

**FACULTAD DE INGENIERÍA**

**TESIS DE MAESTRÍA**

*Una propuesta de mejora para la arquitectura de Sistemas de Monitoreo de Servidores, basada en un Modelo Distribuido, utilizando la tecnología Cloud*

*Computing.*

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO

MASTER EN TECNOLOGÍA DE LA INFORMACIÓN

**ALUMNO:**

JUAN CARLOS JIMÉNEZ LOZANO

**TUTOR DE TESIS:**

NICOLÁS D’IPPOLITO

BUENOS AIRES – ARGENTINA

2014

**APROBACIÓN DEL COMITÉ**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tutor

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Presidente del jurado

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Jurado

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Jurado Externo

Defensa de la tesis

Buenos Aires, Capital Federal, A los \_\_\_ días del mes \_\_\_\_\_\_\_\_\_\_\_\_\_ de \_\_\_\_\_\_\_\_ **DECLARACIÓN Y AUTORIZACIÓN**

Yo, Juan Carlos Jiménez Lozano, declaro bajo juramento que el proyecto aquí descrito es de mi autoría y no ha sido previamente presentado para ningún grado o calificación profesional, así mismo el trabajo ha sido realizado basándome en las reseñas bibliográficas incluidas en el presente trabajo.

Por medio de esta declaración, cedo mis derechos de propiedad intelectual del presente trabajo a la Universidad de Palermo, según lo establecido por el reglamento vigente de la misma Universidad de Palermo.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Juan Carlos Jiménez Lozano

**CERTIFICACIÓN**

Certifico que el presente trabajo de tesis fue desarrollado por Juan Carlos Jiménez Lozano bajo mi supervisión

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Nicolás D’Ippolito

**TUTOR DE TESIS**

**AGRADECIMIENTOS**

A todas las personas, que durante mi estadía en Argentina supieron darme su apoyo y buenos deseos para culminar con mis estudios de maestría.

Mis sinceros agradecimientos.

Juan Carlos Jiménez Lozano

**DEDICATORIA**

Quiero dedicar este trabajo a mis padres, por ser las personas más importantes y que las aprecio con todo mi corazón. La culminación de este trabajo de investigación es la muestra de cariño que les tengo y sobretodo el haber cumplido con los retos propuestos durante importante etapa de mi vida.

*Juan Carlos*

**RESUMEN DE LA TESIS**

En este trabajo de tesis se presenta una arquitectura de referencia para Sistemas de

Monitoreo de Servidores, basadas en un modelo distribuido, utilizando la tecnología Cloud Computing. Dicha arquitectura representa una propuesta de mejora en dos sentidos; por un lado, es una mejora desde el modelo arquitectónico de los Sistemas de Monitoreo Tradicionales, que optimiza las técnicas que estos utilizan para almacenar y procesar la información que recopilan de los diferentes componentes de los centros de datos; y por otro, utiliza métodos de recolección de datos basados en aquellos existentes en los sistemas tradicionales, que permiten optimizar la recolección y el posterior procesamiento de los datos.

La arquitectura de referencia propuesta se obtiene a partir de un análisis arquitectónico tanto de los sistemas de monitoreo tradicionales y los distribuidos. Dicho análisis esta basado en los atributos de calidad de cada uno de los componentes de la solución quienes son utilizados para identificar y corregir los puntos críticos de falla de las diferentes arquitecturas.

[1.INTRODUCCIÓN 1](#_Toc39564184)

[1.1 DEFINICIÓN DEL PROBLEMA 1](#_Toc39564185)

[1.2 ANTECEDENTES 2](#_Toc39564186)

[1.3 JUSTIFICACIÓN 4](#_Toc39564187)

[1.4 ALCANCE Y LIMITACIONES 4](#_Toc39564188)

[1.5 OBJETIVOS 5](#_Toc39564189)

[1.6 MATERIALES Y MÉTODOS 5](#_Toc39564190)

[CAPÍTULO 2 6](#_Toc39564191)

[2.ESTUDIO PRELIMINAR DE LOS SISTEMAS DE MONITOREO 6](#_Toc39564192)

[2.1 INTRODUCCIÓN DEL CAPÍTULO 6](#_Toc39564193)

[2.2 DEFINICIÓN DE TECNOLOGÍAS BASE 7](#_Toc39564194)

[2.2.1. Nagios 7](#_Toc39564195)

[2.2.2. MapReduce y la plataforma Hadoop 13](#_Toc39564196)

[2.2.3. Chukwa 18](#_Toc39564197)

[2.3 ARQUITECTURA DE LOS SISTEMAS DE MONITOREO 20](#_Toc39564198)

[2.3.1. Arquitectura de sistemas de monitoreo tradicionales 21](#_Toc39564199)

[2.3.2. Arquitectura de sistemas de monitoreo distribuidos 24](#_Toc39564200)

[2.4 CONCLUSIONES DEL CAPÍTULO 30](#_Toc39564201)

[CAPÍTULO 3 32](#_Toc39564202)

[3.PROPUESTA DE MEJORA 32](#_Toc39564203)

[3.1 INTRODUCCIÓN DEL CAPÍTULO 32](#_Toc39564204)

[3.2 TRABAJOS RELACIONADOS 32](#_Toc39564205)

[3.3 PLANTEAMIENTO DE LA SOLUCIÓN 34](#_Toc39564206)

[3.3.1. Vista de componentes de la solución 34](#_Toc39564207)

[3.3.2. Arquitectura de la solución 36](#_Toc39564208)

[3.3.3. Guía general para la implementación arquitectónica 39](#_Toc39564209)

[3.4 EVALUACIÓN DE LA ARQUITECTURA 42](#_Toc39564210)

[3.5 CONCLUSIONES DEL CAPÍTULO 44](#_Toc39564211)

[CAPÍTULO 4 45](#_Toc39564212)

[4.IMPLEMENTACIÓN 45](#_Toc39564213)

[4.1 INTRODUCCIÓN DEL CAPÍTULO 45](#_Toc39564214)

[4.2 IMPLEMENTACIÓN DEL PROTOTIPO FUNCIONAL 46](#_Toc39564215)

[4.3 EJECUCIÓN Y RESULTADO DE LAS PRUEBAS 48](#_Toc39564216)

[4.4 CONCLUSIONES DEL CAPÍTULO 58](#_Toc39564217)

[CAPÍTULO 5 58](#_Toc39564218)

[5.DISCUSIÓN GENERAL DEL PROYECTO 58](#_Toc39564219)

[5.1 DISCUSIÓN 59](#_Toc39564220)

[5.2 CONCLUSIONES 62](#_Toc39564221)

[5.3 TRABAJO FUTURO 63](#_Toc39564222)

[5.4 COMENTARIOS FINALES 65](#_Toc39564223)

[REFERENCIAS 66](#_Toc39564224)

[ANEXOS 69](#_Toc39564225)

**Lista de figuras**

*Figura 2.1: Proceso de chequeo de servicios en NRPE ............................................................................................. 11*

*Figura 2.2: Flujo de procesos entre chequeos activos y pasivos .......................................................................... 13*

*Figura 2.3: Portal de monitoreo de Chukwa (HICC) ................................................................................................. 21*

*Figura 2.4: Arquitectura de Nagios ................................................................................................................................. 23*

*Figura 2.5: Arquitectura básica de Hadoop ................................................................................................................. 26*

*Figura 2.6: Arquitectura de Chukwa ............................................................................................................................... 29*

*Figura 3.1: Vista de componentes de la solución ....................................................................................................... 36*

*Figura 3.2: Arquitectura de la solución ......................................................................................................................... 38*

*Figura 3.3: Diagrama de despliegue del prototipo funcional .............................................................................. 42*

*Figura 4.1: Topología de Red para las pruebas .......................................................................................................... 48*

*Figura 4.2: Pantalla de inicio de Nagios ........................................................................................................................ 49*

*Figura 4.3: Visualización de hosts desde Nagios Master ........................................................................................ 52*

*Figura 4.4: Visualización del estado de los servicios en Nagios .......................................................................... 52*

*Figura 4.5: Visualización del mapa de red Nagios .................................................................................................... 53*

*Figura 4.6: Información del Namenode y el cluster de Hadoop .......................................................................... 53*

*Figura 4.7: Información del Datanode configurado en el cluster ...................................................................... 54*

*Figura 4.8: Estructura del directorio raíz del sistema HDFS ............................................................................... 54*

*Figura 4.9: Directorio de almacenamiento de archivos log en el sistema HDFS ......................................... 55*

*Figura 4.10: Resultado de la ejecución de una tarea MapReduce ...................................................................... 57*

*Figura 5.1: Componentes del colector de alertas ....................................................................................................... 61*

**Lista de tablas**

*Tabla 2.1: Códigos de salida de los plugins de Nagios ............................................................................................. 24*

*Tabla 3.1: Descripción de campos del archivo Colector ......................................................................................... 39*

*Tabla 3.2: Esquema de pruebas del prototipo funcional ........................................................................................ 43*

*Tabla 3.3: Árbol de utilidad de los atributos de calidad ......................................................................................... 46*

*Tabla 4.1: Esquema de pruebas del prototipo funcional ........................................................................................ 51*

**CAPÍTULO 1**

# 1.INTRODUCCIÓN

## 1.1 DEFINICIÓN DEL PROBLEMA

Los sistemas de monitoreo son herramientas que se utilizan para la gestión de incidentes dentro de una infraestructura de redes y servidores. Es decir, que ayudan a los administradores de sistemas a inspeccionar frecuentemente el comportamiento de los diferentes componentes de la infraestructura de servidores.

Cuanto más grande sea la infraestructura a ser monitoreada, estos sistemas comienzan a presentar limitaciones a nivel de la gestión de la información que se almacena y se procesa dentro de sus componentes, es por esto que con el pasar del tiempo, estas limitaciones impactan directamente en el desempeño y capacidad de almacenamiento de la información.

Esta información tiene que ver con el estado de los componentes del centro de datos y dependiendo del sistema de monitoreo que se utilice, estos generan registros en diferentes formatos, ya sean: archivos de registros log, bases de datos o gráficas en tiempo real; para luego ser procesados e interpretados por un sistema centralizado que se encarga de generar alertas y estadísticas sobre los componentes monitoreados.

Para el presente proyecto, se toma como referencia de sistema de monitoreo tradicional, Nagios[16], sobre el cual se realiza un estudio de sus características tecnológicas y componentes arquitectónicos, enfocándose principalmente en el almacenamiento y análisis de los datos recolectados. A partir del estudio realizado se provee la propuesta de mejora para las arquitecturas de los sistemas de monitoreo tradicionales.

Para definir la solución, se toma como modelo de referencia la arquitectura de los sistemas de monitoreo distribuidos, específicamente se utiliza uno de los mas reconocidos y utilizados en la industria: Chukwa[12]. Dicho sistema gestiona las tareas de recolección y análisis de datos para el monitoreo de infraestructuras a gran escala; esta tecnología se encuentra implementada sobre la plataforma Hadoop[9], que sirve como base para definir el nuevo modelo arquitectónico para la mejora de los sistemas de monitoreo.

El propósito del estudio de las técnicas de recolección de datos que utiliza esta tecnología, es la de establecer una plataforma de almacenamiento distribuido en donde se recolecta los datos de monitoreo provenientes de cada uno de los componentes de la infraestructura de servidores, teniendo así una base de conocimientos que sirven como datos de entrada para la implementación de técnicas de análisis de la información, al aplicar dichas técnicas se puede detectar o establecer patrones de comportamiento de los componentes de la infraestructura y su evolución durante el paso del tiempo.

Durante el desarrollo del proyecto, se explicará en mayor detalle los atributos de calidad que poseen cada uno de estas arquitecturas, para luego ser evaluadas y definir los componentes y atributos que se usan como referencia para el desarrollo de la solución propuesta.

## 1.2 ANTECEDENTES

En la actualidad, existe una variedad de sistemas de monitoreo para infraestructuras de servidores, entre los más utilizados tenemos: Nagios, el cual es un sistema de monitoreo de infraestructura open-source que esta basado en la utilización de plugins para realizar el monitoreo de los componentes de la infraestructura de servidores.

Los plugins que utiliza Nagios, son pequeños programas desarrollados por terceras personas y son los encargados de realizar el análisis de los servicios, una vez que se obtenga el valor, se le asigna un estado y este es solicitado por el servidor de Nagios por medio de una consulta para que de acuerdo al estado se genere la alerta.

Este sistema es lo suficientemente versátil para consultar cualquier parámetro de interés de un sistema, y generar alertas que son recibidas por los responsables correspondientes mediante correo electrónico, mensajes SMS de acuerdo a los criterios establecidos por los administradores de sistemas del centro de datos.

Por otro lado, tenemos el modelo de programación MapReduce, mencionado en la sección anterior, el cual se encuentra implementado principalmente en la plataforma Hadoop. Dentro de este proyecto desarrollado por la Fundación Apache, se encuentran algunos sub-proyectos tales como Hadoop Common, Chukwa, HBase, HDFS, Hive, Pig, entre otros.

Hadoop cuentan con un sistema de archivos distribuido llamado HDFS, el cual se encarga de almacenar de manera distribuida, es decir, que la información se encuentra en varios componentes o nodos que en conjunto funcionan como un solo sistema almacenamiento virtual. En este sistema se almacenan y se gestiona la metadata o información descriptiva de los archivos y los datos o ficheros que ocupan un espacio físico en el sistema de almacenamiento.

El nodo encargado de almacenar la metadata, es un servidor dedicado llamado NameNode, mientras que los ficheros se distribuyen en diferentes componentes llamados DataNodes, los mismos que en conjunto forman el cluster de la plataforma Hadoop y se encuentran interconectados a través del protocolo de internet TCP/IP.

Sobre esta plataforma, existe un sub-proyecto de Hadoop, llamado Chukwa, es cual es un sistema de colección de datos para monitorear y analizar grandes sistemas

distribuidos.

Para este proyecto se realizará el análisis de los componentes arquitectónicos sistema de monitoreo Chukwa y posteriormente se tomará como referencia para definir la propuesta de mejora para la arquitectura de los sistemas de monitoreo tradicionales.

## 1.3 JUSTIFICACIÓN

La necesidad que surge al momento de plantear el presente proyecto de tesis, es la de proponer una mezcla de arquitecturas de dos tecnologías que tienen funcionalidades similares, pero con diferentes componentes y especificaciones, al realizar esta mezcla se presentará una arquitectura, para optimizar la forma tradicional de realizar las tareas de monitorización, orientando así los resultados al análisis del comportamiento de los componentes existentes dentro de un centro de datos, independientemente en cantidad o en ubicación geográfica.

Como arquitectura tradicional, se va tomar la del sistema open-source Nagios, el cual se realizará un análisis de sus componentes y los posibles puntos de mejora.

La arquitectura que se utilizará para realizar la mezcla, será la del sub-proyecto de Apache Hadoop, el sistema Chukwa; el cual se definirá un modelo en el que estas dos herramientas puedan aprovechar las ventajas de cada una para definir la nueva arquitectura mejorada.

## 1.4 ALCANCE Y LIMITACIONES

A continuación se presentan los alcances del proyecto:

* Realizar un estudio preliminar de las tecnologías base de los sistemas de monitoreo tradicionales y distribuidos.
* Análisis de los componentes arquitectónicos de Nagios, la plataforma Hadoop y como está implementado el sub proyecto Chukwa.
* Presentación de la propuesta de mejora y definición de componentes para la implementación de un prototipo funcional.
* Evaluación de la arquitectura propuesta y análisis de los atributos de calidad.

## 1.5 OBJETIVOS

* Determinar qué beneficios se obtienen al adoptar un sistema de monitoreo distribuido para el control de una infraestructura de servidores.
* Exponer qué componentes de un sistema de monitoreo tradicional se pueden mejorar incorporando las herramientas y técnicas que brinda un ambiente de monitoreo distribuido.
* Establecer los criterios de factibilidad para la implementación de la propuesta de mejora para la arquitectura de los sistemas de monitoreo.

## 1.6 MATERIALES Y MÉTODOS

Durante el tiempo de elaboración de la tesis, los materiales utilizados en el proyecto se basan en la revisión de documentación teórica y técnica sobre el estado del arte en el que se encuentran los sistemas de monitoreo y a la aplicación de estas tecnologías en un prototipo elaborado sobre una red local cuyas características se detallan más adelante.

La metodología utilizada para el desarrollo de este proyecto, esta enfocada en dos fases; en primer lugar, se utiliza el *método científico[[1]](#footnote-1),* de gran utilidad para la investigación y análisis de los sistemas de monitoreo tradicionales y distribuidos definidos en un estudio preliminar de sus especificaciones tecnológicas y componentes arquitectónicos; los cuales son la base para el desarrollo de una novedosa arquitectura que mezcla ambas tecnologías, lo que permite solucionar las limitaciones de los sistemas tradicionales y la complejidad tecnológica de los sistemas distribuidos.

Con los datos obtenidos en este estudio, en la siguiente fase se utiliza el *método experimental[[2]](#footnote-2)* para la planificación y diseño de un prototipo funcional, que basado en la arquitectura propuesta, permite la implementación de este nuevo sistema, el mismo que abre las puertas para una futura implementación a gran escala.

El prototipo consiste en el diseño e implementación de un sistema de monitoreo básico que se encuentra constantemente revisando el funcionamiento de un equipo cliente, al cual se inducen pruebas de estrés para generar alertas, con un formato específico, de acuerdo a los servicios que se proveen, las mismas que serán guardadas en un sistema de almacenamiento distribuido. Posteriormente, se realiza el análisis correspondiente para la clasificación de los incidentes, cuyos resultados representan las métricas sobre el comportamiento de los clientes dentro de una infraestructura.

# CAPÍTULO 2

# 2.ESTUDIO PRELIMINAR DE LOS SISTEMAS DE MONITOREO

## 2.1 INTRODUCCIÓN DEL CAPÍTULO

En este capítulo se presenta un análisis preliminar de los sistemas de monitoreo, sus componentes tecnológicos y arquitecturas, los mismos que se utilizan como referencia para definir la propuesta de la solución. El análisis de estos sistemas se ha dividido en dos secciones: en la primera, se tiene la definición de las tecnologías base, en donde Nagios[16] ha sido elegido para representar a los sistemas de monitoreo tradicionales.

Para los sistemas de monitoreo distribuidos, se toma como representante a Chukwa[12], este sistema se encuentra implementado sobre la plataforma Hadoop, dentro del cual se destaca el modelo de programación MapReduce, que es uno de los componentes más importantes de la plataforma para el procesamiento de grandes cantidades de información.

En la segunda parte de este capítulo, se muestra el análisis conceptual de las arquitecturas de los sistemas de monitoreo tradicionales y distribuidos, enfocando los conceptos y características de cada uno de sus componentes hacia las tecnologías base que se definieron en la sección inicial del capítulo.

## 2.2 DEFINICIÓN DE TECNOLOGÍAS BASE

### 2.2.1. Nagios

Nagios es una herramienta para el monitoreo de infraestructuras de redes y servidores. Esto significa que es un sistema automatizado, que se encuentra revisando de forma permanente el estado de los componentes de una infraestructura[3]. Esta herramienta tiene como propósito detectar y reportar cualquier anomalía que se presente en un host o servicio, lo más rápido posible, con el fin de que los administradores de TI puedan enterarse del problema mucho antes que los usuarios o clientes de los sistemas.

• **Monitoreo de objetos**

Nagios no realiza ninguna verificación hacia un objeto(host o servicio) por su propia cuenta. En su lugar, Nagios utiliza plugins[17], que son extensiones independientes a Nagios y proveen un nivel de inteligencia a bajo nivel, para realizar el monitoreo de cualquier objeto utilizando el núcleo de Nagios.

Los plugins ejecutan conjuntamente comandos y argumentos, dependiendo del servicio que estén monitoreando y los resultados se envían al núcleo de Nagios para ser procesados. Estos plugins pueden ser de dos tipos: *archivos binarios ejecutables* (escritos en C, C++, etc.) o también *scripts ejecutables* (shell, Perl, PHP, etc.).

Los objetos que son monitoreados por Nagios se dividen en dos categorías: hosts y servicios. Los hosts son equipos físicos (servidores, routers, estaciones de trabajo, impresoras de red), mientras que los servicios son funcionalidades o procesos particulares, como por ejemplo: el proceso que ejecuta un servidor web(httpd), donde se encuentra alojado una página web o aplicación, puede ser definido como un servicio a ser monitoreado y esta asociado a un host que es donde se ejecuta el proceso.

Los servicios se clasifican en dos grupos:

* **Públicos**, a los que se accede a través de una red pública(Internet), privada o local. Por ejemplo: HTTP (Web), POP3/IMAP/SMTP (Correo Electrónico),

FTP (Transferencia de archivos), SSH (Acceso Remoto).

* **Privados**, estos servicios requieren un agente intermediario para que Nagios realice el *chequeo*[[3]](#footnote-3) del servicio. Por ejemplo: carga del CPU, uso de memoria, uso de disco duro, sesiones de usuario, información de procesos, etc.

Nagios posee dos características fundamentales al momento de realizar el monitoreo de los objetos, la primera es, que en lugar de revisar valores puntuales, se utilizan cuatro estados o niveles de criticidad, para describir el estado de un servicio, estos estados son: *ok*, *warning*, *critical* y *unknown*. Estos valores abstractos, permiten a los administradores de la infraestructura evitar revisar los valores de monitoreo y centrarse solamente en los niveles de criticidad en el que se encuentra el servicio. Al tener un nivel estricto de criticidad, se puede capturar o identificar las potenciales fallas de manera anticipada, con el fin de definir un plan de mitigación para el incidente.

La segunda característica es el manejo de reportes del estado de los servicios, que se encuentran tanto en su funcionamiento normal(*ok*) como en estado de *warning* o *critical*, permitiendo tener un reporte con información detallada del estado actual de los componentes de la infraestructura.

Además del monitoreo, Nagios posee un sistema de notificaciones[3], para informar a los diferentes encargados del centro de datos sobre los servicios o host que presenten anomalías en sus componentes. Se pueden configurar diferentes tipos de

notificaciones: correo electrónico, mensajes de texto, mensajería instantánea o través de la web. Así mismo, es posible dividir a los usuarios por grupos de contacto, para que las notificaciones sean recibidas únicamente por los interesados en la administración de servicios concretos.

• **Recolección y análisis de datos**

Como se menciona en la sección anterior, los plugins son extensiones independientes a Nagios, que pueden ser binarios ejecutables o scripts, estos se ejecutan desde la línea de comandos del sistema operativo para chequear el estado de un host o servicio y con la opción de recibir argumentos que definen los niveles de criticidad de las

alertas.

Nagios ejecuta un plugin siempre que exista la necesidad de revisar el estado de un host o servicio. Luego el plugin, ejecuta el chequeo y devuelve el resultado al servidor principal.

Finalmente, Nagios procesa el resultado que reciba del plugin para tomar las acciones necesarias ante el resultado obtenido(generar alertas, ejecutar un comando, enviar notificaciones).

Los plugins representan una capa de abstracción[18], entre la lógica de monitoreo de Nagios y los objetos que se encuentran monitoreados. En el segunda parte de este capítulo se explicará con mayor detalle las diferentes capas que conforman la arquitectura de Nagios.

• **Complementos de Nagios**

Para que Nagios tenga la capacidad de gestionar la ejecución de los plugins que se encuentran configurados en los diferentes hosts o servicios de una infraestructura, es necesario la instalación de complementos[17] que agreguen funcionalidades al núcleo principal de Nagios.

A continuación se detallan dos de los complementos mayormente utilizados por los usuarios de Nagios:

• **NRPE**

Nagios Remote Plugin Executor(NRPE)[18] es un complemento que permite la ejecución remota y segura de los plugins en host Linux/Unix, para realizar el chequeo de servicios privados (CPU, memoria, disco duro, etc.) y públicos (HTTP, Mail, FTP, etc.) de manera local, es decir en el mismo host donde se ejecuta el servicio.

NRPE utiliza un protocolo TCP[5] personalizado para la comunicación entre el servidor principal de Nagios y el host a ser monitoreado, su principal característica es la baja sobrecarga en el canal de comunicaciones, en comparación con otros métodos, tales como los chequeos a través del protocolo SSH[3].

Para brindar una mayor seguridad de la información, NRPE puede utilizar SSL(Secure Socket Layer) para cifrar la información al momento de transmitir los datos de los equipos remotos a través de la red. Por defecto, NRPE se comunica a través del puerto 5666.

En la siguiente figura, se muestra el esquema de funcionamiento del chequeo de servicios a través de NRPE[3]:

Nagios server

Remote machine

NRPE daemon

Nagios plugin

Accept connection

Run plugin check

Perform check

Pass output and exit

code

Nagios process

check\_nrpe

Schedule an active

check

Connect to remote

machine

Receive results and

pass them to Nagios

Store check results

*Figura 2.1: Proceso de chequeo de servicios en NRPE*

Como primer paso, el proceso principal de Nagios decide el host y el chequeo que se va ejecutar, luego de esto, en el mismo servidor principal se ejecuta el comando *check\_nrpe,* que es un plugin de Nagios encargado de conectarse con el proceso NRPE del host remoto, al aceptar la conexión el host remoto, se envía el comando que se debe ejecutar con los parámetros correspondientes al host.

Una vez ejecutado el comando, el proceso NRPE del host, envía los resultados al comando *check\_nrpe* del servidor de Nagios, para que finalmente el proceso principal de Nagios realice la operación de procesamiento y almacenamiento del resultado ya sea en los archivos de registro log o en una base de datos.

• **Chequeos pasivos y NSCA**

Además del chequeo de hosts y servicios a través del complemento NRPE, Nagios ofrece una alternativa de monitoreo, donde se puede configurar el servidor principal para que reciba información del estado de los servicios desde una aplicación que se va encargar de generar y enviar la información a Nagios a través de un “command pipe”[3].

El objetivo del chequeo pasivo es que las mismas aplicaciones sean las encargadas de enviar el reporte de estado hacia Nagios, esto se da cuando se realizan tareas de verificación que tomen un tiempo prolongado o que se deban enviar múltiples resultados para ser procesados.

Active check flow

Passive check flow

Nagios process

External application

Schedule test to

perform / data to send

Gather information /

perform tests

Send results over

external command pipe

Parse passive check

results

Store check results

Nagios process

Check command

Scheduled / on

demand checks

Parse command output

Store check results

Perform check

Run external command

*Figura 2.2: Flujo de procesos entre chequeos activos y pasivos*

La Figura 2.2, muestra la diferencia del flujo de procesos que Nagios ejecuta al realizar chequeos activos y pasivos[3].

En los **chequeos activos**, Nagios es el encargado de gestionar y programar la ejecución de los chequeos bajo demanda, a través del comando externo, en el caso de NRPE, el comando *check\_nrpe*; para luego analizar y almacenar los resultados.

Para el flujo de los **chequeos pasivos**, la aplicación externa es la encargada de programar la verificación del estado y preparar los resultados para ser enviados al servidor de Nagios y una vez que reciba los resultados a través del “command pipe”, se proceda a analizara y almacenar los datos.

Al igual que NRPE, existe un complemento de Nagios llamado NSCA (Nagios Service Check Acceptor), el mismo que se encarga de enviar a Nagios los resultados de los chequeos pasivos a través de la red. Este complemento consta de dos partes: *cliente* y *servidor*. El responsable de recibir los resultados de los chequeos y enviarlos a Nagios es el *servidor*. Este proceso se encuentra escuchando a través de un puerto TCP las peticiones de los clientes NSCA, y luego de autenticar la información proveniente de los clientes, envía los resultados hacia el proceso de Nagios.

El *cliente* NSCA, acepta los resultados de los chequeos de servicios de uno o más hosts, a través de la entrada estándar y se envían al servidor NSCA usando la dirección IP y el puerto del mismo.

Una aplicación importante para el complemento NSCA es la configuración de ambientes distribuidos y configuraciones redundantes/con tolerancia a fallos[18]. Esto quiere decir, transmitir información a través de varios servidores Nagios que se encuentren monitoreando del funcionamiento de un conjunto de hosts y servicios en redes independientes.

En el siguiente capítulo, se explica con mayor detalle la configuración de un ambiente distribuido de Nagios, el cual forma parte de la solución propuesta de este proyecto de

tesis.

### 2.2.2. MapReduce y la plataforma Hadoop

• **MapReduce**

MapReduce[6], es un modelo de programación que se utiliza para dar soporte al procesamiento de grandes cantidades de información. Su funcionamiento se basa en distribuir de manera automática las tareas de procesamiento a los diferentes componentes o nodos que conforman un *cluster de computadoras[[4]](#footnote-4)* de bajo costo, en donde estas tareas se ejecutan en paralelo y su resultado se guarda en un medio de almacenamiento.

En este modelo se definen dos funciones primordiales: la función ***map()***, que se encarga de procesar una estructura de datos de tipo llave/valor, la cual devuelve un conjunto ordenado de *datos intermedios* también de tipo llave/valor; estos datos son la entrada de la función ***reduce()***, que se encarga de mezclar y agrupar todos estos valores intermedios, tomando como criterio la llave generada por la función *map()*, devolviendo así un conjunto de valores combinados apropiadamente.

A nivel de arquitectura[7], la función map() se ejecuta de forma distribuida en el cluster de computadoras. Los datos de entrada están conformados por un archivo de gran tamaño y se divide en un conjunto de M particiones de entrada.

Al dar inicio a una ejecución MapReduce, se realizan las siguientes operaciones:

1. Se dividen los datos entrada en M particiones, de 16 a 64 megabytes. Luego se inician copias o instancias[[5]](#footnote-5) del programa MapReduce en cada una de las máquinas que conforman el cluster. Por regla general, el número de copias se configura en los parámetros de la aplicación que implementa el algoritmo[6].
2. Una de las instancias del programa se denomina ***nodo master***, el resto de instancias son los ***nodos workers***, los mismos que ejecutan las tareas map/reduce que son asignadas por el nodo master tomando en cuenta el estado en que se encuentren los workers. Un worker se puede encontrar en tres estados: reposo, trabajando y completo.
3. Al ejecutarse un tarea map(), el worker toma como entrada una partición de datos M, y se encargará de crear el conjunto de datos intermedios tipo llave/valor, los mismos que serán almacenados en un buffer de memoria[[6]](#footnote-6).
4. Periódicamente, los datos tipo llave/valor almacenados en el buffer se escriben en el disco local del worker repartidos en R regiones. Las regiones de los datos llave/valor son pasados al nodo master, que es responsable de redirigir hacia los workers que ejecutan la función reduce().
5. Los workers que ejecutan la función reduce(), reciben del nodo master la localización de la partición procesada por la función map(), y utiliza llamadas a procedimientos remotos (RPC)[[7]](#footnote-7) para acceder a los datos y agruparlos por los elementos que posean la misma llave. El resultado de la función reduce() se agrega al archivo de salida del proceso MapReduce.
6. Cuando todas las tareas map() y reduce() se han completado, el nodo master devuelve el control al código del usuario y las salidas se distribuyen en uno o varios archivos R. Estos pueden ser la entrada de otro programa MapReduce o puede ser procesado por cualquier otra aplicación que requiera estos datos.

* **Hadoop**

Hadoop es un proyecto desarrollado y soportado por la Fundación de Software Apache[19], con otros sub proyectos para el desarrollo de software libre y aplicaciones de computación distribuida altamente escalables.

Hadoop implementa el modelo de programación *MapReduce* y provee un sistema de archivos distribuido llamado HDFS (Hadoop Distributed File System)[7], utilizando un cluster de computadoras de similares características, para distribuir a través de cada uno de sus componentes el almacenamiento de datos y la ejecución de tareas en paralelo.

Las componentes fundamentales de Hadoop son: el framework de desarrollo para el modelo MapReduce y el sistema de archivos distribuido HDFS.

* **El Núcleo de Hadoop MapReduce**

Hadoop provee un *framework[[8]](#footnote-8)* para la gestión de ejecución de las tareas MapReduce que serán procesadas en el cluster de computadoras que implementa la plataforma Hadoop.

Es necesario que el usuario especifique en el framework los siguientes requerimientos:

* La localización de los datos de entrada de la tarea, en el sistema de archivos distribuido.
* La localización de los datos de salida de la tarea, en el sistema de archivos distribuido.
* El formato de entrada.
* El formato de salida.
* La clase o programa que contiene la función map().
* Opcionalmente, la clase que contiene la función reduce().
* Los archivos JAR o librerías que contienen las funciones map/reduce y

cualquier otra clase para el soporte de la tarea.

El framework provee dos procesos que ejecutan las tareas MapReduce:

* **JobTracker**, acepta el envío de trabajos o tareas map/reduce, provee el monitoreo y control de los mismos y gestiona la distribución de tareas hacia los nodos TaskTracker.
* **TaskTracker**, este proceso gestiona la ejecución de tareas individuales map/reduce, que se estén llevando a cabo en cada uno de los nodos del cluster. Por lo general existe un proceso JobTracker por cluster y uno o más procesos

TaskTracker por nodo en cluster.

* **El sistema de archivos distribuido de Hadoop**

HDFS es un sistema de archivos distribuido que está diseñado como medio de almacenamiento para la ejecución de tareas MapReduce, las mismas que utilizan un método de obtención de datos basados en grandes pedazos(chunks) de información, estos se procesan y escriben en nuevos datos de salida y se agrupan para formar nuevos pedazos de información procesada.

Para garantizar la confiabilidad de la información, los datos son replicados[7] hacia múltiples nodos de almacenamiento. Mientras exista una réplica de los datos disponible, el consumidor de la información no sabrá que existe una falla en el servicio de almacenamiento.

Al igual que el framework MapReduce, el sistema HDFS está compuesto por dos procesos:

* **NameNode**, es el encargado de la gestión de la *metadata*[[9]](#footnote-9) del sistema de archivos distribuido, posee un espacio de nombres que define la jerarquía de archivos y directorios.
* **DataNode**, este proceso provee almacenamiento por bloques y servicios de recuperación, al iniciarse se comunica directamente al NameNode para sincronizar la información de la metadata y almacenar los bloques de datos que se procesan en el cluster.

Existe un solo proceso NameNode que gestiona el sistema de archivos HDFS, por lo que esto implica un punto crítico de falla para la plataforma. El núcleo de Hadoop provee recuperación y respaldos automáticos del NameNode, pero no provee soporte para recuperación del servicio en tiempo real.

Por otro lado existen múltiples procesos DataNode dentro del cluster, típicamente un proceso por nodo de almacenamiento en el cluster.

### 2.2.3. Chukwa

Chukwa es un sistema open source de recolección y almacenamiento de datos para el monitoreo y análisis de grandes sistemas distribuidos[10]. Es uno de los sub proyectos de la plataforma Hadoop[9], por lo que sus componentes se basan en el uso del modelo programación Mapreduce y se puede usar el sistema de archivos distribuido HDFS para el almacenamiento de la información.

Además de los componentes de la plataforma Hadoop, Chukwa posee un conjunto de herramientas para representar los resultados de análisis y monitoreo de la información que se recolecta.

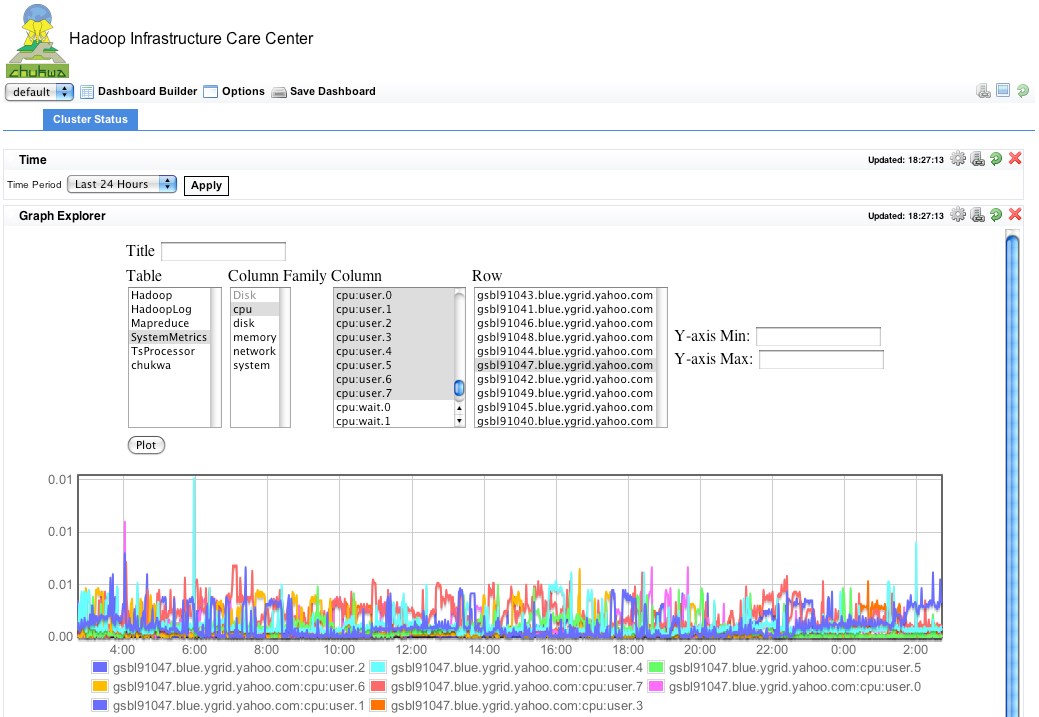
Chukwa se encuentra formado por cuatro componentes principales: Agentes, Colectores, Tareas de MapReduce y el componente HICC.

Los ***Agentes*** se encargan de recolectar los datos a través de *adaptadores*; los mismos que son los que interactúan directamente con el origen de los datos. Dentro de un agente pueden existir múltiples adaptadores para recolectar diferentes tipos de información al mismo tiempo. Los agentes se ejecutan en cada uno de los nodos del cluster de Hadoop, por lo que se tiene un origen datos diferente por cada nodo.

Los ***Colectores*** reúnen los datos de los agentes a través del protocolo HTTP, y se encargan de almacenar la información en la plataforma Hadoop como un archivo de secuencia llamado “sink file”; el mismo que contiene los registros recolectados de los agentes. Este archivo al alcanzar un cierto tamaño, el colector se encarga de cerrar y renombrar el archivo y prepararlo para que sea procesado por las *tareas MapReduce*.

Una ***tarea MapReduce***, está conformada por dos trabajos: Archiving(Archivar) y Demux(Desmultiplexar), estos se encargan de organizar los datos recolectados de una manera más conveniente para el análisis y generan registros que son procesados por Chukwa.

***HICC*** (Hadoop Infrastructure Care Center), es un portal web que está incorporado a la plataforma Chukwa, para la visualización y monitoreo de la información que ha sido recolectada y analizada.



*Figura 2.3: Portal de monitoreo de Chukwa (HICC)*

Una vez que se han definido las tecnologías base a utilizar como referencia para generar la propuesta de mejora, en la siguiente sección se realiza el análisis de la arquitectura de los sistemas de monitoreo basándose en la especificación de componentes y atributos de calidad de los mismos.

## 2.3 ARQUITECTURA DE LOS SISTEMAS DE MONITOREO

La arquitectura de un sistema es el resultado de un conjunto de decisiones de diseño, y representa la estructura o estructuras de un sistema que comprende elementos de software, propiedades externas y relaciones entre si. El estándar de la IEEE 14712000 [20] define a la arquitectura de software como: la organización fundamental de un sistema, basada en sus componentes y relaciones, así como el ambiente y principios para su diseño y evolución.

En esta sección se muestra las arquitecturas de los sistemas de monitoreo tradicionales y distribuidos, en el que primeramente se analizan cada uno de los componentes arquitectónicos de Nagios, el cual se toma como referencia para representar a los sistemas de monitoreo tradicionales. En segunda instancia, se presenta la arquitectura de Hadoop y cómo se complementa con Chukwa para ser el representante de los sistemas de monitoreo distribuidos.

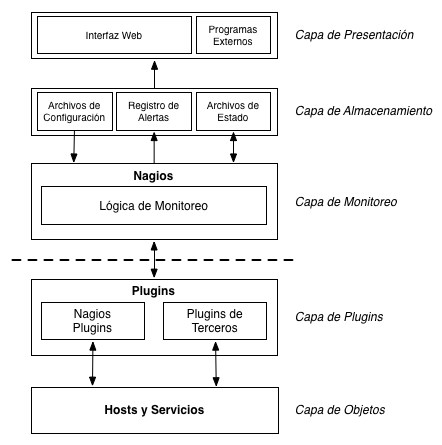
Las arquitecturas están representadas por capas[21], con la finalidad de identificar con mayor facilidad los diferentes componentes y describir el rol que desempeñan cada uno de estos dentro de los sistemas de monitoreo.

### 2.3.1. Arquitectura de sistemas de monitoreo tradicionales

Como se menciona en la sección 2.2.1, Nagios es una herramienta para el monitoreo de infraestructuras de redes y servidores, que no realiza verificaciones o chequeos directos hacia los objetos monitoreados. Para esto utiliza plugins, que son extensiones independientes que se encargan de ejecutar el chequeo del host o servicio, que pueden ser archivos binarios o scripts ejecutables.

Este esquema de trabajo, define a nivel de arquitectura una capa de abstracción[18] entre el sistema de monitoreo y los diferentes elementos de la infraestructura monitoreada, estableciendo así una arquitectura flexible que se pueda configurar de acuerdo a las necesidades o a cualquier tipo de infraestructura en la que se requiera implementar, además esta arquitectura brinda un ambiente interoperable entre diferentes plataformas y tecnologías que existan dentro del centro de datos.

A continuación se presenta la arquitectura de Nagios basada en el uso de plugins para monitorear los hosts y servicios de red:



*Figura 2.4: Arquitectura de Nagios*

1. **Capa de Objetos:** Esta capa representa el nivel donde se encuentran los host y servicios de la infraestructura de red. Estos objetos pueden ser:
   1. **A nivel de hosts**: servidores, estaciones de trabajo, hardware de red

(switches, routers, impresoras).

* 1. **A nivel de servicios**: Web, E-mail, FTP, puertos TCP/UDP, consumo de memoria, disco duro, carga de CPU y otros elementos internos que se encuentren dentro de un host.

1. **Capa de Plugins:** Esta capa representa una de las fortalezas que posee Nagios, ya que brinda la posibilidad de utilizar plugins externos para monitorear el funcionamiento de los servicios, independientemente del tipo de objeto que se esté monitoreando.

Los plugins ejecutan comandos externos, que interactúan directamente con el objeto monitoreado y devuelve un código de resultado específico, que pueden ser:

|  |  |  |
| --- | --- | --- |
| **Código de**  **Salida** | **Estado** | **Descripción** |
| 0 | OK | Trabajando correctamente. |
| 1 | WARNING | Trabajando, pero necesita atención. |
| 2 | CRITICAL | No se encuentra trabajando correctamente, requiere atención. |
| 3 | UNKNOWN | El plugin no puede determinar el status para el host o servicio. |

*Tabla 2.1: Códigos de salida de los plugins de Nagios*

La salida estándar de los comandos posee el siguiente formato:

ESTADO DEL PLUGIN – descripción del estado

Normalmente, la descripción contiene información comprensible sobre el estado del servicio, esto sirve para que el mensaje se pueda visualizar a través de la interfaz web de Nagios. Entre algunos ejemplos de servicios tenemos:

PING OK - Packet loss = 0%, RTA = 0.18 ms

DNS WARNING: 0.015 seconds response time

DISK CRITICAL - free space: /boot 18 MB (8% inode=99%) Los plugins de Nagios pueden utilizar opciones o argumentos, que sirven para definir la configuración de los mismos, para establecer los umbrales(límites) que definen el tipo de alerta, la dirección del host, el puerto del servicio a monitorear, el tiempo de respuesta y la ayuda o descripción de la utilización del plugin.

1. **Capa de Monitoreo:** Dentro de esta capa se encuentra el núcleo principal de Nagios, el que se encarga de enviar las solicitudes de chequeo hacia la *capa de plugins*, una vez que se ejecutan, se reciben las respuestas y se emiten las alertas correspondientes de acuerdo al estado de los objetos.

El núcleo de Nagios utiliza comandos internos para enviar las solicitudes de chequeo y las despacha a través de la red mediante un protocolo TCP propio.

1. **Capa de Almacenamiento:** En esta capa, se encuentran los archivos de configuración de Nagios, los archivos de estado y los registros de alertas. Dependiendo de la configuración del sistema, los registros de alertas o logs pueden ser almacenados en archivos de texto plano o se almacenen en una base de datos.
2. **Capa de Presentación:** En la capa de presentación, se encuentra el servicio web, en donde se aloja la interfaz gráfica de Nagios la cual funciona como el centro de monitoreo donde los administradores de la infraestructura pueden visualizar en tiempo real el estado y las alertas de los objetos, los grupos de hosts, contactos y un esquema gráfico de la infraestructura.

Otra parte importante de esta capa, son los Programas Externos. Nagios al ser un software open source, provee un API para el desarrollo de aplicaciones, en donde es posible interactuar, con los datos de capa de almacenamiento y realizar consultas personalizadas para obtener los datos.

### 2.3.2. Arquitectura de sistemas de monitoreo distribuidos

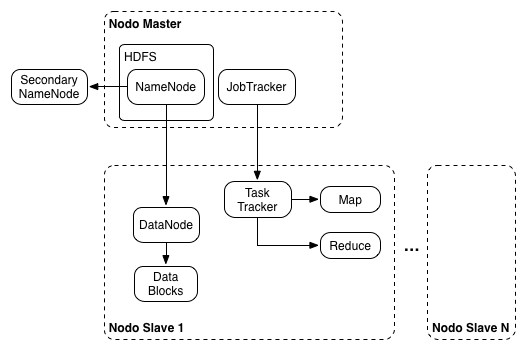
El modelo arquitectónico de un sistema distribuido trata sobre la colocación de sus partes y relaciones entre ellas[5]. Como se menciona anteriormente, se ha tomado Chukwa como representante de los sistemas de monitoreo distribuidos y para detallar los componentes de su arquitectura, esta sección se divide en dos partes, la primera consiste en la definición de los componentes de la arquitectura Hadoop, la misma que es la base de este sistema de monitoreo distribuido. En la segunda parte, se detallan los componentes de la arquitectura de Chukwa y cómo se comunica con la plataforma

Hadoop para realizar las tareas de monitoreo sobre una infraestructura de red.

• **Arquitectura de Hadoop**

La arquitectura de Hadoop consiste en dos modelos fundamentales: el sistema de archivos distribuido HDFS que se encarga de la gestión del almacenamiento, disponibilidad y redundancia de la información. Y la arquitectura MapReduce, la cual permite la ejecución y control de las tareas de procesamiento y análisis de la información almacenada en la plataforma. Estos modelos están conformados por distintos componentes y se comunican por medio de mensajes a través de la red que conforman el cluster de Hadoop.

En la siguiente figura se muestra la arquitectura que define al cluster de Hadoop, en donde se especifican sus componentes y la interacción entre cada uno de los nodos que forman el cluster:



*Figura 2.5: Arquitectura básica de Hadoop*

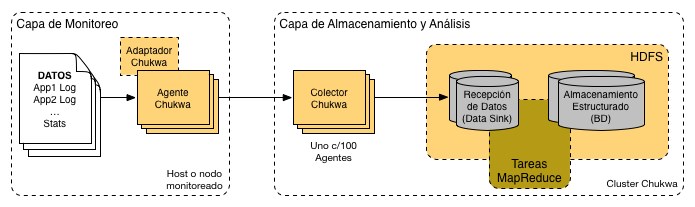
1. **Nodo Master:** Este es el nodo principal de la plataforma Hadoop, existe un nodo master por cada cluster, los componentes que conforman a este nodo son los siguientes:
   1. **NameNode**,mantiene el árbol jerárquico de archivos y directorios del sistema de archivos HDFS, y se encarga de gestionar el correcto almacenamiento y réplica[[10]](#footnote-10) de la información que se almacena de forma distribuida en los diferentes nodos del cluster. Cuando un cliente desea acceder a los datos almacenados en el sistemas de archivos, se realiza una consulta hacia el NameNode, el cual contiene la información exacta de la ubicación de los datos, gracias al espacio de nombres y a la metadata del sistema de archivos.
   2. **JobTracker,**  es el encargado de recibir las solicitudes de ejecución de tareas MapReduce de los clientes. Una vez que reciba la tarea, el JobTracker se encarga de asignar y monitorear la ejecución de la misma hacia los nodos TaskTracker, y se comunica constantemente para determinar el estado en el que se encuentran las tareas asignadas. Así mismo el JobTracker posee mecanismos de recuperación de tareas que hayan sido mal ejecutadas o que se perdieron durante la comunicación entre los nodos. Al igual que el NameNode, este proceso se ejecuta únicamente en el nodo master del cluster.
   3. **Secondary NameNode,** este proceso no debe ser considerado como un respaldo del NameNode principal, ya que su función es la de realizar el mantenimiento de los archivos que conforman la metadata y los registros (logs) de los cambios realizados en el sistema de archivos HDFS.
2. **Nodos Slave:** El cluster de Hadoop puede estar conformado de uno a miles de nodos que se encargan de realizar las tareas de procesamiento y almacenamiento de información, mientras mayor cantidad de nodos, el rendimiento del sistema mejora. Es necesario tomar en cuenta que: el número de nodos tiene que ser proporcional a la cantidad y volumen de información que se va a procesar en el cluster. Los procesos correspondientes a la arquitectura HDFS y MapReduce son los siguientes:
   1. **DataNode,** son los nodos encargados de almacenar los bloques de datos en el sistema de archivos HDFS. Periódicamente notifican al NameNode la lista de bloques que están almacenando, y contienen las réplicas de datos de otros DataNodes en caso de algún inconveniente para obtener la información de un nodo.
   2. **TaskTracker,** es el responsable de crear instancias de las funciones map/reduce y notificar al JobTracker sobre el estado de la ejecución de las tareas asignadas. Es de vital importancia que este proceso conviva con el DataNode para que pueda acceder rápidamente a los datos que se encuentran almacenados en el sistema de archivos HDFS.

• **Arquitectura de Chukwa**

La arquitectura de Chukwa está compuesta de dos capas dentro de las cuales existen una variedad de componentes que cumplen con una función específica para realizar las tareas de monitoreo de una infraestructura de servidores.

Chukwa está diseñado para el monitoreo de múltiples clusters conformados por cientos de hosts, los mismos que generan varios terabytes de datos diariamente[11].

En la siguiente figura, se muestra la definición de las capas de la arquitectura y los componentes que conforman a las mismas:



*Figura 2.6: Arquitectura de Chukwa*

1. **Capa de Monitoreo:** Dentro de esta capa se encuentran todos los hosts que están distribuidos en toda la infraestructura de red, cada host o nodo monitoreado posee un proceso llamado *Agente Chukwa*, el mismo que se encarga de monitorear de manera constante el estado del host, los agentes representan el origen de los datos que van ser enviados a través de un pipeline hacia la capa de almacenamiento y análisis. Esta tarea es realizada por medio de los *Adaptadores Chukwa*, que son herramientas de línea de comando que ejecutan un tarea en particular; entre algunas comandos se encuentran: lectura de archivos log, estado del consumo de memoria y disco, estado del sistema operativo, métricas de uso de aplicaciones, entre otras utilidades.

Una vez que el adaptador ejecuta el comando, el agente de Chukwa se encarga de la comunicación y el streaming de datoshacia los componentes de la capa de almacenamiento y análisis.

1. **Capa de Almacenamiento y Análisis**

En esta capa se tiene el cluster de Chukwa, el mismo que se encuentra implementado sobre Hadoop, es por esto que el almacenamiento de los datos de monitoreo es gestionado por el sistema de archivos HDFS y el

procesamiento y análisis de estos datos se realiza utilizando tareas MapReduce específicas de Chukwa.

La forma en que Chukwa aprovecha las prestaciones de la plataforma Hadoop, es utilizando los siguientes componentes:

1. **Colector Chukwa:** Es un proceso que recibe los datos de un número particular de agentes, dentro de un cluster de Chukwa es recomendable tener un Colector por cada 100 agentes. Este componente gestiona la escritura de los datos dentro del sistema de archivos HDFS, escribiendo la información en un archivo llamado Sink File[11], el cual consiste en pedazos de información y la metadata con la descripción del origen y el formato de los datos. Periódicamente estos archivos se cierran, se renombran y se marcan para que estén disponibles para su posterior procesamiento.
2. **Análisis MapReduce:** Los colectores escriben en el sistema de archivos HDFS los Sink File de manera secuencial, lo cual es conveniente para obtener rápidamente los datos almacenados. Pero esto es menos conveniente al momento de realizar el análisis o búsqueda de determinados elementos, es por esto que Chukwa posee tareas MapReduce para organizar y procesar los datos que se encuentran dentro del sistema HDFS[12]. Estas tareas se dividen en dos: *Archivar y Desmultiplexar(DEMUX)*; la primera tarea, se encarga de tomar pedazos(Chuks) de datos de entrada y devolver como resultado nuevos pedazos de datos ordenados y agrupados. Por otra parte la tarea DEMUX, toma los pedazos de la tarea anterior y los procesa para generar los *Registros de Chukwa*, los mismos que son un conjunto de datos llave/valor que se almacenan ya sea en el sistema HDFS mediante los Sink Files o en una unidad de *almacenamiento estructurado* como puede ser una *base de datos relacional*.

## 2.4 CONCLUSIONES DEL CAPÍTULO

Es importante que en todo trabajo de investigación, exista un estudio preliminar de las herramientas que se van a utilizar para dar solución a un problema. Este capítulo se ha destinado para presentar los conceptos y definiciones principales de las tecnologías base de los sistemas de monitoreo tradiciones y distribuidos, para lo cual se ha elegido a Nagios y Chukwa como los representantes de estas tecnologías.

Es así que Nagios es un sistema multiplataforma de monitoreo basado en la utilización de plugins para la ejecución de chequeos sobre los objetos a ser monitoreados. Su arquitectura es de tipo cliente-servidor en donde existe un servidor centralizado que es el encargado de gestionar toda la lógica de análisis del estado de los host y servicios monitoreados. Una de las principales ventajas de este sistema es la extensa documentación existente para su configuración e implementación, así mismo, al utilizar estándares abiertos, facilita la integración y el desarrollo de sistemas en los que se requiera el núcleo de Nagios para su funcionamiento.

Por otra parte Chukwa, es una de las nuevas herramientas para el monitoreo de infraestructuras a gran escala, se enfoca principalmente en clusters que posean cientos de equipos físicos o virtuales los mismos que producen una variedad de datos y métricas sobre la utilización de sus recursos tanto en hardware como en software, por lo que es necesario un sistema de almacenamiento y análisis de grandes cantidades de información, es por esto que su arquitectura es soportada por la plataforma Hadoop, la cual se encuentra formada por dos componentes principales: el sistema de archivos distribuido HDFS y la implementación del modelo de programación MapReduce.

Con el estudio preliminar de estas tecnologías, es posible plantear la propuesta de mejora enfocándose en las ventajas y herramientas que brindan cada uno de estos sistemas de monitoreo.

# CAPÍTULO 3

# 3.PROPUESTA DE MEJORA

## 3.1 INTRODUCCIÓN DEL CAPÍTULO

Luego de realizar el estudio preliminar de los sistemas de monitoreo tradicionales y distribuidos, en este capítulo se presenta la propuesta de mejora la cual se especifica como objetivo principal del proyecto de tesis. La misma consiste en la combinación de componentes particulares de las arquitecturas tanto de Nagios como de Chukwa, tomando en cuenta la factibilidad tecnológica y las mejores prácticas para la implementación de sistemas de monitoreo. La propuesta establece tres pilares fundamentales: la *gestión de Infraestructura*, *gestión del Monitoreo* y la *gestión del Almacenamiento y Análisis*. En cada uno de estos pilares se va a realizar una comparación y confrontación de componentes frente a sus respectivas arquitecturas.

En las secciones de este capítulo se tiene en primer lugar una breve descripción de los trabajos que motivaron el desarrollo de este proyecto; luego se realiza el planteamiento de la solución, en donde se expone los detalles de cada uno de los componentes que conforman la propuesta de mejora y los requerimientos para la implementación de un prototipo funcional utilizando las tecnologías de los sistemas de monitoreo; para la finalización del capítulo, se realiza la evaluación de los atributos de calidad de las arquitecturas basándose en la confrontación de características de los componentes de los sistemas de monitoreo y la solución propuesta.

## 3.2 TRABAJOS RELACIONADOS

Existen algunos proyectos sobre el estudio de los sistemas de monitoreo y el análisis de incidentes en centros de datos; entre algunos de estos se destacan:

* Pervilä M.[15], describe las posibilidades de interconectar Nagios con otras aplicaciones para facilitar y automatizar la recuperación después de presentarse fallas en los servicios, el autor propone el uso de *manejadores de eventos(event handlers)* en todos los cambios de estado de los servicios, permitiendo así ejecutar acciones proactivas ante cualquier anomalía; además de esto sugiere la implementación de servidores Nagios redundantes para infraestructuras a gran escala, con el fin de evitar fallos en el mismo servidor de monitoreo.
* Sahoo R. et al.[30], analizan las propiedades empíricas y estadísticas de errores del sistema y fallos de una red de casi 400 servidores de características heterogéneas, gracias a los registros log que se almacenan en cada uno de los servidores, como resultado se demuestra que los errores del sistema y patrones de falla se componen de un comportamiento variable en el tiempo y largos intervalos fijos, los mismos que impactan directamente en el rendimiento de la red de servidores.
* Jia B. et al.[14], realizan la investigación de la plataforma Hadoop para el análisis de la información que almacenan las industrias de Gas y Aceite, y utilizan el sistema de monitoreo Chukwa para desarrollar las pruebas de adquisición de datos que serán procesados en el cluster de Hadoop y así demostrar la eficiencia de esta plataforma en el procesamiento de archivos de gran tamaño.
* Entre los artículos que estudian las diferentes tecnologías planteadas en este proyecto se tiene a: Dean J.[6], con la implementación del modelo de programación MapReduce; Shvachko K. et al.[7] con el estudio del sistema de archivos de Hadoop HDFS; Boulon J. et al.[10] y Rabkin A.[11] presentan a Chukwa como un sistema de monitoreo a gran escala.

## 3.3 PLANTEAMIENTO DE LA SOLUCIÓN

Como se menciona en la introducción del capítulo, la propuesta de mejora consiste en la combinación de los componentes arquitectónicos de los sistemas de monitoreo tradicionales y distribuidos. La principal motivación para el desarrollo de esta propuesta es la de demostrar la forma en que dos arquitecturas pueden colaborar entre si para generar una base de conocimientos que sirve para el análisis del comportamiento de los componentes dentro una infraestructura de red.

La información recolectada se puede utilizar para descubrir patrones de

comportamiento, estadísticas y métricas de uso y acceso a los sistemas

### 3.3.1. Vista de componentes de la solución

En la siguiente imagen se muestra la vista de componentes que conforman la propuesta de la solución:

*Figura*

*3*

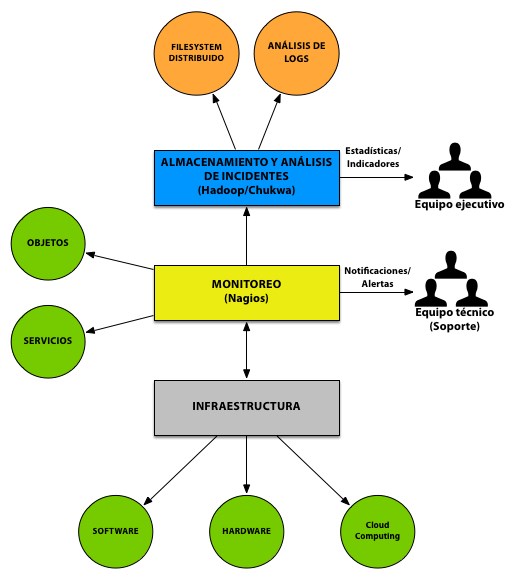
*.*

*1*

*:*

*Vista de componentes de la*

*solución*



* **Infraestructura**

Esta capa es el punto inicial de la arquitectura y donde se proveen los datos de origen para las capas superiores, aquí se encuentra el hardware, software y servicios de Cloud Computing que conforman la infraestructura de red que posee una empresa o negocio. La función principal es proveer a la capa de monitoreo el estado de los hosts y servicios que se alojan en la infraestructura.

Es importante destacar que la arquitectura propuesta se encuentra orientada a infraestructuras distribuidas, que posean múltiples centros de datos (datacenters) ubicados en distintas zonas geográficas.

* **Monitoreo**

En esta capa se encuentra el sistema de monitoreo encargado de vigilar

constantemente los elementos que se encuentran en la capa de infraestructura. Por cada centro de datos que posea una empresa, se va tener un agente de monitoreo que recolecta las alertas de cada uno de sus objetos (hosts y servicios).

Las alertas son visualizadas por el *Equipo Técnico*, que son los encargados del soporte, la resolución y gestión de los incidentes dentro de la infraestructura.

Para la gestión de alertas e incidentes, se propone la utilización de Nagios en su configuración distribuida[18]. Esta configuración permite gestionar de forma descentralizada el monitoreo de cada datacenter, por lo que toda la información que se genera, será recolectada y enviada a la capa de Almacenamiento y análisis de incidentes.

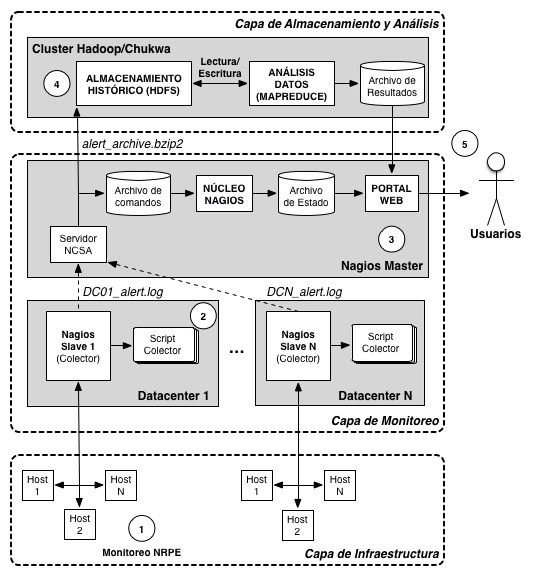
• **Almacenamiento y análisis de incidentes**

En esta capa se propone la utilización de la plataforma Hadoop, es decir la utilización del sistema de archivos distribuido HDFS y el modelo de programación MapReduce, esto con el fin de tener de manera independiente y descentralizada un sistema de almacenamiento que contiene la información recolectada por la capa de monitoreo sobre los incidentes y alertas que ocurran en la infraestructura.

Esta información se almacena como archivos de texto plano, para luego ser procesados y analizados utilizando tareas MapReduce que están enfocadas hacia el descubrimiento de patrones de comportamiento y métricas de utilización de recursos, cuyos resultados son de utilidad para el *Equipo Ejecutivo* de la empresa.

### 3.3.2. Arquitectura de la solución

Una vez analizados los componentes generales del esquema de solución propuesta, en este apartado, se especifica la arquitectura, tomando en cuenta los elementos dentro de cada capa, los detalles técnicos y el flujo de información que representa la interacción de los componentes de la arquitectura. (fig. 3.2)



*Figura 3.2: Arquitectura de la solución*

1. Cada uno de los host y servicios que se encuentran dentro de la capa de infraestructura poseen el servicio NRPE de Nagios[17], el mismo que se encarga de la gestión de los plugins que realizan el chequeo de los servicios dentro de los hosts. Los resultados de los chequeos son enviados hacia un servidor de Nagios denominado *Nagios Slave*, este servidor realiza el monitoreo de un determinado número de hosts que se encuentran dentro de un centro de datos o una serie de nodos que conforman un cluster de computadores.
2. Como se menciona en un principio, se propone la utilización de un servidor Nagios Slave por cada centro de datos que posea la infraestructura de servidores, de acuerdo a la arquitectura de Chukwa[11], se aplica el concepto de *Colector* a estos servidores de Nagios esclavos, ya que son el punto de recepción de los datos de origen que se generan en cada uno de los hosts y servicios de la capa de infraestructura. La tarea de recolección de alertas se ejecuta mediante un script(ver Anexo 2), el mismo que tiene la función de escribir en un archivo de texto plano, todos los resultados de los chequeos enviados por el servicio NRPE. Se han definido los siguientes campos que representan el formato del archivo de alertas:

|  |  |
| --- | --- |
| **Campo** | **Descripción** |
| **Fecha** | Formato tipo dd/mm/aa |
| **Hora** | Formato tipo HH/MM/SS |
| **Hostname** | El nombre corto del host a ser monitoreado |
| **Servicio** | El nombre corto del servicio a ser monitoreado |
| **Estado** | Identificador del estado de la alerta, 0 = OK, 1=WARNING,  2=CRITICAL |
| **Descripción** | Mensaje detallado sobre el estado de la alerta |
| **Grupo de Host** | Grupo al que pertenece el host monitoreado |

*Tabla 3.1: Descripción de campos del archivo Colector*

Cada determinado tiempo, el archivo se cierra, se renombra y se envía a través de la red hacia el servidor Nagios Master, el mismo que recibe las alertas de todos los colectores de cada uno de los datacenters mediante el complemento de Nagios llamado NCSA.

1. El complemento NCSA, permite a los colectores enviar de forma pasiva las alertas de los hosts y servicios dentro de su respectivo datacenter, para luego ser analizadas por el núcleo de Nagios y generar el archivo de estados para poder emitir las alertas y presentarlas en el portal web que viene incluido en el sistema de Nagios.

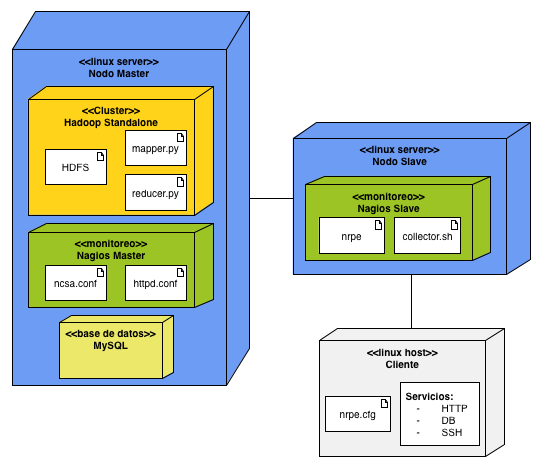
De forma paralela, al recibir el archivo de alertas de cada uno de los colectores, el servidor Nagios Master, es el encargado de unir estos archivos y almacenarlos en el cluster de Hadoop, esta tarea se realiza utilizando un script(ver Anexo 2), que utiliza las librerías de la plataforma Hadoop para la comunicación con el cluster.

1. Dentro de la capa de Almacenamiento y Análisis, se tiene el sistema de archivos HDFS, que constituye el sistema de almacenamiento histórico de incidentes de toda la infraestructura de servidores. Al igual que en la capa de monitoreo, periódicamente se ejecutan una tarea MapReduce(ver Anexo 2), la misma que consiste en procesar los archivos que se almacenan en el sistema, y el resultado final del análisis se guarda en una base de datos relacional, para posteriormente ser utilizada para la interpretación de resultados.
2. El último paso consiste en la lectura de los resultados almacenados en la base de datos relacional y presentarlos al usuario en el portal web del sistema de monitoreo, a futuro se puede implementar gráficas que representen la interpretación de los resultados sobre las anomalías de los componentes de una infraestructura de servidores.

### 3.3.3. Guía general para la implementación arquitectónica

Para dar soporte a la factibilidad técnica del diseño de la arquitectura propuesta, en esta sección se presenta una guía general para su implementación, reflejada en un prototipo funcional, el mismo que contiene los principales componentes de la arquitectura. El prototipo esta conformado por el diagrama de despliegue, el hardware y software utilizado y un esquema de pruebas acorde a las capas en las que se encuentra divida la arquitectura. Es importante aclarar, que en el siguiente capítulo, se explica con mayor detalle la fase de implementación del prototipo funcional.

* Se utiliza el diagrama de despliegue[22], para representar la arquitectura propuesta desde el punto de vista de la distribución de componentes de hardware y software, en donde se especifican tres nodos: el ***nodo master*** que contiene el servidor de Nagios master, el cluster de Hadoop en modo Standalone[[11]](#footnote-11) y una base de datos relacional (MySQL); el ***nodo slave*** que representa al colector de alertas y que también posee una instalación de Nagios básica para monitorear los servicios del ***cliente*** tales como: web, base de datos, SSH y los componentes de hardware como disco duro, memoria, carga del CPU, memoria SWAP, entre otros. En la siguiente figura, se muestra el diagrama de despliegue del prototipo funcional:



*Figura 3.3: Diagrama de despliegue del prototipo funcional*

* El prototipo funcional utiliza los siguientes componentes de Hardware y Software:

**Hardware:**

* + Computadores de escritorio con hardware de características genéricas.o Red de área local (LAN) para la interconexión de los computadores.

**Software:**

* + Sistema operativo Linux.
  + Sistema de monitoreo Nagios (herramientas de monitoreo, complementos

NRPE y NSCA, plugins y servidor web)o Base de datos MySQL.

* + La plataforma Hadoop con todas sus librerías.o Herramientas de desarrollo y shell script.
* Una vez definidos los componentes y herramientas a utilizar en el prototipo funcional, a continuación se describe el esquema de pruebas en base a las tres capas de la arquitectura propuesta:

|  |  |
| --- | --- |
| **Capa** | **Descripción/Propósito de las pruebas** |
| **Infraestructura** | En esta capa se tiene al nodo *Cliente*, el mismo que posee los servicios básicos que son monitoreados y son la fuente de datos de la arquitectura. Las pruebas realizadas se basan en la disponibilidad de servicios, estado de los componentes de hardware del equipo (memoria, cpu, almacenamiento) y pruebas de estrés para generar alertas y sean reportadas al sistema Nagios. |
| **Monitoreo** | El objetivo de las pruebas consiste en demostrar que las alertas que se presentan en la capa de infraestructura, son monitoreadas y recolectadas por el *nodo slave* y escritas en un archivo de texto plano que  periódicamente se enviará al *nodo master*.  Al recibir el archivo de alertas, en el *nodo master*, se realizarán pruebas de escritura hacia el cluster de Hadoop, y se ejecutará el script encargado de cerrar los archivos recolectados y almacenarlos en el cluster. |
| **Almacenamiento y**  **Análisis** | Para la parte de almacenamiento, las pruebas se basan en la comprobación de integridad de los archivos que se almacenan en el sistema HDFS, es importante aclarar que al contar con un solo nodo, no es posible realizar pruebas de alta disponibilidad y tolerancia a fallos, ya que como mínimo se requiere un cluster de tres nodos, para verificar la replicación de los datos. Con respecto al análisis de los datos, las pruebas se realizan al verificar los resultados de las tareas MapReduce encargadas de analizar y procesar los archivos de alertas que se encuentran almacenados en el sistema HDFS, una vez analizados, los datos se almacenan en un tabla dentro de la base de datos relacional. |

*Tabla 3.2: Esquema de pruebas del prototipo funcional*

## 3.4 EVALUACIÓN DE LA ARQUITECTURA

En cualquier proyecto informático que se desarrolle utilizando metodologías formales de diseño, la evaluación de la arquitectura es de vital importancia para el éxito del funcionamiento del sistema, es por esto que las decisiones de diseño deben enfocarse a determinar el impacto que existe con respecto a los atributos de calidad[[12]](#footnote-12).

En este caso, la evaluación consiste en tomar los atributos de calidad más importantes basándose en las mejores prácticas y técnicas de evaluación de arquitecturas[21], para confrontar las características de los componentes arquitectónicos y determinar los escenarios que definen los criterios de aceptación y cumplimiento de estos.

* **Identificación de atributos**

Los atributos identificados de alta prioridad para la arquitectura propuesta son:

* + *Disponibilidad*. Los sistemas de monitoreo deben vigilar constantemente los host y servicios de una infraestructura, por lo que es de vital importancia la disponibilidad del sistema, con el fin de poder detectar a tiempo los incidentes que ocurran dentro de la infraestructura.
  + *Performance*. Al haber incorporado a esta arquitectura una capa de almacenamiento y análisis, es importante que el sistema ejecute estas tareas con un tiempo de respuesta óptimo en la generación de métricas y patrones de comportamiento de la infraestructura.
  + *Escalabilidad*. Este atributo representa una de las características más importantes de los sistemas distribuidos, por lo cual el sistema debe implementar métodos para el crecimiento de su capacidad tanto en procesamiento como en almacenamiento.
* **Árbol de utilidad de los atributos de calidad**

Esta técnica tomada del método de evaluación de arquitecturas ATAM[23], sirve para determinar los escenarios en los que el sistema debe responder para cumplir con los atributos de calidad que caracterizan a los sistemas de monitoreo. Este árbol sirve como guía para analizar aspectos de la arquitectura que son de interés para los stakeholders[[13]](#footnote-13) del sistema.

En la tabla 3.3 se presenta el árbol de utilidad de los atributos antes identificados:

|  |  |  |  |
| --- | --- | --- | --- |
| **Atributo de Calidad** | **Refinamiento del Atributo** | **Escenarios** | |
| Disponibilidad | Tolerancia a fallos | * Detección y recuperación ante el fallo en uno de los servidores Nagios Slave   dentro de un datacenter.   * Verificación de la integridad de la información ante la pérdida de un nodo que forma parte del sistema de archivos HDFS. * Recuperación ante el fallo del servidor principal Nagios Master. | |
| Performance | Tiempo de Respuesta | • Realizar pruebas para el procesamiento de archivos log de más de 500MB y medir el desempeño del cluster durante la ejecución de las tareas MapReduce, tanto en lectura como en escritura de datos. | |
| Escalabilidad | Crecimiento del sistema | • El sistema de archivos de archivos distribuido puede escalar fácilmente en | |
|  |  |  | su capacidad de almacenamiento al  agregar nuevos nodos al cluster. |
|  |  | • | La configuración distribuida de Nagios, permite la implementación de múltiples servidores esclavos para segmentar la infraestructura y distribuir la carga a un solo servidor centralizado. |

*Tabla 3.3: Árbol de utilidad de los atributos de calidad*

## 3.5 CONCLUSIONES DEL CAPÍTULO

La principal motivación para definir la propuesta de mejora de los sistemas de monitoreo es la de encontrar una alternativa para la combinación de técnicas, herramientas y componentes de la arquitectura de un sistema tradicional y otro con características cien por ciento distribuidas. Para esto es de vital importancia tener en claro cual va ser la interacción de cada uno de los componentes arquitectónicos y que esta propuesta se vea reflejada en un prototipo funcional que sirva como modelo para una futura implementación.

Así mismo, la evaluación de la arquitectura propuesta permite identificar cuales son los atributos de calidad que deberían cumplirse para poder satisfacer los requerimientos que buscan los stakeholders, es por esto que los escenarios ofrecen criterios de aceptación para el desarrollo e implementación de un sistema óptimo y eficaz.

En el siguiente capítulo se explica en detalle la implementación del prototipo y las diferentes pruebas que se realizaron para validar los resultados de la evaluación de la arquitectura propuesta.

# CAPÍTULO 4

# 4.IMPLEMENTACIÓN

## 4.1 INTRODUCCIÓN DEL CAPÍTULO

Al haber realizado el planteamiento de la solución, este capítulo describe las especificaciones técnicas y pruebas realizadas durante la fase de implementación del prototipo funcional, en donde se destacan los componentes de hardware utilizados, sistema operativo, especificaciones de software, librerías y paquetes adicionales, lenguajes de programación, utilitarios y topología de red.

Además, se muestra como se ejecuta el proceso de recolección de datos de incidentes y los resultados obtenidos durante la ejecución del experimento para validar la arquitectura propuesta.

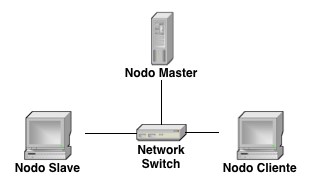
## 4.2 IMPLEMENTACIÓN DEL PROTOTIPO FUNCIONAL

Este prototipo ha sido implementado y desplegado sobre una infraestructura de pruebas constituida por componentes de hardware, software y máquinas virtuales; cada uno cumple con una función específica acorde a la arquitectura que representa la combinación de los sistemas de monitoreo tradicionales y distribuidos.

A continuación se presenta la descripción detallada de los componentes:

**Hardware**

* Una máquina virtual(VM) denominada *Nodo Master*; la misma que está configurada con un procesador Intel Core i7 de 2.7GHz de un solo núcleo, 2GB de RAM y 20GB de almacenamiento. En esta máquina virtual se ha implementado el cluster de Hadoop en modo Standalone o no distribuido, tanto con el sistema de archivos HDFS y los procesos para la ejecución de tareas MapReduce. Así mismo esta VM posee una instalación de Nagios estándar, con todos los complementos y plugins para recibir las notificaciones y alertas sobre los incidentes desde el nodo slave.
* Un computador físico denominado *Nodo Slave*; este posee un procesador Intel Pentium 4 de 2.66Ghz, 512 MB de RAM y 120 GB de disco duro. Este nodo esclavo posee una instalación de Nagios básica con los componentes NRPE para el monitoreo del cliente y las librerías estándar de Hadoop para la comunicación y almacenamiento de los logs de alertas en el nodo master.
* Una máquina virtual que representa al *Nodo Cliente*; esta VM se ejecuta con un procesador AMD Radeon de 32 bits, 1GB de RAM y 8GB de disco duro. El cliente es monitoreado por el nodo slave, y posee los servicios básicos para ejecutar las pruebas de recolección de alertas.
* La topología de red utilizada es de tipo Bus, en donde los dispositivos utilizan un único canal de comunicaciones para la interconexión de los mismos a través de un switch o conmutador de capa 3.



*Figura 4.1: Topología de Red para las pruebas*

**Software**

* El sistema operativo utilizado en cada uno de los nodos es CentOS en su versión 6, con una arquitectura de 64 bits en el nodo master y 32 bits en los nodos slave y cliente.
* La versión de Nagios utilizada en los nodos master y slave es la 3.5



*Figura 4.2: Pantalla de inicio de Nagios*

* Para la plataforma Hadoop, se utilizó una distribución libre desarrollada por la empresa Cloudera[24], la misma que ofrece servicios de soporte y capacitación para toda la plataforma Hadoop, posee una gran cantidad de recursos y documentación para la implementación de la plataforma. En el prototipo se hace uso de la versión 0.20.2-cdh3u6.
* Entre otros programas utilizados, tenemos el servidor web Apache, la base de datos MySQL y el lenguaje de programación Python como herramientas de desarrollo de scripts para el procesamiento de la información.

## 4.3 EJECUCIÓN Y RESULTADO DE LAS PRUEBAS

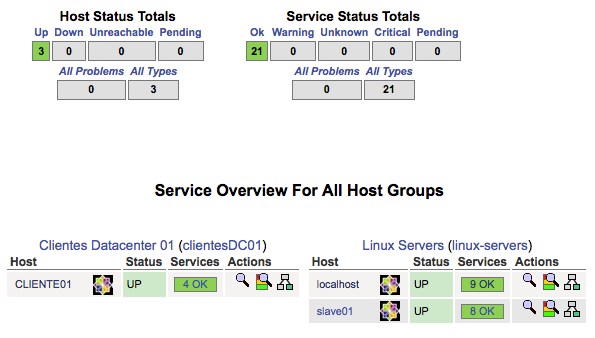
Luego de la instalación e implementación del prototipo funcional, se define un esquema de pruebas de cada uno de los componentes que conforman la arquitectura, en la siguiente tabla se muestra el detalle del esquema utilizado para la ejecución de las pruebas:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Prueba** |  | | **Descripción** | **Componentes involucrados** | **Resultados** |
| **Monitoreo cliente** | **del** | | Definir los servicios a ser monitoreados por el Nodo  Slave. | Nodo Slave  Nodo Cliente | Los servicios configurados en el cliente son: espacio en disco, carga de CPU, memoria y servidor web Apache. |
| **Recolección**  **Alertas** | **de** | | Comprobar que los chequeos se almacenan en un archivo de texto plano previamente definido. | Nodo Slave | Se ha definido un directorio dentro del Nodo Slave para almacenar los archivos log que contienen los resultados de los chequeos al cliente, el sistema Nagios es el encargado de la creación y escritura de los registros en el archivo de texto. |
| **Notificación**  **Alertas** | **de** | | Por cada chequeo realizado a los clientes, se envía al Nodo Master el estado del servicio mediante el complemento  NSCA | Nodo Slave  Nodo Master | Se puede comprobar en los logs del nodo master que los chequeos realizados están siendo recibidos por el complemento NSCA y se puede visualizar en el portal web de Nagios el estado del cliente. |
| **Almacenamiento de Alertas** | |  | Configurar un script de rotación para realizar el streaming de los archivos log hacia el sistema HDFS que se encuentra en el Nodo Master. | Nodo Slave  Nodo Master  Hadoop HDFS | El nodo slave tiene configurado una tarea cron[[14]](#footnote-14), en donde cada 20 minutos se ejecuta un script, el mismo que tiene la función de cerrar el archivo de alertas log y darle un nombre con un formato en particular, que contiene el nombre del nodo slave, la fecha y hora en la que se cerró el archivo. Luego el nodo slave utiliza las librerías de hadoop para realizar el envío o streaming del archivo log de alertas. Una vez enviado el archivo, se procede a la rotación del mismo. |
| **Análisis de**  **Alertas** | | | Realizar pruebas del proceso de análisis y configurar su frecuencia de ejecución. | Nodo Master  HDFS  MapReduce | El proceso de análisis de alertas consiste en un script desarrollado en el lenguaje  Python, el mismo que ejecuta una tarea  MapReduce para clasificar y contabilizar el número de incidentes, tomando en cuenta el nombre del host y el servicio monitoreado. Los archivos analizados se mueven a un directorio que sirve como historial para futuras consultas. |

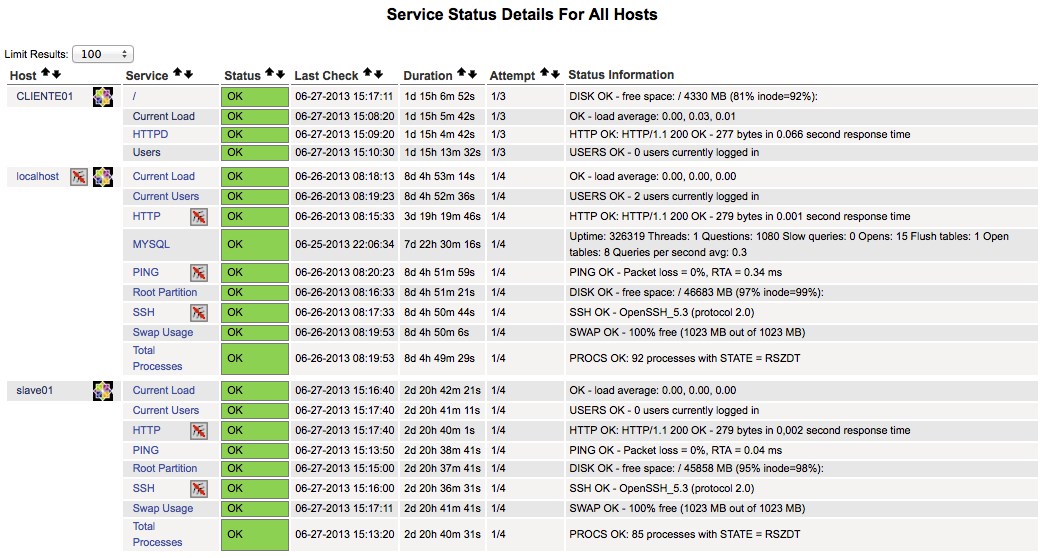
*Tabla 4.1: Esquema de pruebas del prototipo funcional*

En las siguientes capturas de pantalla, se muestra el funcionamiento del prototipo funcional, empezando por el sistema Nagios, la visualización de alertas y los archivos log que se almacenan en el cluster de Hadoop:

**NAGIOS**



*Figura 4.3: Visualización de hosts desde Nagios Master*



*Figura 4.4: Visualización del estado de los servicios en Nagios*

*Figura*

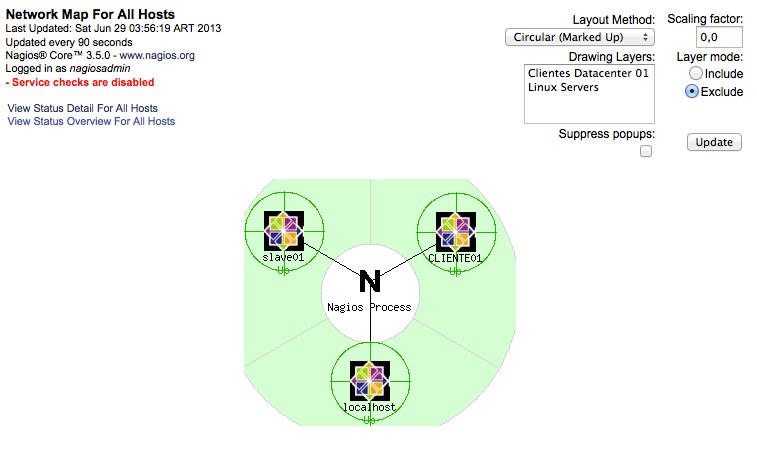
*4*

*.*

*5*

*:*

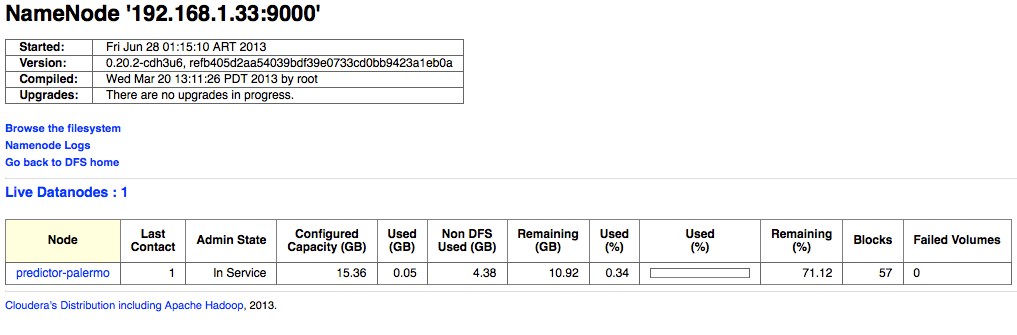
*Visualización del mapa de red Nagios*



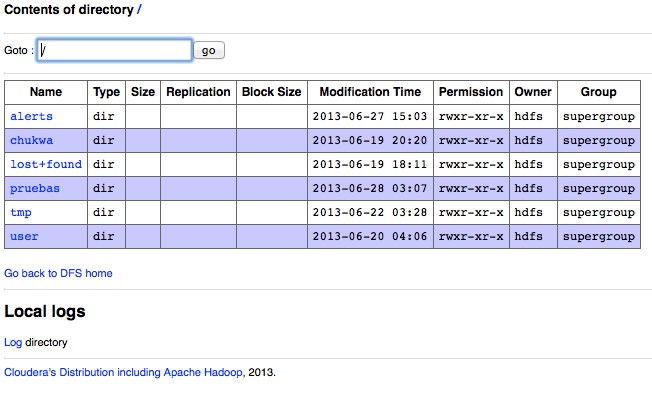
**HADOOP**



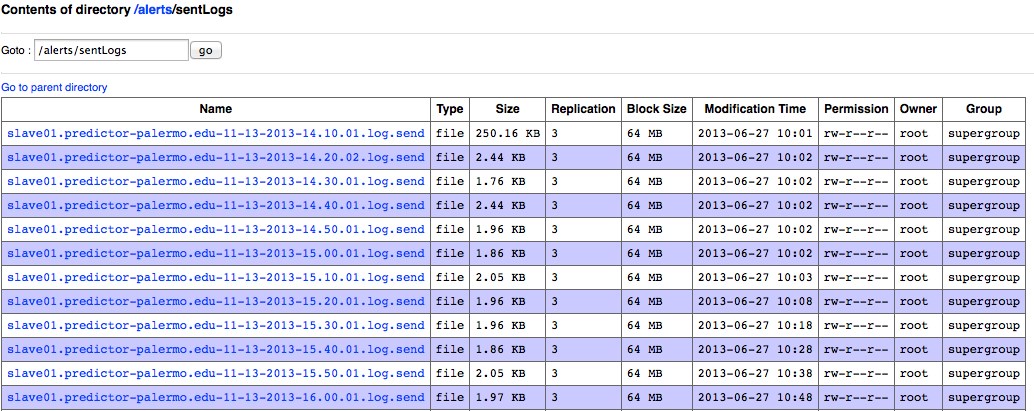
*Figura 4.6: Información del Namenode y el cluster de Hadoop*



*Figura 4.7: Información del Datanode configurado en el cluster*



*Figura 4.8: Estructura del directorio raíz del sistema HDFS*



*Figura 4.9: Directorio de almacenamiento de archivos log en el sistema HDFS*

Para la ejecución de tareas MapReduce, se utilizan la librería Hadoop Streaming API, la misma que recibe como parámetros: el código fuente de los métodos map y reduce (ver Anexo 3), la ruta del directorio dentro del sistema HDFS que contiene los datos de entrada y la ruta donde se van a almacenar los resultados del análisis:

hadoop jar **contrib/streaming/hadoop-\*streaming\*.jar** -mapper

/home/hadoop/tesis/**mapper.py** -reducer

/home/hadoop/tesis/**reducer.py** -input **/alerts/sentLogs/\*** -output

**/alerts/tmp/output-test/**

*Comando para la ejecución de la tarea MapReduce*

Al ejecutar la tarea MapReduce dentro del Nodo Master, los resultados que se pueden visualizar en la línea de comandos son los siguientes:

packageJobJar: [/tmp/hadoop-hdfs/hadoop-unjar5868278976075733163/] [] /tmp/streamjob883102395475620376.jar tmpDir=null

13/06/29 14:20:03 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

13/06/29 14:20:03 WARN snappy.LoadSnappy: Snappy native library not loaded

13/06/29 14:20:04 INFO mapred.FileInputFormat: **Total input paths to process : 25**

13/06/29 14:20:05 INFO streaming.StreamJob: getLocalDirs(): [/tmp/hadoop-hdfs/mapred/local]

13/06/29 14:20:05 INFO streaming.StreamJob: Running job: job\_201306280115\_0021

13/06/29 14:20:05 INFO streaming.StreamJob: To kill this job, run:

13/06/29 14:20:05 INFO streaming.StreamJob: /usr/lib/hadoop0.20/bin/hadoop job -Dmapred.job.tracker=localhost:9001 -kill job\_201306280115\_0021

13/06/29 14:20:05 INFO streaming.StreamJob: **Tracking URL: http://localhost:50030/jobdetails.jsp?jobid=job\_201306280115\_0021** 13/06/29 14:20:06 INFO streaming.StreamJob: map 0% reduce 0%

13/06/29 14:20:30 INFO streaming.StreamJob: map 4% reduce 0%

13/06/29 14:20:31 INFO streaming.StreamJob: map 8% reduce 0%

13/06/29 14:20:36 INFO streaming.StreamJob: map 12% reduce 0%

13/06/29 14:20:37 INFO streaming.StreamJob: map 15% reduce 0%

13/06/29 14:20:41 INFO streaming.StreamJob: map 19% reduce 0%

13/06/29 14:20:42 INFO streaming.StreamJob: map 23% reduce 0%

13/06/29 14:20:46 INFO streaming.StreamJob: map 27% reduce 0%

13/06/29 14:20:47 INFO streaming.StreamJob: map 31% reduce 0%

13/06/29 14:20:48 INFO streaming.StreamJob: map 31% reduce 9%

13/06/29 14:20:51 INFO streaming.StreamJob: map 38% reduce 9%

13/06/29 14:20:55 INFO streaming.StreamJob: map 42% reduce 9%

13/06/29 14:20:56 INFO streaming.StreamJob: map 46% reduce 9%

13/06/29 14:20:57 INFO streaming.StreamJob: map 46% reduce 13%

13/06/29 14:21:01 INFO streaming.StreamJob: map 54% reduce 13%

13/06/29 14:21:03 INFO streaming.StreamJob: map 54% reduce 18%

13/06/29 14:21:05 INFO streaming.StreamJob: map 58% reduce 18%

13/06/29 14:21:06 INFO streaming.StreamJob: map 62% reduce 18%

13/06/29 14:21:10 INFO streaming.StreamJob: map 65% reduce 18%

13/06/29 14:21:11 INFO streaming.StreamJob: map 69% reduce 18%

13/06/29 14:21:12 INFO streaming.StreamJob: map 69% reduce 21%

13/06/29 14:21:14 INFO streaming.StreamJob: map 73% reduce 21%

13/06/29 14:21:15 INFO streaming.StreamJob: map 77% reduce 21%

13/06/29 14:21:18 INFO streaming.StreamJob: map 81% reduce 26%

13/06/29 14:21:20 INFO streaming.StreamJob: map 85% reduce 26%

13/06/29 14:21:22 INFO streaming.StreamJob: map 88% reduce 26%

13/06/29 14:21:24 INFO streaming.StreamJob: map 92% reduce 26%

13/06/29 14:21:27 INFO streaming.StreamJob: map 96% reduce 32%

13/06/29 14:21:28 INFO streaming.StreamJob: map 100% reduce 32%

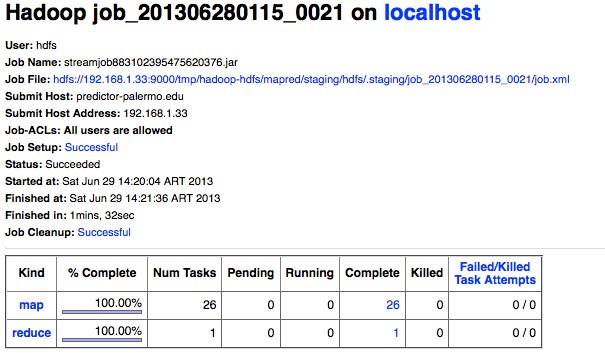
13/06/29 14:21:35 INFO streaming.StreamJob: map 100% reduce 100% 13/06/29 14:21:37 INFO streaming.StreamJob: **Job complete**: job\_201306280115\_0021

13/06/29 14:21:37 INFO streaming.StreamJob: **Output**:

/alerts/tmp/output-test/

*Salida del comando para la ejecución de la tarea MapReduce*

Durante la ejecución de la tarea MapReduce, el sistema genera una URL(http://192.168.1.33:50030/jobdetails.jsp?jobid=job\_201306280115\_0021), para realizar el seguimiento de la tarea, una vez finalizada, los resultados se actualizan de manera automática:



*Figura 4.10: Resultado de la ejecución de una tarea MapReduce*

En la figura anterior, se puede observar que el número de tareas Map es de 26, cuyo valor representa a la cantidad de archivos log que se encuentran en el directorio especificado en el comando. Para el proceso Reduce, se tiene únicamente una sola tarea, por defecto la cantidad de estas tareas se establece de acuerdo al número de nodos que existen en un cluster, al ser un prototipo de pruebas, solamente se tiene un solo nodo para la ejecución de las mismas.

Para verificar los resultados, se puede utilizar los comandos de las librerías de Hadoop para imprimir en pantalla el archivo que se crea al finalizar con la

clasificación de las alertas existentes en los archivos log:

hadoop fs -cat /alerts/tmp/output-test/part-00000

*Comando para imprimir el contenido del archivo de resultados*

Gracias al comando *cat*, se puede visualizar en pantalla el contenido del archivo *part00000,* el cual contiene en texto plano, los resultados del análisis de los archivos log, a continuación se presenta una porción del archivo con el conteo de alertas encontradas y clasificadas por: fecha, host, servicio y estado:

…

2013-11-11:CLIENTE01:/:OK 31

|  |  |  |
| --- | --- | --- |
| 2013-11-11:CLIENTE01:Current Load:**CRITICAL** | | 2 |
| 2013-11-11:CLIENTE01:Current Load:OK | 30 |
| 2013-11-11:CLIENTE01:HTTPD:**CRITICAL**  2013-11-11:CLIENTE01:HTTPD:OK 22 | 12 |  |
| 2013-11-11:CLIENTE01:Users:**CRITICAL**  2013-11-11:CLIENTE01:Users:OK 30 | 2 |  |
| 2013-11-11:slave01:Current Load:OK | 62 |  |
| 2013-11-11:slave01:Current Users:OK  2013-11-11:slave01:HTTP:OK 63  2013-11-11:slave01:PING:OK 62 | 62 |  |
| 2013-11-11:slave01:Root Partition:OK  2013-11-11:slave01:SSH:OK 62  2013-11-11:slave01:Swap Usage:OK 62 | 63 |  |
| 2013-11-11:slave01:Total Processes:OK 62 | |

…

*Contenido del archivo de resultados*

## 4.4 CONCLUSIONES DEL CAPÍTULO

Durante la fase de implementación del prototipo funcional, se han identificado los beneficios de utilizar las herramientas que nos brinda la plataforma Hadoop, como son: la facilidad de integración con otras plataformas para la obtención y procesamiento de información, la posibilidad de utilizar cualquier lenguaje de programación para el uso de las librerías de Hadoop para la comunicación con el sistema de archivos distribuido, y sobretodo la capacidad de escalabilidad de recursos bajo demanda lo cual representa una solución ideal para una implementación a gran escala usando la tecnología Cloud Computing.

Así mismo se ha podido comprobar, que el modelo de programación MapReduce utilizado para el análisis de los logs de alertas, establece las bases para la implementación de sistemas de procesamiento de archivos de texto, tanto estructurados como no estructurados, utilizando su capacidad de paralelismo, escalabilidad y el máximo aprovechamiento de los recursos tanto de hardware como software.

En el último de capítulo de esta tesis se presenta la discusión sobre los resultados obtenidos y cual es el futuro de las tecnologías utilizadas en este proyecto.

# CAPÍTULO 5

# 5.DISCUSIÓN GENERAL DEL PROYECTO

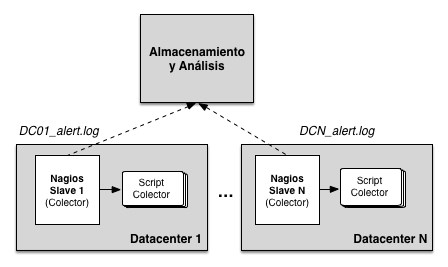
Este capítulo muestra una discusión a nivel general sobre los resultados obtenidos a lo largo del desarrollo de este proyecto, tomando como referencia las experiencias que se han dado en los trabajos relacionados señalados anteriormente. Así mismo, se plantean las conclusiones finales del proyecto; y el trabajo futuro en donde se expresa los criterios y recomendaciones que determinan la factibilidad hacia la implementación de la arquitectura propuesta a gran escala.

## 5.1 DISCUSIÓN

Gracias a la investigación previa de proyectos relacionados con: sistemas de monitoreo; análisis de anomalías en centros de datos; estudios sobre la plataforma Hadoop y Chukwa; se ha conseguido el desarrollo de la presente tesis. Es así que la propuesta de mejora de los sistemas de monitoreo, brinda como aporte las estrategias y técnicas para la integración de dos arquitecturas, que en conjunto permiten: la organización de grandes infraestructuras de servidores, la correcta gestión del monitoreo de incidentes y el almacenamiento histórico de alertas para su posterior análisis.

En base a los objetivos establecidos al inicio del proyecto, se detallan los siguientes puntos de discusión:

* Los sistemas de monitoreo distribuidos cumplen un rol importante en empresas en las que su mayor prioridad es la disponibilidad de los servicios, ya que la función de estos sistemas es mantener informado y comunicar constantemente el estado en el que se encuentran los componentes de una infraestructura. La implementación de un sistema de monitoreo distribuido nos beneficia en la creación de múltiples centros de control, disminuyendo así el tráfico de información a un solo servidor principal, es así que cada uno de estos centros realizan de forma individual las tareas de monitoreo a los hosts y servicios que están bajo su supervisión y al presentarse alguna anomalía o incidente, se notifica al servidor principal para generar las alertas correspondientes. En este punto la arquitectura propuesta, adopta en uno de sus componentes el concepto de *Colector de alertas*, implementados en los nodos slave, los que se encargan de guardar la información sobre los incidentes en archivos de texto plano y con una determinada frecuencia se envían hacia el sistema de almacenamiento y análisis.



*Figura 5.1: Componentes del colector de alertas*

* Las herramientas de almacenamiento y análisis, representan uno de los componentes de mejora de los sistemas de monitoreo tradicionales; para la parte de almacenamiento, los sistemas distribuidos tienen la ventaja de que la información se puede establecer en múltiples ubicaciones geográficas, lo cual permite la escalabilidad de los recursos, así como también, la posibilidad de crear un repositorio en el que se almacenen los registros históricos de todos los incidentes, para luego utilizar las herramientas de análisis de datos para el procesamiento de estos bancos de información. Dentro del proyecto se ha tomado para la gestión del almacenamiento el sistema de archivos HDFS[7], el cual posee entre las principales ventajas la capacidad de almacenar grandes cantidades de información, divididas en bloques de datos de 64MB, y la fiabilidad del sistema se consigue mediante la replicación de los datos a través de los múltiples nodos que conforman el cluster, por lo cual no se requiere la implementación de hardware o software de redundancia como por ejemplo la tecnología RAID[[15]](#footnote-15).
* Con respecto al análisis de la información, la decisión de utilizar el modelo MapReduce[6] se basa en la explotación y descubrimiento de información en los archivos log de alertas generados por la capa de monitoreo. Una de las principales ventajas es el análisis en paralelo de grandes archivos, ya que cada nodo se encarga de procesar un bloque de datos y posteriormente la unificación de resultados para la ejecución de nuevos análisis. Es importante aclarar que esta tecnología trabaja de manera óptima con archivos de gran tamaño debido a su configuración distribuida, frente a otras tecnologías de procesamiento de datos que utilizan diferentes técnicas para la computación de resultados. Por lo que es de vital importancia realizar el análisis de la cantidad de información que va ser generada y almacenada dentro del sistema.

A pesar de las ventajas que nos da esta tecnología, no se pudo medir en su totalidad el máximo desempeño o performance del cluster de Hadoop, debido a que la cantidad de logs recolectados por día era aproximadamente archivos de 1MB, registrando el estado de 4 a 8 servicios por host. En un ambiente real, en el que se tenga cientos de hosts, se podrían realizar más pruebas como: rendimiento en la ejecución de tareas MapReduce, cantidad de tráfico en la red, velocidad de escritura de datos, replicación de la información, alta disponibilidad y tolerancia a fallos, entre otras.

Finalmente se presentan los criterios de factibilidad para la implementación de la nueva arquitectura propuesta en el proyecto:

* La combinación de arquitecturas tradicionales y distribuidas, permiten reducir el impacto al momento de migrar a nuevas tecnologías, para este proyecto, Nagios ha brindado la facilidad de integración con la plataforma Hadoop, gracias a que es un sistema abierto y sus características tecnológicas han permitido definir un sólido recurso para la generación de datos.
* Los sistemas de monitoreo tradicionales poseen herramientas para la generación de reportes detallados sobre el estado de la infraestructura de un centro de datos, pero esta información podría ser analizada con técnicas de minería de datos para la creación de patrones de comportamiento, existen ya algunos proyectos en los que se utilizan estas técnicas para la predicción de anomalías, las mismas que representan una evolución en el manejo de grandes centros de datos y a la mejora continua de los servicios que brinda una institución o empresa.
* Tanto Nagios como Hadoop, se basan en tecnologías open-source las mismas que entre algunas de sus ventajas, está la posibilidad de contar con una extensa

comunidad que brinda soporte y actualizaciones continuas de sus

herramientas, adaptarse a las necesidades de los usuarios y a la utilización de recursos económicamente convenientes al momento de crear un ambiente de pruebas y experimentos.

## 5.2 CONCLUSIONES

* Los beneficios de adoptar un sistema de monitoreo distribuido para el control de una infraestructura de servidores se dan con la creación de múltiples centros de monitoreo, delegando la ejecución de chequeos de hosts y la recolección de incidentes a equipos especializados para disminuir el tráfico en la red y a mejorar la gestión de infraestructuras de gran tamaño.
* La implementación de un sistema para el análisis y almacenamiento de la información que genera un sistema de monitoreo tradicional, se considera como un potencial aspecto de mejora, ya que se pueden utilizar técnicas de análisis de grandes cantidades de datos para el descubrimiento de información de interés que no se pueda obtener con las herramientas incluidas en los sistemas tradicionales.
* La implementación de la arquitectura propuesta, reduce el impacto tecnológico al momento de migrar a nuevas tecnologías, ya que se puede utilizar los sistemas tradicionales para la recopilación de información histórica y la generación de nuevos datos como fuentes de entrada para el procesamiento y análisis de los mismos.
* La tecnología utilizada en la implementación de la arquitectura, brinda la posibilidad de utilizar recursos de características económicamente accesibles (commodity), con el fin de aprovechar al máximo sus características tecnológicas y a la facilidad de escalar sus capacidades bajo demanda.
* El diseño de esta arquitectura, propone la utilización de soluciones Cloud

Computing para la implementación del sistema de almacenamiento y análisis de incidentes, con la finalidad de aprovechar la capacidad de procesamiento que tienen los proveedores y la distribución de recursos en diferentes ubicaciones geográficas.

## 5.3 TRABAJO FUTURO

En esta sección se presentan las áreas en las que este proyecto se podría implementar como herramienta de análisis del comportamiento de grandes infraestructuras de servidores; así como también el trabajo futuro de los sistemas de monitoreo y cuales son los nuevos desafíos para la implementación de estas tecnologías.

La propuesta de mejora de los sistemas de monitoreo, se enfoca hacia empresas que posean infraestructuras de gran escala, en la que se manejan cientos de hosts y servicios, entre algunos ejemplos se tiene: proveedores de servicios de internet y hosting, implementaciones basadas en Cloud Computing ya sean públicas o privadas, instituciones educativas o laboratorios que requieran el monitoreo de clusters de computadores, entre otros.

Otro punto importante es la futura implementación del prototipo funcional a gran escala, en donde el escenario ideal sería en un ambiente cien por ciento distribuido, aplicado a las tecnologías Cloud Computing, en donde se tenga un repositorio para almacenar información sobre los incidentes que ocurran tanto en la infraestructura de servidores como en el mismo sistema de monitoreo distribuido. Los resultados de esta implementación permitirán estimar los puntos críticos de fallo que puedan afectar a la continuidad de los servicios.

Entre algunas innovaciones tecnológicas de los sistemas de monitoreo tenemos que Nagios es considerado como uno de los sistemas mas populares para el monitoreo de redes, sistemas y servicios, por lo cual su futuro ofrece nuevas prestaciones y tecnologías, gracias a la comunidad de usuarios que dan mantenimiento a esta plataforma. En este proyecto al utilizar el modo distribuido de Nagios, es importante mencionar que existen algunas alternativas que optimizan la forma del monitoreo distribuido de este sistema, una de estas es el proyecto Distributed Nagios Executor (DNX)[29], el mismo que consiste en una extensión modular de Nagios que descarga una parte significativa del trabajo realizado por los nodos slave de Nagios, asegurando así una distribución justa y equitativa de los servidores que interactúan con el nodo master.

Los sistemas de monitoreo tradicionales dan prioridad a la recolección y notificación de incidentes, mas no al análisis profundo de la información que se puede obtener de los componentes de una infraestructura. Es por esto que los sistemas de monitoreo distribuidos como Chukwa, se basan en la recolección de grandes cantidades de información, y con la ayuda de la plataforma Hadoop nos permite gestionar de manera óptima el almacenamiento, análisis y visualización de los datos.

Hadoop al ser un sistema open-source, su evolución es soportada por múltiples desarrolladores, en especial empresas de soluciones analíticas y Business Intelligence(BI), las mismas que están generando nuevos proyectos como el análisis del lenguaje natural y hablado, en los que en su mayor parte implementan la plataforma para el procesamiento de la información.

En la actualidad los proveedores de servicios Cloud Computing están ofreciendo soluciones de Software como Servicio (SaaS) soportados por Hadoop, entre algunos de estos proveedores tenemos: Amazon Web Service con su producto Elastic MapReduce[26], Google Compute Engine[27], Microsoft Windows Azure ofrece el servicio HDInsight[28], entre otras empresas. Es así que el futuro de esta tecnología promete la evolución de los servicios en la nube y la mejora continua de las soluciones de análisis de grandes cantidades de información.

## 5.4 COMENTARIOS FINALES

Para finalizar la memoria de este proyecto, se tienen algunos comentarios finales enfocados hacia una futura implementación del prototipo funcional.

* Como se menciona anteriormente, la factibilidad en la utilización de la plataforma Hadoop, impacta en la cantidad de información que se va almacenar y procesar al momento de configurar el cluster.
* Si se opta por hacer uso de proveedores de Cloud Computing, como Amazon, Microsoft o Google, es importante haber realizado pruebas para medir el tráfico de información, ya que estos servicios tienen costos de acuerdo a la utilización de sus recursos, y es de vital importancia, obtener métricas de utilización para el máximo aprovechamiento de estas tecnologías.
* El modelo de programación MapReduce, nos da potenciales alternativas para mejorar la forma en que se analiza la información recolectada, es así que se puede investigar nuevas técnicas de clasificación de información, y la posibilidad de adaptar algoritmos inteligentes y aprendizaje automático, que sirvan de herramientas para la predicción de anomalías.

# REFERENCIAS

1. Venner, J. (2009). *Pro Hadoop: Build scalable, distributed applications in the cloud*. New York: Apress.
2. Lam, C. (2011). *Hadoop in Action*. Greenwich: Manning.
3. Kocjan, W. (2008). *Learning Nagios 3.0.* Mumbai: Packt Publishing.
4. Turnbull, J. (2006). Pro Nagios 2.0. New York: Apress.
5. Coulouris, G.; Dollimore, J.; Kindberg, T. (2005). *Distributed systems. Concepts and design. Sistemas Distribuidos. Conceptos y Diseño (3ra. ed.)*. Londres: Pearson Education / Addison Wesley, 2001.
6. Dean, J., & Ghemawat, S. (2008). *MapReduce: simplified data processing on large clusters*. Communications of the ACM, 51(1), 107-113.
7. Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010, May). *The hadoop distributed file system*. In Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on (pp. 1-10). IEEE.
8. Shafer, J., Rixner, S., & Cox, A. L. (2010, March). *The Hadoop distributed filesystem: Balancing portability and performance*. In Performance Analysis of Systems & Software (ISPASS), 2010 IEEE International Symposium on

(pp. 122-133). IEEE.

1. *Página principal del proyecto Hadoop*, http://hadoop.apache.org/.
2. Boulon, J., Konwinski, A., Qi, R., Rabkin, A., Yang, E., & Yang, M. (2008, October). *Chukwa, a large-scale monitoring system*. In Proceedings of CCA

(Vol. 8).

1. Rabkin, A., & Katz, R. (2010, November). *Chukwa: A system for reliable large-scale log collection*. In Proceedings of the 24th international conference on Large installation system administration (pp. 1-15). USENIX Association.
2. *Página principal del proyecto Chukwa*, http://incubator.apache.org/chukwa/.
3. Xu, G., Xu, F., & Ma, H. (2012, August). *Deploying and researching Hadoop in virtual machines*. In Automation and Logistics (ICAL), 2012 IEEE International Conference on (pp. 395-399). IEEE.
4. Jia, B., Wlodarczyk, T. W., & Rong, C. (2010, November). *Performance considerations of data acquisition in hadoop system*. In Cloud Computing

Technology and Science (CloudCom), 2010 IEEE Second International Conference on (pp. 545-549). IEEE.

1. Pervilä, M. A. (2007). *Using Nagios to monitor faults in a self-healing environment*. In Seminar on Self-Healing Systems, University of Helsinki.
2. *Página oficial de Nagios*, http://www.nagios.org.
3. *Página oficial de los plugins de Nagios*,

http://www.nagios.org/projects/nagiosplugins.

1. *Nagios Core Documentation*, http://nagios.sourceforge.net/docs/nagioscore-3en.pdf.
2. *Fundación de Software Apache*, http://incubator.apache.org/.
3. *IEEE-Standards Association*, http://standards.ieee.org/findstds/standard/1471-

2000.html

1. Bass, L., Clements, P., Lazman, R. (2003). *Software Architecture in Practice*

*(2nd edition)*, Addison-Wesley.

1. *UML 2 Documentation*, http://www.omg.org/spec/UML/2.1.2/
2. *Architecture Tradeoff Analysis Method (ATAM)*,

http://www.sei.cmu.edu/architecture/tools/evaluate/atam.cfm

1. *Cloudera*, http://www.cloudera.com
2. *Writing an Hadoop MapReduce Program in Python*, http://www.michaelnoll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/
3. *Amazon Elastic MapReduce*, http://aws.amazon.com/es/elasticmapreduce/
4. *Google Compute Engine*, https://cloud.google.com/products/compute-engine
5. *Microsoft HDInsight*, http://www.windowsazure.com/es-es/services/hdinsight/
6. *Distributed Nagios Executor*, http://dnx.sourceforge.net/
7. Sahoo, R. K., Squillante, M. S., Sivasubramaniam, A., & Zhang, Y. (2004, June). Failure data analysis of a large-scale heterogeneous server environment.

In Dependable Systems and Networks, 2004 International Conference on (pp.

772-781). IEEE.

# ANEXOS

**ANEXO 1: ARCHIVOS DE CONFIGURACIÓN DEL PROTOTIPO**

**FUNCIONAL**

**Configuración del servicio NRPE en el Nodo Cliente:** */etc/nagios/nrpe.cfg*

# Configuracion de los plugins y chequeos

command[check\_ssh]=/usr/lib/nagios/plugins/check\_ssh 127.0.0.1 command[check\_users]=/usr/lib/nagios/plugins/check\_users -w 5 -c

10

command[check\_load]=/usr/lib/nagios/plugins/check\_load -w 15,10,5

-c 30,25,20 command[check\_disk]=/usr/lib/nagios/plugins/check\_disk -w 20% -c

10% -p /dev/mapper/vg\_cliente-lv\_root command[check\_zombie\_procs]=/usr/lib/nagios/plugins/check\_procs -w

5 -c 10 -s Z command[check\_total\_procs]=/usr/lib/nagios/plugins/check\_procs -w

150 -c 200 command[check\_httpd]=/usr/lib/nagios/plugins/check\_http -H localhost -u /

**Configuración de Nagios en el Nodo Slave:** */etc/nagios/nagios.cfg*

# Cambios realizados en el archivo de configuracion de Nagios log\_file=/var/log/nagios/nagios.log

cfg\_file=/etc/nagios/objects/clientes.cfg cfg\_file=/etc/nagios/objects/localhost.cfg

enable\_notifications=0 obsess\_over\_services=1 ocsp\_command=submit\_check\_result

**ANEXO 2: SCRIPTS**

**Nombre del script:** *submit\_check\_result.sh*

**Descripción:** Shell script que luego de realizar un chequeo activo en los host configurados, el resultado se envía a través del complemento NSCA al nodo master y al mismo tiempo se escribe en el archivo log que se encuentra especificado en las variables de configuración.

#!/bin/sh

# Script: submit\_check\_result.sh

#

# Descripcion: Luego de realizar un chequeo activo, el resultado se envia a traves del complemento NSCA al Servidor Nagios Master.

#

# Argumentos:

#

# $1 = host\_name (Nombre corto del host que el servicio tiene asociado)

#

# $2 = svc\_description (Descripcion del servicio)

#

# $3 = state\_string (Una cadena de texto que representa el estado del servicio - "OK", "WARNING", "CRITICAL" o "UNKNOWN")

#

# $4 = plugin\_output (Una cadena de texto que devuelve el resultado del chequeo realizado al servicio)

#

# $5 = hostgroup\_name (Nombre corto del grupo de hosts que contiene al host monitoreado)

# Nagios Master

#NAGIOS\_MASTER = 192.168.1.33

# NSCA variables

#NSCA\_CFG=/etc/nagios/send\_nsca.cfg

#NSCA\_CMD=/usr/sbin/send\_nsca

#Configuracion de log de alertas

LOG\_DIR=/opt/nagios/logs/

LOG\_FILE=nagios\_collector.log

# Verify if LOG\_DIR exists

if [ ! -d $LOG\_DIR ] then

mkdir -p $LOG\_DIR cd $LOG\_DIR else cd $LOG\_DIR fi

# Verify if LOG\_FILE is created

if [ ! -f $LOG\_FILE ] then touch $LOG\_FILE fi

# Convertir la variable state\_string a una variable numérica

RETURN\_CODE=-1

case "$3" in

OK)

RETURN\_CODE=0

;;

WARNING)

RETURN\_CODE=1

;;

CRITICAL)

RETURN\_CODE=2

;;

UNKNOWN)

RETURN\_CODE=-1

;; esac

# Definir la fecha en formato YYYY-MM-DD HH:MM:SS

FECHA=`date +%F%t%T`

#Escribir el resultado del chequeo en el archivo de logs para posteriormete enviarlo al sistema HDFS

/usr/bin/printf "%s\t%s\t%s\t%s\t%s\t%s\n" "$FECHA" "$1" "$2" "$3"

"$4" "$5" >> $LOG\_DIR$LOG\_FILE

# Enviar la alerta pasiva hacia el servidor Nagios Master a traves del programa send\_ncsa

# Se especifica la direccion del servidor y el archivo de configuración

/usr/bin/printf "%s\t%s\t%s\t%s\n" "$1" "$2" "$RETURN\_CODE" "$4" |

/usr/sbin/send\_nsca -H 192.168.1.33 -c /etc/nagios/send\_nsca.cfg

**Nombre del script:** *dispatch\_alert\_log.sh*

**Descripción:** Shell script que ejecuta el proceso para el envío de los archivos log de alertas hacia el nodo master.

#!/bin/sh

#

# Script: dispatch\_alert\_log.sh

#

# Descripcion: Proceso para el envío de logs hacia el sistema de almacenamiento y análisis

#

# Creado: 12-nov-2013

#

# Obtener el nombre del host del colector

HOSTNAME=`hostname`

#Variables de configuracion

PRINTF\_CMD=/usr/bin/printf

#Variables de Hadoop

HADOOP\_BIN=/usr/bin/hadoop

HDFS\_LOG\_DIR=/alerts/sentLogs/

# Configuracion de logs

LOG\_DIR=/opt/nagios/logs/

LOG\_FILENAME=nagios\_collector.log

cd $LOG\_DIR

LOG\_FILE=`ls -1 $LOG\_FILENAME 2> /dev/null`

#Verificar la existencia del archivo

if [ -z $LOG\_FILE ]

then echo "No existe archivo log... Finalizando Script"

else echo "Archivo log encontrado..."

DATE=$(date +"%m-%d-%Y-%H.%M.%S")

SEND\_FILE="$HOSTNAME-$DATE.log.send"

echo "Renombrando $LOG\_FILE a $SEND\_FILE..."

# Renombrar el archivo al formato SEND\_FILE

mv $LOG\_FILE $SEND\_FILE && /etc/init.d/nagios reload

# Enviar archivo al sistema HDFS

echo "Enviando $LOG\_DIR$SEND\_FILE al sistema HDFS..."

/usr/bin/hadoop fs -put $LOG\_DIR$SEND\_FILE $HDFS\_LOG\_DIR

# Comprobar que el archivo se copio correctamente

HDFS\_EXIT\_CODE=$?

if [ $HDFS\_EXIT\_CODE -eq 0 ]; then

echo "Archivo $SEND\_FILE guardado correctamente en el sistema HDFS... Eliminando archivo..."

rm $LOG\_DIR$SEND\_FILE

else

echo "Hubo un error durante la transferencia del archivo..."

fi

fi

**Nombre del script:** *run\_map\_reduce.sh*

**Descripción:** Shell script utiliza las librerías de Hadoop para la ejecución de la tarea

MapReduce para la contabilización de incidentes.

#!/bin/sh

#

# Script: run\_map\_reduce.sh

#

# Descripcion: Proceso para la ejecucion de la tarea Map Reduce para la contabilizacion de incidentes

#

# Creado: 12-nov-2013

#

#Variables de configuracion

SOURCE\_CODE=/home/hadoop/tesis/

#Variables de Hadoop

HADOOP\_LIB=/usr/lib/hadoop-0.20/

HDFS\_LOG\_DIR=/alerts/sentLogs/

HDFS\_PROCESS\_DIR=/alerts/processingLogs/

HDFS\_ARCHIVE\_DIR=/alerts/archiveLogs/

#Ir al directorio de hadoop

cd $HADOOP\_LIB

#Mover archivos al directorio de procesamiento

/usr/bin/hadoop fs -mv /alerts/sentLogs/\* /alerts/processingLogs/

#Ejecutar tarea Map Reduce

echo "Iniciando tarea MapReduce..."

/usr/bin/hadoop jar contrib/streaming/hadoop-\*streaming\*.jar -

mapper /home/hadoop/tesis/mapper.py -reducer

/home/hadoop/tesis/reducer.py -input /alerts/processingLogs/\* output /alerts/tmp/output-job/

echo "Tarea MapReduce finalizada... Moviendo archivos procesados..."

#Mover archivos procesados

/usr/bin/hadoop fs -mv /alerts/processingLogs/\*

/alerts/archiveLogs/

#Archivar los resultados con un formato especifico

echo "Archivando resultados..."

DATE=$(date +"%m-%d-%Y-%H.%M.%S")

/usr/bin/hadoop fs -mv /alerts/tmp/output-job/part-00000

/alerts/finalLogs/job-$DATE

#Eliminar el directorio temporal

echo "Eliminando directorio temporal..."

/usr/bin/hadoop fs -rmr /alerts/tmp/output-job

echo "Fin del script..."

**ANEXO 3: CÓDIGO FUENTE**

**Nombre del programa:** *mapper.py*

**Descripción:** Código fuente del proceso mapper, el cual se encarga de leer los archivos log en el sistema HDFS y devuelve como resultado un conjunto de pares llave/valor expresadas en texto plano, que sirven como entrada para el proceso reducer.

#!/usr/bin/env python

"""Mapper que devuelve los valores de las alertas en los archivos log de nagios"""

import sys

#Funcion para leer el archivo de datos def read\_input(file): for line in file:

# split the line into words yield line.split('\t')

#Funcion principal def main(separator='\t', dots=':'): # La entrada proviene del STDIN data = read\_input(sys.stdin)

#dictionary ={} for words in data:

#El campo consiste en un timestamp(fecha, hora), host, servicio y estado

#El timestamp sirve para

#field = (words[0], words[1], words[2], words[3], words[4])

#dictionary[field]=1

print '%s%s%d' %

(words[0]+dots+words[2]+dots+words[3]+dots+words[4], separator, 1)

#print dictionary if \_\_name\_\_ == "\_\_main\_\_": main()

**Nombre del programa:** *reducer.py*

**Descripción:** Código fuente genérico del proceso reducer[25], toma como entrada el conjunto de datos llave-valor de la función mapper y procede a agruparlos y generar un nuevo conjunto llave-valor.

#!/usr/bin/env python

"""Proceso reducer, toma como entrada un conjunto de datos llavevalor y los ordena"""

from itertools import groupby from operator import itemgetter import sys def read\_mapper\_output(file, separator='\t'): for line in file:

yield line.rstrip().split(separator, 1)

def main(separator='\t'):

# input comes from STDIN (standard input) data = read\_mapper\_output(sys.stdin, separator=separator)

# groupby groups multiple word-count pairs by word,

# and creates an iterator that returns consecutive keys and their group:

# current\_word - string containing a word (the key)

# group - iterator yielding all ["&lt;current\_word&gt;",

"&lt;count&gt;"] items for current\_word, group in groupby(data, itemgetter(0)): try:

total\_count = sum(int(count) for current\_word, count in group) print "%s%s%d" % (current\_word, separator, total\_count) except ValueError:

|  |  |  |
| --- | --- | --- |
| this item |  | # count was not a number, so silently discard |
|  |  | pass |

if \_\_name\_\_ == "\_\_main\_\_": main()

1. El método científico es un método de investigación usado principalmente en la producción de conocimiento en las ciencias. Para ser llamado científico, un método de investigación debe basarse en la empírica y en la medición, sujeto a los principios específicos de las pruebas de razonamiento. [↑](#footnote-ref-1)
2. En el método experimental el investigador interviene sobre el objeto de estudio modificando a este directa o indirectamente para crear las condiciones necesarias que permitan revelar sus características fundamentales y sus relaciones esenciales. [↑](#footnote-ref-2)
3. En Nagios, se define el término ***chequeo***, a la acción que ejecuta un plugin para verificar y devolver el estado actual de un host o servicio. Este término se utiliza en la mayor parte del proyecto. [↑](#footnote-ref-3)
4. Un cluster es un conjunto o agrupación de computadoras de similares características, tanto en hardware como en software y su objetivo es el de compartir recursos para la resolución de tareas complejas. [↑](#footnote-ref-4)
5. La instancia de un programa MapReduce, se refiere a la ejecución del programa dentro de cada nodo o componente que conforma un cluster de computadoras. [↑](#footnote-ref-5)
6. Un buffer de memoria es un espacio de almacenamiento temporal que se utilizan al momento de transferir datos dentro de una aplicación, la principal ventaja es la disponibilidad de recursos tanto para componentes de hardware o software. [↑](#footnote-ref-6)
7. Una llamada a procedimiento remoto es un protocolo que permite a un programa de ordenador ejecutar código en otra máquina remota sin tener que preocuparse por las comunicaciones entre ambos. [↑](#footnote-ref-7)
8. En el ambiente de desarrollo de software, un framework es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de software concretos, que puede servir de base para la organización y desarrollo de software. [↑](#footnote-ref-8)
9. La *metadata*, son los registros que contienen la información sobre los atributos que posee un archivo o directorio a ser almacenado dentro de un sistema de archivos. Esta información tiene que ver con los permisos de accesos, fechas de acceso y modificación, jerarquía y cuotas de espacio en disco. [↑](#footnote-ref-9)
10. Los datos que se almacenan en el sistema de archivos HDFS se guardan en una secuencia de bloques del mismo tamaño y se replican copias de seguridad a los nodos más cercanos con el fin de conseguir que la información sea tolerante a fallos. [↑](#footnote-ref-10)
11. El modo Standalone de Hadoop, es la instalación por defecto de la plataforma, consiste en la configuración de un cluster con un solo nodo, que ejecuta todos los componentes de Hadoop en un solo proceso. [↑](#footnote-ref-11)
12. Los atributos de calidad son requerimientos que especifican los criterios para juzgar la operación de un sistema en lugar de su comportamiento específico para satisfacer las necesidades de los usuarios. [↑](#footnote-ref-12)
13. Stakeholder es un término utilizado en el desarrollo de sistemas para referirse a aquellas personas o entidades que están interesadas en la realización de un proyecto o tarea auspiciando el mismo ya sea mediante su poder de decisión o financiamiento a través de su propio esfuerzo. [↑](#footnote-ref-13)
14. Dentro de los sistemas Unix, **cron** es un administrador regular de procesos en segundo plano (demonio) que ejecuta procesos o scripts a intervalos regulares (por ejemplo, cada minuto, día, semana o mes) [↑](#footnote-ref-14)
15. RAID es un *conjunto redundante de discos independientes*, en donde se utilizan múltiples unidades de almacenamiento entre los cuales se distribuyen o se replican los datos, con la finalidad de obtener mayor integridad, tolerancia a fallos, rendimiento y mayor capacidad en la gestión de la información. [↑](#footnote-ref-15)