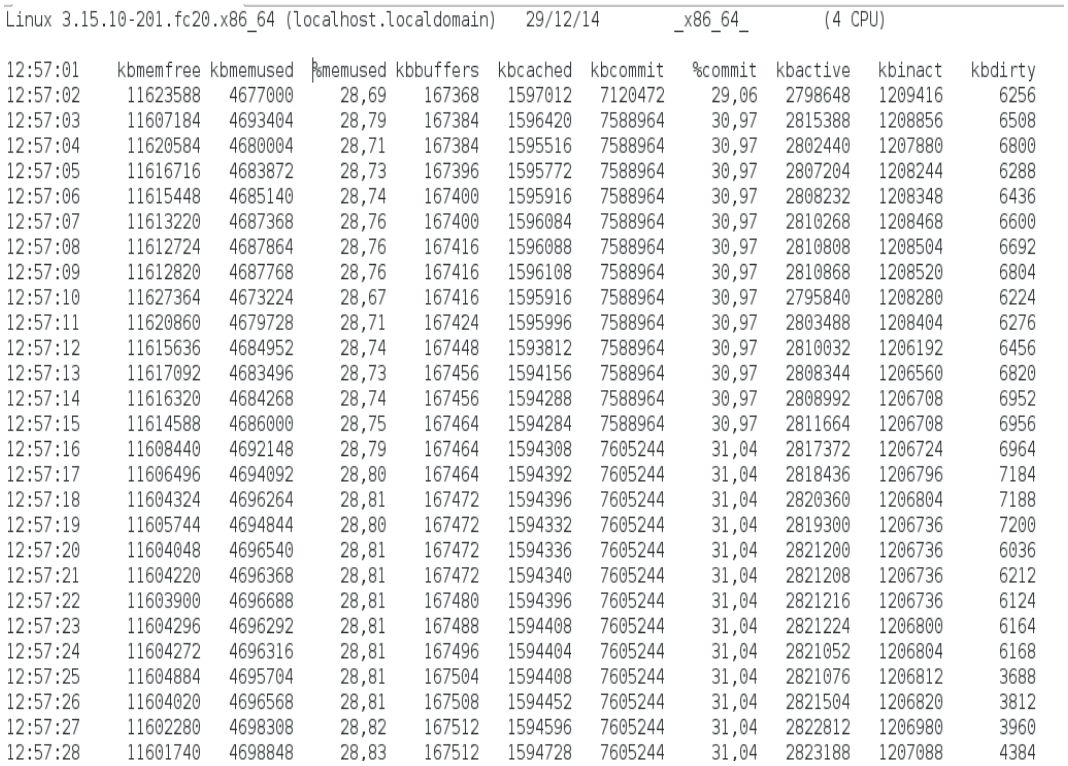
4.9.3. Medición de Recursos utilizados por Zookeeper

Una vez se ha ejecutado la aplicación se procedio a una toma de datos de consumo de recursos del computador como Consumo de CPU , Consumo de Memoria, Consumo de Disco, Consumo de RED (Paquetes de entrada y paquetes de salida). Para lograr esto se utilizó las herramintas SAR para la toma de datos y GNUPLOT para los Gráficos[23].

Se obtuvo los siguientes resultados.

Consumo de Memoria

Los datos capturados para el análisis son la columna cuarta referente al porcentaje de utilización de memoria como se muestra en la siguiente figura.



```
Linux 3.15.10-201.fc20.x86_64 (localhost.localdomain)   29/12/14        _x86_64_        (4 CPU)

12:57:01    kbmemfree kbmemused  %memused kbbuffers  kbcached  kbcommit   %commit  kbactive   kbinact   kbdirty
12:57:02     11623588   4677000     28,69    167368   1597012   7120472     29,06   2798648   1209416      6256
12:57:03     11607184   4693404     28,79    167384   1596420   7588964     30,97   2815388   1208856      6508
12:57:04     11620584   4680004     28,71    167384   1595516   7588964     30,97   2802440   1207880      6800
12:57:05     11616716   4683872     28,73    167396   1595772   7588964     30,97   2807204   1208244      6288
12:57:06     11615448   4685140     28,74    167400   1595916   7588964     30,97   2808232   1208348      6436
12:57:07     11613220   4687368     28,76    167400   1596084   7588964     30,97   2810268   1208468      6600
12:57:08     11612724   4687864     28,76    167416   1596088   7588964     30,97   2810808   1208504      6692
12:57:09     11612820   4687768     28,76    167416   1596108   7588964     30,97   2810868   1208520      6804
12:57:10     11627364   4673224     28,67    167416   1595916   7588964     30,97   2795840   1208280      6224
12:57:11     11620860   4679728     28,71    167424   1595996   7588964     30,97   2803488   1208404      6276
12:57:12     11615636   4684952     28,74    167448   1593812   7588964     30,97   2810032   1206192      6456
12:57:13     11617092   4683496     28,73    167456   1594156   7588964     30,97   2808344   1206560      6820
12:57:14     11616320   4684268     28,74    167456   1594288   7588964     30,97   2808992   1206708      6952
12:57:15     11614588   4686000     28,75    167464   1594284   7588964     30,97   2811664   1206708      6956
12:57:16     11608440   4692148     28,79    167464   1594308   7605244     31,04   2817372   1206724      6964
12:57:17     11606496   4694092     28,80    167464   1594392   7605244     31,04   2818436   1206796      7184
12:57:18     11604324   4696264     28,81    167472   1594396   7605244     31,04   2820360   1206804      7188
12:57:19     11605744   4694844     28,80    167472   1594332   7605244     31,04   2819300   1206736      7200
12:57:20     11604048   4696540     28,81    167472   1594336   7605244     31,04   2821200   1206736      6036
12:57:21     11604220   4696368     28,81    167472   1594340   7605244     31,04   2821208   1206736      6212
12:57:22     11603900   4696688     28,81    167480   1594396   7605244     31,04   2821216   1206736      6124
12:57:23     11604296   4696292     28,81    167488   1594408   7605244     31,04   2821224   1206800      6164
12:57:24     11604272   4696316     28,81    167496   1594404   7605244     31,04   2821052   1206804      6168
12:57:25     11604884   4695704     28,81    167504   1594408   7605244     31,04   2821076   1206812      3688
12:57:26     11604020   4696568     28,81    167508   1594452   7605244     31,04   2821504   1206820      3812
12:57:27     11602280   4698308     28,82    167512   1594596   7605244     31,04   2822812   1206980      3960
12:57:28     11601740   4698848     28,83    167512   1594728   7605244     31,04   2823188   1207088      4384
```

Figura 4.38: Muestra de datos capturados para la memoria

Ahora se muestra a continuación la comparación de uso de memoria entre Centos y Fedora cuando los servidores de zookeeeper estan funcionando correctamente o cuando hay caídas de los mismos.

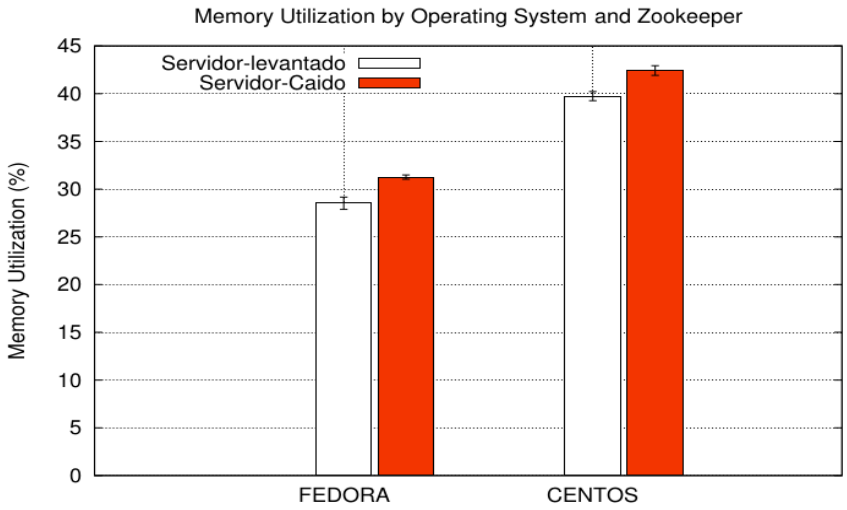


Figura 4.39: Gráfico de consumo de memoria por Sistema Operativo
Elaborado por: Renato Toasa

De las gráficas se púede concluir que:

- Fedora tiene un porcentaje de uso de memoria de 28 % cuando los servidores de Zookeeper estan levantados y un porcentaje de 31 % cuando los servidores sufren caídas.
- Centos tiene un porcentaje de uso de memor a de 39.5 % cuando los servidores de Zookeeper estan levantados y un orcentaje de 42.5 % cuando los servidores sufren caídas.

En la gráfica se puede evidenciar claramente el incremento de consumo de recursos. Siendo fedora el sistema operativo que consume la menor cantidad de memoria.

Consumo de CPU

Para la CPU se referencia el % de utilización como se muestra en la siguiente figura.

```
Linux 3.15.10-201.fc20.x86_64 (localhost.localdomain)   29/12/14        _x86_64_

12:14:30        CPU     %user     %nice   %system   %iowait    %steal     %idle
12:14:31        all      3,25      0,25      3,00      0,00      0,00     93,50
12:14:32        all      9,55      0,00      3,02      3,02      0,00     84,42
12:14:33        all      2,77      0,00      2,52      0,00      0,00     94,71
12:14:34        all      2,25      0,00     10,25      1,25      0,00     86,25
12:14:35        all      2,52      0,00      3,02      0,00      0,00     94,46
12:14:36        all      2,76      0,00      4,01      1,25      0,00     91,98
12:14:37        all      3,03      0,00      3,28      1,01      0,00     92,68
12:14:38        all      3,26      0,00      3,51      0,25      0,00     92,98
12:14:39        all      3,02      0,00      3,77      1,01      0,00     92,21
12:14:40        all      2,54      0,00      2,54      0,00      0,00     94,92
12:14:41        all      2,78      0,00      3,54      0,00      0,00     93,69
12:14:42        all      2,76      0,00      3,76      0,00      0,00     93,48
12:14:43        all      3,26      0,00      3,26      0,00      0,00     93,48
12:14:44        all      2,78      0,00      3,03      0,00      0,00     94,19
12:14:45        all      3,02      0,00      2,76      1,51      0,00     92,71
12:14:46        all      3,24      0,25      3,74      1,00      0,00     91,77
12:14:47        all      2,54      0,00      2,79      0,00      0,00     94,67
12:14:48        all      2,99      0,00      3,49      0,25      0,00     93,27
12:14:49        all      7,61      0,00      2,28      0,00      0,00     90,10
12:14:50        all      3,47      0,00      2,98      1,49      0,00     92,06
12:14:51        all      2,53      0,00      2,03      0,00      0,00     95,44
12:14:52        all      6,03      0,00      3,27      0,25      0,00     90,45
12:14:53        all      3,79      0,00      2,78      0,00      0,00     93,43
12:14:54        all     13,75      0,00      3,00      3,25      0,00     80,00
12:14:55        all      3,29      0,00      2,03      0,51      0,00     94,18
12:14:56        all      6,55      0,00      2,52      2,77      0,00     88,16
12:14:57        all      8,00      0,00      3,00      0,00      0,00     89,00
12:14:58        all      6,52      0,00      2,76      0,25      0,00     90,48
```

Figura 4.40: Muestra del % de utilización de la CPU
Elaborado por: Renato Toasa

De igual forma se muestra la comparación de uso de CPU entre Centos y Fedora.

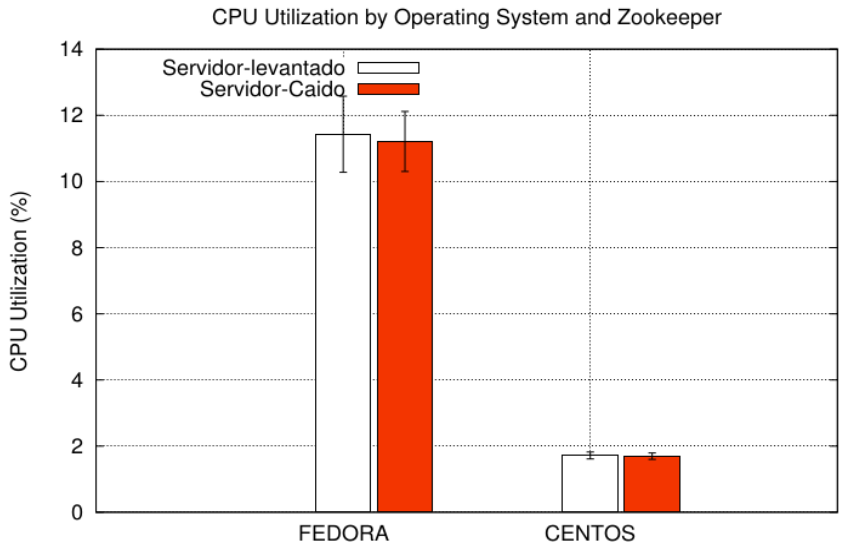


Figura 4.41: Gráfico de consumo de CPU por Sistema Operativo
Elaborado por: Renato Toasa

Del gráfico se puede concluir que:

- Fedora alcanza el 11.5 % de utilización del CPU cuando los servidores estan levantados y un 11 % de utilización cuando los servidores sufren caídas.
- Centos alcanza el 1.8 % de utilización del CPU cuando los servidores estan levantados y cuando los servidores sufren caídas.

En la gráfica se puede evidenciar claramente el incremento de consumo de recursos. Siendo Centos el sistema operativo que consume la menor cantidad de CPU.

Consumo de Disco

Para analizar el rendimiento del disco duro se tomado el % de utilización del mismo como se muestra en la siguiente figura.

```
Linux 3.15.10-201.fc20.x86_64 (localhost.localdomain)   29/12/14        _x86_64_        (4 CPU)

12:59:54          DEV       tps  rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz     await     svctm     %util
12:59:55       dev8-0      3,00      0,00    128,00     42,67      0,03      8,67      8,67      2,60
12:59:55      dev11-0      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00
12:59:55     dev253-0      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00
12:59:55     dev253-1      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00
12:59:55     dev253-2     16,00      0,00    128,00      8,00      0,03      1,62      1,62      2,60

12:59:55          DEV       tps  rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz     await     svctm     %util
12:59:56       dev8-0     32,00      0,00    392,00     12,25      0,34     10,59      3,34     10,70
12:59:56      dev11-0      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00
12:59:56     dev253-0      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00
12:59:56     dev253-1     40,00      0,00    296,00      7,40      0,32      8,07      1,98      7,90
12:59:56     dev253-2     12,00      0,00     96,00      8,00      0,03      2,33      2,33      2,80

12:59:56          DEV       tps  rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz     await     svctm     %util
12:59:57       dev8-0      2,00     16,00      0,00      8,00      0,03     15,50     15,50      3,10
12:59:57      dev11-0      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00
12:59:57     dev253-0      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00
12:59:57     dev253-1      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00
12:59:57     dev253-2      2,00     16,00      0,00      8,00      0,03     15,50     15,50      3,10

12:59:57          DEV       tps  rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz     await     svctm     %util
12:59:58       dev8-0      7,00      2,00    242,00     34,86      0,04      5,14      5,00      3,50
12:59:58      dev11-0      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00
12:59:58     dev253-0      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00
12:59:58     dev253-1      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00
12:59:58     dev253-2      7,00      2,00    242,00     34,86      0,04      5,14      5,00      3,50
```

Figura 4.42: Muestra del % de utilización del disco duro
Elaborado por: Renato Toasa

De igual forma se muestra la comparación de uso de CPU entre Centos y Fedora.

De la gráfica se concluye:

- Fedora alcanza el 2.75 % de utilización de Disco cuando los servidores estan levantados, y un 2.1 % cuanso los servidores sufren caídas.

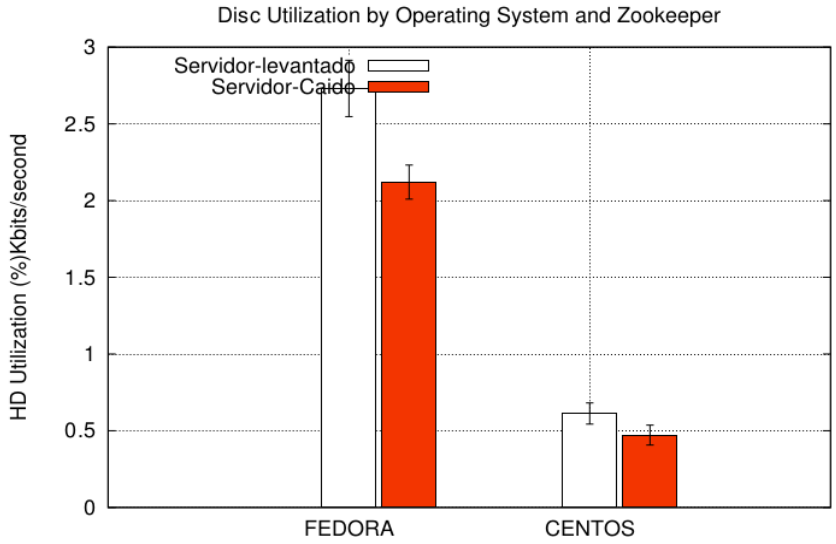


Figura 4.43: Gráfico de consumo de Disco Duro por Sistema Operativo
Elaborado por: Renato Toasa

- Centos alcanza 0.6 % de utilización de Disco cuando los servidores estan levantados, y un 0.48 % cuanso los servidores sufren caídas.

De la misma forma se puede ver que en la gráfica se puede evidenciar claramente el incremento de consumo de recursos. Siendo Centos el sistema operativo que consume la menor cantidad de Disco.

Consumo de Red - Paquetes transmitidos

Para poder analizar la red se a tomado la columna 6 de la captura referente a la tasa de transmisión en KB por segundo como se muestra en la siguiente figura.

```
Linux 3.15.10-201.fc20.x86_64 (localhost.localdomain)   29/12/14        _x86_64_        (4 CPU)

12:22:23        IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s   %ifutil
12:22:24       wlp2s0      0,00      2,00      0,00      0,17      0,00      0,00      0,00      0,00
12:22:24         p8p1      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00
12:22:24           lo     18,00     18,00      1,17      1,17      0,00      0,00      0,00      0,00

12:22:23        IFACE   rxerr/s   txerr/s    coll/s  rxdrop/s  txdrop/s  txcarr/s  rxfram/s  rxfifo/s  txfifo/s
12:22:24       wlp2s0      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00
12:22:24         p8p1      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00
12:22:24           lo      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00

12:22:23      call/s retrans/s    read/s   write/s  access/s  getatt/s
12:22:24        0,00      0,00      0,00      0,00      0,00      0,00

12:22:23     scall/s badcall/s  packet/s     udp/s     tcp/s     hit/s    miss/s   sread/s  swrite/s saccess/s sgetatt/s
12:22:24        0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00

12:22:23      totsck    tcpsck    udpsck    rawsck   ip-frag    tcp-tw
12:22:24         662        13        11         0         0         1

12:22:23      irec/s  fwddgm/s    idel/s     orq/s   asmrq/s   asmok/s  fragok/s fragcrt/s
12:22:24       18,00      0,00     18,00     19,00      0,00      0,00      0,00      0,00

12:22:23   ihdrerr/s iadrerr/s iukwnpr/s   idisc/s   odisc/s   onort/s    asmf/s   fragf/s
12:22:24        0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00

12:22:23      imsg/s    omsg/s    iech/s   iechr/s    oech/s   oechr/s     itm/s    itmr/s     otm/s    otmr/s  iadrmk/s iadrmkr/s  oadrmk/s
oadrmkr/s
12:22:24        0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00
0,00
```

Figura 4.44: Muestra de la tasa de transmisión de KB en la red
Elaborado por: Renato Toasa

De igual forma se muestra la comparación de uso de CPU entre Centos y Fedora.

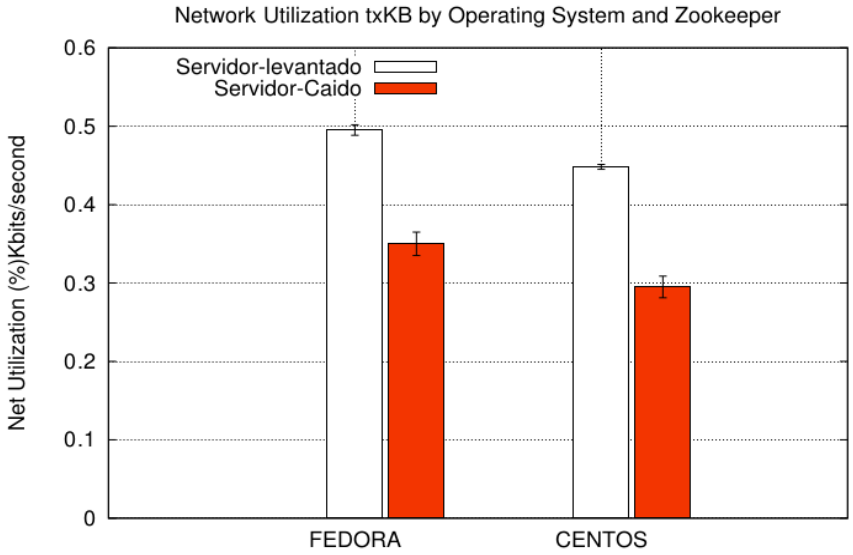


Figura 4.45: Gráfico de consumo de datos Transmitidos por RED por Sistema Operativo
Elaborado por: Renato Toasa

En el análisis podemos concluir que los paquetes transmitidos por cada Sistema operativo son

- Fedora alcanza el 0.49 % cuando los servidores estan levantados y un 0.35 % cuando los servidores sufren caídas.
- Centos alcanza el 0.45 % cuando los servidores estan levantados y un 0.29 % cuando los servidores sufren caídas.

De la misma forma se puede ver que en la gráfica se puede evidenciar claramente el incremento de consumo de recursos. Siendo Centos el sistema operativo que consume la menor RED cuando se transmiten datos.

Consumo de Red - Paquetes Recibidos

Para poder analizar la red se a tomado la columna 5 de la captura referente a la tasa de recepción en KB por segundo.

```
Linux 3.15.10-201.fc20.x86_64 (localhost.localdomain)   29/12/14        _x86_64_        (4 CPU)

12:22:23       IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s   %ifutil
12:22:24      wlp2s0      0,00      2,00      0,00      0,17      0,00      0,00      0,00      0,00
12:22:24        p8p1      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00
12:22:24          lo     18,00     18,00      1,17      1,17      0,00      0,00      0,00      0,00

12:22:23       IFACE   rxerr/s   txerr/s    coll/s  rxdrop/s  txdrop/s  txcarr/s  rxfram/s  rxfifo/s  txfifo/s
12:22:24      wlp2s0      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00
12:22:24        p8p1      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00
12:22:24          lo      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00

12:22:23      call/s retrans/s    read/s   write/s  access/s  getatt/s
12:22:24        0,00      0,00      0,00      0,00      0,00      0,00

12:22:23     scall/s badcall/s  packet/s     udp/s     tcp/s     hit/s    miss/s   sread/s  swrite/s saccess/s sgetatt/s
12:22:24        0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00

12:22:23      totsck    tcpsck    udpsck    rawsck   ip-frag    tcp-tw
12:22:24         662        13        11         0         0         1

12:22:23      irec/s  fwddgm/s    idel/s     orq/s   asmrq/s   asmok/s  fragok/s fragcrt/s
12:22:24       18,00      0,00     18,00     19,00      0,00      0,00      0,00      0,00

12:22:23   ihdrerr/s iadrerr/s iukwnpr/s  idisc/s  odisc/s  onort/s   asmf/s   fragf/s
12:22:24        0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00

12:22:23      imsg/s    omsg/s    iech/s   iechr/s    oech/s   oechr/s    itm/s   itmr/s    otm/s   otmr/s  iadrmk/s iadrmkr/s  oadrmk/s
oadrmkr/s
12:22:24        0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00      0,00
0,00
```

Figura 4.46: Muestra de la Tasa de Recepción de KB en la red
Elaborado por: Renato Toasa

De igual forma se muestra la comparación de uso de CPU entre Centos y Fedora.

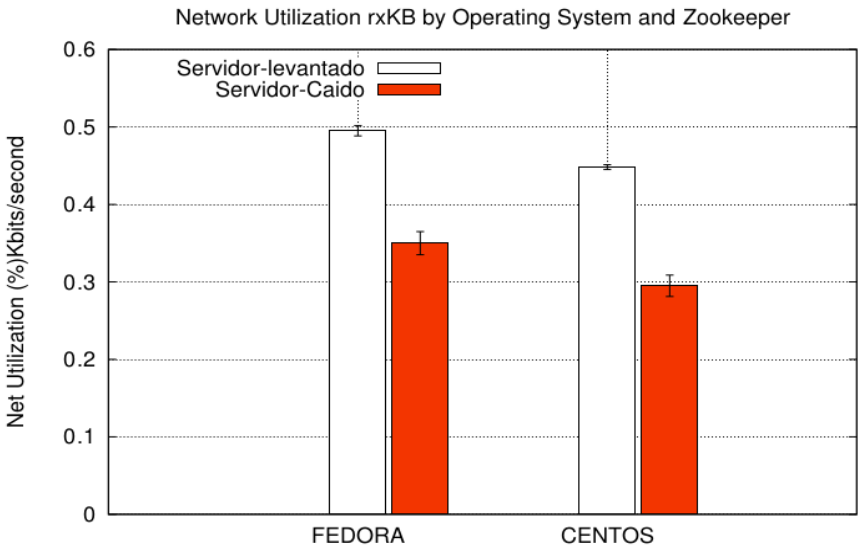


Figura 4.47: Gráfico de Consumo de Datos Recibidos por RED por Sistema Operativo

Elaborado por: Renato Toasa

En el análisis podemos concluir que los paquetes transmitidos por cada Sistema operativo son

- Fedora alcanza el 0.49 % cuando los servidores estan levantados y un 0.35 % cuando los servidores sufren caídas.
- Centos alcanza el 0.45 % cuando los servidores estan levantados y un 0.29 % cuando los servidores sufren caídas.

De la misma forma se puede ver que en la gráfica se puede evidenciar claramente el incremento de consumo de recursos. Siendo Centos el sistema operativo que consume la menor cantidad de RED cuando se transmiten datos.

CAPÍTULO 5

Conclusiones y Recomendaciones

5.1. CONCLUSIONES

- Se comprobó que Zookeeper es un servicio de coordinación estable, simple y de alto rendimiento que proporciona las herramientas necesarias para administrar aplicaciones distribuidas sin preocuparse por la pérdida de informacíon con la que se trabaja y la inconsistencia de los datos.
- Con la simulación programada se determina que ZooKeeper alcanza valores de rendimiento de cientos de operaciones y peticiones por segundo para las cargas de trabajo, además muestra información de lo que sucede en cada uno de los servidores mediante el uso de observadores y notificaciones.
- En el escenario utilizado se verificó que el consumo de recursos por parte de Zookeeper es mínimo además es multiplataforma lo que asegura el correcto funcionamiento de las aplicaciones que lo usen ya sea con fines académicos o fines empresariales.
- Zookeeper ofrece una gran modularidad lo que permite utilizar tanto el API(*Application Programming Interface*) completa o utilizar por separado alguna de sus características para optimizar algún proceso en funcionamiento o ya estructurado.
- Mediante la pruebas realizadas y el análisis de las gráficas obtenidas se puede asegurar que centos es el sistema operativo adecuado para el correcto funcionamiento de zookeeper.

5.2. RECOMENDACIONES

- Se recomienda utilizar Zookeeper cuando las cargas de trabajo sean altas o los procesos distribuidos sean concurrentes, ya que el usuario podrá ordenarlos y manejarlos de una manera fácil, de esta forma se logrará evitar que el sistema colapse con tanto proceso activo.

- Al momento de la búsqueda de información se recomienda hacerlo en bibliotecas digitales de gran impacto como: IEEE Xplore, Springer, Proquest ya que los artículos y libros que exponen estos sitios contienen información muy confiable sobre un tema específico.

- En la programación con Zookeeper se recomienda utilizar clases con el fin de mejorar la modularización y la reutilización del código así como su rendimiento en cuanto al consumo de recursos hardware y red se refieren.

- Para la toma de datos de los recursos del computador es recomendable cerrar otras aplicaciones que podrían utilizar los recursos del computador, de esta forma evidenciar el consumo que se genera con Zookeeper.

Bibliografía

[1] S. Chaudhuri, "More choices allow more faults: Set consensus problems in totally asynchronous systems," *Information and Computation*, vol. 105, no. 1, pp. 132 – 158, 1993.

[2] T. A. S. Foundation, "Apache zookeeper." http://zookeeper.apache.org, 2010.

[3] G. Storage. https://developers.google.com/storage/docs/overview.

[4] F. Junqueira and B. Reed, *Zookeeper Distributed Process Coordination*. Oreilly, 2013.

[5] E. J. y. J. T. Sergio Arevalo, Carlos Herrera, "Eventual election of multiple leaders for solving consensus in anonymous systems," *distributed systems*, vol. 00, pp. 1–8, 2014.

[6] P. Hunt, F. P. J. Mahadev Konar, and B. Reed, "Zookeeper: Wait-free coordination for internet-scale systems," *distributed systems*, vol. 00, pp. 1–14, 2010.

[7] S. Bespamyatnikh, "An $o(n \log n)$ algorithm for the Zoo-keeper's problem," vol. 24, no. 2, pp. 63–74, 2002.

[8] A. Hadoop, "Bookkeeper getting started guide." http://zookeeper.apache.org/doc/r3.4.2/bookkeeperProgrammer.html, Marzo 2012.

[9] K. Ting, *Building an Impenetrable ZooKeeper*. Open Source Convention (OSCON).

[10] S. A. Vinuales, "Tolerancia a fallos en sistemas distribuidos mediante replicacion de procesos," Master's thesis, UNiversidad Politecnica de Madrid, 1998.

[11] A. W. Services, "Amazon simple queue service." http://aws.amazon.com/es/sqs/.

[12] N. Schiper and S. Toueg, "A robust and lightweight stable leader election service for dynamic systems," in *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*, pp. 207–216, June 2008.

[13] A. Sherman, P. A. Lisiecki, A. Berkheimer, and J. Wein, "Acms: The akamai configuration management system," in *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, (Berkeley, CA, USA), pp. 245–258, USENIX Association, 2005.

[14] R. Santamaria, "Sistemas distribuidos coordinacion y acuerdo." http://vis.usal.es/rodrigo/documentos/aso/teoria/5-coordinaci2014.

[15] S. Leberknight, "Distributed coordination with zookeeper." https://www.altamiracorp.com/blog/employee-posts/distributed-coordination-with-zookeeper-part-3-group-membership-example, 06 2013.

[16] P. Nobar, D. Tien, and Moore, "Basic metrics for performance estimation of computers," vol. 338-341 vol.1, p. 4, Oct 2002.

[17] HardZone, "2013." http://hardzone.es/programas-para-testear-monitorizar-y-comprobar-el-rendimiento-de-tu-pc/.

[18] K. Winston, "Midiendo el rendimiento del sistema con sar." http://mundogeek.net/traducciones/midiendo-el-rendimiento-del-sistema-con-SAR.htm, June 2010.

[19] IBM, "Easy system monitoring with sar." http://www.ibm.com/developerworks/aix/library/au-unix-perfmonsar.html, February 2006.

[20] Matplotlib. http://matplotlib.org/, 2012.

[21] P. K. Janert, *Gnuplot in Action Understanding Data with Graphs*. Manning Publications Co., 2010.

[22] S. Skeirik, R. Bobba, and J. Meseguer, "Formal analysis of fault-tolerant group key management using zookeeper," in *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pp. 636–641, May 2013.

[23] R. Nogales, “El uso de buenas practicas de programacion en el algoritmo de consenso y su incidencia en el consumo de recursos de hardware y red para el proyecto cloud-cedia de la universidad tecnica de ambato,” Master’s thesis, UTA, 2015.

Anexos y Apéndices

- **Código para la toma de datos usando SAR.**

```
#!/bin/bash
Path_toma='/home/taz/Tomas1/'
Path_java='//home/taz/NetBeansProjects/Master/src/master/'
Fecha=$(date +%Y-%m-%d-%H:%M)
Nombre_archivo1='CPU'
Nombre_archivo2='Mem'
Nombre_archivo3='HD'
Nombre_archivo4='RED'

echo ..... Inicia sar
sleep 5s

sar -u 1 60 > "$Path_toma$Nombre_archivo1$Fecha" &
sar -r 1 60 > "$Path_toma$Nombre_archivo2$Fecha" &
sar -d 1 60 > "$Path_toma$Nombre_archivo3$Fecha" &
sar -n ALL 1 60 > "$Path_toma$Nombre_archivo4$Fecha" &


echo "$Path_toma$Nombre_archivo$Fecha"
bg
echo ..... Inicia Monitoreo

sleep 1s
cd $Path_java
ZOOBINDIR="/usr/local/zookeeper-3.4.6/bin"
. "$ZOOBINDIR"/zkEnv.sh
cd /home/taz/NetBeansProjects/Master/src/master && java -cp $CLASSPATH
    Master 127.0.0.1:2181
```

- **Código para obtener la Media de las tomas de CPU aplicando la Distribución T-Student.**

```
#!/bin/bash
FILES=`ls CPU*`
TMP_VALUES="cpu_valuesV1_Fedora.dat"
rm -f $TMP_VALUES
for FILE in $FILES
do
cat $FILE | sed "s/,/\./g" | awk 'BEGIN{}{}END{printf("%3.3f\n",(100-
    $NF))}' >> $TMP_VALUES
done
```

```
9  cat $TMP_VALUES | awk 'BEGIN{i=0;tStudent99[1]=31.82;tStudent99
       [2]=6.965;tStudent99[3]=4.541;tStudent99[4]=3.747;tStudent99
       [5]=3.365;tStudent99[6]=3.143;tStudent99[7]=2.998;tStudent99
       [8]=2.896;tStudent99[9]=2.821;tStudent99[10]=2.764;tStudent99
       [11]=2.718;tStudent99[12]=2.681;tStudent99[13]=2.650;tStudent99
       [14]=2.624;tStudent99[15]=2.602;tStudent99[16]=2.583;tStudent99
       [17]=2.567;tStudent99[18]=2.552;tStudent99[19]=2.539;tStudent99
       [20]=2.528;tStudent99[21]=2.518;tStudent99[22]=2.508;tStudent99
       [23]=2.500;tStudent99[24]=2.492;tStudent99[25]=2.485;tStudent99
       [26]=2.479;tStudent99[27]=2.473;tStudent99[28]=2.467;tStudent99
       [29]=2.462;tStudent99[30]=2.457;Z99=2.575}{
10 i++;
11 values[i]=$1;
12 }END{
13 N=i;
14 for (i=1;i<=N;i++) {
15      valuesSum+=values[i];
16 }
17 valuesAvg=valuesSum/N;
18 valuesVar=0;
19 for (i=1;i<=N;i++) {
20      valuesDiff=values[i]-valuesAvg;
21      valuesVar+=(valuesDiff*valuesDiff);
22 }
23 valuesVar=valuesVar/N;
24 if (N<=30) {
25    valuesI = tStudent99[N-1] *sqrt(valuesVar)/sqrt(N);
26 } else {
27    valuesI = Z99 *sqrt(valuesVar)/sqrt(N);
28 }
29 printf("%10.10f %10.10f\n",valuesAvg,valuesI);
30
31 }' > "cpu_V1_Fedora.dat"
32 #rm -f  $TMP_VALUES
```

- **Código para obtener la Media de las tomas de Memoria aplicando la Distribución T-Student.**

```
1  #!/bin/bash
2  FILES=`ls Mem*`
3  TMP_VALUES="mem_valuesV1_Fedora.dat"
4  rm -f $TMP_VALUES
5  for FILE in $FILES
6  do
```

```
7  cat $FILE | sed "s/,/\./g" | awk 'BEGIN{}{}END{printf("%3.3f\n",($4))}'
       >> $TMP_VALUES
8  done
9  cat $TMP_VALUES | awk 'BEGIN{i=0;tStudent99[1]=31.82;tStudent99
       [2]=6.965;tStudent99[3]=4.541;tStudent99[4]=3.747;tStudent99
       [5]=3.365;tStudent99[6]=3.143;tStudent99[7]=2.998;tStudent99
       [8]=2.896;tStudent99[9]=2.821;tStudent99[10]=2.764;tStudent99
       [11]=2.718;tStudent99[12]=2.681;tStudent99[13]=2.650;tStudent99
       [14]=2.624;tStudent99[15]=2.602;tStudent99[16]=2.583;tStudent99
       [17]=2.567;tStudent99[18]=2.552;tStudent99[19]=2.539;tStudent99
       [20]=2.528;tStudent99[21]=2.518;tStudent99[22]=2.508;tStudent99
       [23]=2.500;tStudent99[24]=2.492;tStudent99[25]=2.485;tStudent99
       [26]=2.479;tStudent99[27]=2.473;tStudent99[28]=2.467;tStudent99
       [29]=2.462;tStudent99[30]=2.457;Z99=2.575}{
10 i++;
11 values[i]=$1;
12 }END{
13 N=i;
14 for (i=1;i<=N;i++) {
15      valuesSum+=values[i];
16 }
17 valuesAvg=valuesSum/N;
18 valuesVar=0;
19 for (i=1;i<=N;i++) {
20      valuesDiff=values[i]-valuesAvg;
21      valuesVar+=(valuesDiff*valuesDiff);
22 }
23 valuesVar=valuesVar/N;
24 if (N<=30) {
25    valuesI = tStudent99[N-1] *sqrt(valuesVar)/sqrt(N);
26 } else {
27    valuesI = Z99 *sqrt(valuesVar)/sqrt(N);
28 }
29 printf("%10.5f %10.5f\n",valuesAvg,valuesI);
30 }' > "mem_V1_Fedora.dat"
31 #rm -f  $TMP_VALUES
```

- **Código para obtener la Media de las tomas de Disco aplicando la Distribución T-Student.**

```
1  #!/bin/bash
2  FILES=`ls HD*`
3  TMP_VALUES="hd_valuesV1_Fedora.dat"
4  rm -f $TMP_VALUES
```

```
5  for FILE in $FILES
6  do
7  cat $FILE | sed "s/,/\./g" |sed '/ dev253-1/d'| awk 'BEGIN{}{}
       END{printf("%3.3f\n",($NF))}' >> $TMP_VALUES
8  done
9  cat $TMP_VALUES | awk 'BEGIN{i=0;tStudent99[1]=31.82;tStudent99
       [2]=6.965;tStudent99[3]=4.541;tStudent99[4]=3.747;tStudent99
       [5]=3.365;tStudent99[6]=3.143;tStudent99[7]=2.998;tStudent99
       [8]=2.896;tStudent99[9]=2.821;tStudent99[10]=2.764;tStudent99
       [11]=2.718;tStudent99[12]=2.681;tStudent99[13]=2.650;
       tStudent99[14]=2.624;tStudent99[15]=2.602;tStudent99
       [16]=2.583;tStudent99[17]=2.567;tStudent99[18]=2.552;
       tStudent99[19]=2.539;tStudent99[20]=2.528;tStudent99
       [21]=2.518;tStudent99[22]=2.508;tStudent99[23]=2.500;
       tStudent99[24]=2.492;tStudent99[25]=2.485;tStudent99
       [26]=2.479;tStudent99[27]=2.473;tStudent99[28]=2.467;
       tStudent99[29]=2.462;tStudent99[30]=2.457;Z99=2.575}{
10 i++;
11 values[i]=$1;
12 }END{
13 N=i;
14 for (i=1;i<=N;i++) {
15      valuesSum+=values[i];
16 }
17 valuesAvg=valuesSum/N;
18 valuesVar=0;
19 for (i=1;i<=N;i++) {
20      valuesDiff=values[i]-valuesAvg;
21      valuesVar+=(valuesDiff*valuesDiff);
22 }
23 valuesVar=valuesVar/N;
24 if (N<=30) {
25     valuesI = tStudent99[N-1] *sqrt(valuesVar)/sqrt(N);
26 } else {
27     valuesI = Z99 *sqrt(valuesVar)/sqrt(N);
28 }
29 printf("%10.10f␣%10.10f\n",valuesAvg,valuesI);
30 }' > "hd_V1_Fedora.dat"
31 #rm -f  $TMP_VALUES
```

- **Código para obtener la Media de las tomas de Red en los paquetes transmitidos aplicando la Distribución T-Student.**

```
#!/bin/bash
FILES=`ls RED*`
TMP_VALUES="net_tx_valuesV1_Fedora.dat"
rm -f $TMP_VALUES
for FILE in $FILES
do
#cat $FILE | sed "s/,/\./g" | awk 'BEGIN{}{}END{printf("%3.3f\n",(100-$NF))}' >> $TMP_VALUES
cat $FILE| sed "s/,/\./g" |egrep -A3 IFACE|egrep "Media:"|egrep -v ^$|egrep -A3 rxkB|egrep lo|awk '{print $6}' >> $TMP_VALUES
done
cat $TMP_VALUES | awk 'BEGIN{i=0;tStudent99[1]=31.82;tStudent99[2]=6.965;tStudent99[3]=4.541;tStudent99[4]=3.747;tStudent99[5]=3.365;tStudent99[6]=3.143;tStudent99[7]=2.998;tStudent99[8]=2.896;tStudent99[9]=2.821;tStudent99[10]=2.764;tStudent99[11]=2.718;tStudent99[12]=2.681;tStudent99[13]=2.650;tStudent99[14]=2.624;tStudent99[15]=2.602;tStudent99[16]=2.583;tStudent99[17]=2.567;tStudent99[18]=2.552;tStudent99[19]=2.539;tStudent99[20]=2.528;tStudent99[21]=2.518;tStudent99[22]=2.508;tStudent99[23]=2.500;tStudent99[24]=2.492;tStudent99[25]=2.485;tStudent99[26]=2.479;tStudent99[27]=2.473;tStudent99[28]=2.467;tStudent99[29]=2.462;tStudent99[30]=2.457;Z99=2.575}
i++;
values[i]=$1;
}END{
N=i;
for (i=1;i<=N;i++) {
    valuesSum+=values[i];
}
valuesAvg=valuesSum/N;
valuesVar=0;
for (i=1;i<=N;i++) {
    valuesDiff=values[i]-valuesAvg;
    valuesVar+=(valuesDiff*valuesDiff);
}
valuesVar=valuesVar/N;
if (N<=30) {
   valuesI = tStudent99[N-1] *sqrt(valuesVar)/sqrt(N);
} else {
   valuesI = Z99 *sqrt(valuesVar)/sqrt(N);
}
printf("%10.10f %10.10f\n",valuesAvg,valuesI);
```

```
31 }' > "net_tx_V1_Fedora.dat"
32 #rm -f  $TMP_VALUES
```

- **Código para obtener la Media de las tomas de Red en los paquetes recibidos aplicando la Distribución T-Student.**

```
1  #!/bin/bash
2
3  FILES=`ls RED*`
4  TMP_VALUES="net_rx_valuesV1_Fedora.dat"
5
6  rm -f $TMP_VALUES
7
8  for FILE in $FILES
9  do
10
11 #cat $FILE | sed "s/,/\./g" | awk 'BEGIN{}{}END{printf("%3.3f\n",(100-
       $NF))}' >> $TMP_VALUES
12
13 cat $FILE | sed "s/,/\./g" |egrep -A3  IFACE|egrep "Media:"|egrep -v ^$
       |egrep -A3 rxkB|egrep lo |awk '{print $5}' >> $TMP_VALUES
14
15 done
16
17 cat $TMP_VALUES | awk 'BEGIN{i=0;tStudent99[1]=31.82;tStudent99
       [2]=6.965;tStudent99[3]=4.541;tStudent99[4]=3.747;tStudent99
       [5]=3.365;tStudent99[6]=3.143;tStudent99[7]=2.998;tStudent99
       [8]=2.896;tStudent99[9]=2.821;tStudent99[10]=2.764;tStudent99
       [11]=2.718;tStudent99[12]=2.681;tStudent99[13]=2.650;tStudent99
       [14]=2.624;tStudent99[15]=2.602;tStudent99[16]=2.583;tStudent99
       [17]=2.567;tStudent99[18]=2.552;tStudent99[19]=2.539;tStudent99
       [20]=2.528;tStudent99[21]=2.518;tStudent99[22]=2.508;tStudent99
       [23]=2.500;tStudent99[24]=2.492;tStudent99[25]=2.485;tStudent99
       [26]=2.479;tStudent99[27]=2.473;tStudent99[28]=2.467;tStudent99
       [29]=2.462;tStudent99[30]=2.457;Z99=2.575}{
18
19 i++;
20 values[i]=$1;
21
22
23 }END{
24
25 N=i;
26
27 for (i=1;i<=N;i++) {
```

```
28      valuesSum+=values[i];
29  }
30
31  valuesAvg=valuesSum/N;
32  valuesVar=0;
33
34  for (i=1;i<=N;i++) {
35      valuesDiff=values[i]-valuesAvg;
36      valuesVar+=(valuesDiff*valuesDiff);
37  }
38
39  valuesVar=valuesVar/N;
40
41  if (N<=30) {
42     valuesI = tStudent99[N-1] *sqrt(valuesVar)/sqrt(N);
43  } else {
44     valuesI = Z99 *sqrt(valuesVar)/sqrt(N);
45  }
46
47  printf("%10.10f %10.10f\n",valuesAvg,valuesI);
48
49  }' > "net_rx_V1_Fedora.dat"
50
51  #rm -f  $TMP_VALUES
```

}

- **Código para realizar las graficas utilizando los valores de Medias obtenidos mediante la Herramienta Gnuplot.**

```
1  #!/bin/bash
2  function plot {
3  cat $PLT_TEMPLATE | sed "s/OUTPUTFILE/$2/g" | sed "s/INPUTFILE/$1/g" |
       sed "s/TITLE/$3/g" | sed "s/UNITS/$4/g" | sed "s/MAXY/$5/g"  > tmp.
       metric.plt
4
5  gnuplot  tmp.metric.plt
6  ps2pdf $2.ps
7  rm -f $2.ps
8  rm -f tmp.metric.plt
9  }
10
11 PLT_TEMPLATE="metric_graph.plt"
12
13 plot "cpu.dat" "cpu" "CPU Utilization by Operating System and Zookeeper
       " "CPU Utilization (%)" ""
```

```
14
15 plot "mem.dat" "mem" "Memory␣Utilization␣by␣Operating␣System␣and␣
       Zookeeper" "Memory␣Utilization␣(%)" ""
16
17 plot "hd.dat" "disc_rd" "Disc␣Utilization␣by␣Operating␣System␣and␣
       Zookeeper" "HD␣Utilization␣(%)Kbits\/second" ""
18
19 plot "net_rx.dat" "net_rx" "Network␣Utilization␣rxKB␣by␣Operating␣
       System␣and␣Zookeeper" "Net␣Utilization␣(%)Kbits\/second" ""
20
21 plot "net_tx.dat" "net_tx" "Network␣Utilization␣txKB␣by␣Operating␣
       System␣and␣Zookeeper" "Net␣Utilization␣(%)Kbits\/second" ""
22
23 set terminal postscript color solid 18 set output "OUTPUTFILE.ps" set
       key left top
24 set boxwidth 0.9 absolute set style fill solid 1.00 border -1 set
       datafile missing '-' set style data histograms #set style histogram
        clustered gap 1 title  offset character 0, 0, 0 set style
       histogram errorbars gap 2 lw 2
25 set xtics border in scale 1,0.5 nomirror offset character 0, 0, 0 set
       xtics("␣" 0.00000)
26 set title "TITLE" set xlabel "␣" set ylabel "UNITS" set grid
27 set autoscale y set yrange [0:MAXY]
28 plot "INPUTFILE" using 2:3:xtic(1) ti col fs solid lc rgb "#FFFFFF", ''
        u 4:5 ti col fs solid lc rgb "#f33500" reset
```