Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Centro de Computación Distribuida y Paralela

Expansión de la distribución Linux para Ciencia de Datos Live-Hadoop

Trabajo Especial de Grado presentado ante la ilustre
Universidad Central de Venezuela por el

Br. Pedro Valdivieso
Br. Sebastian Ziegler

Tutor:
Prof. Jesús Lares
Prof. José R. Sosa

Caracas, Agosto del 2015

Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación

ACTA

Quienes suscriben, miembros del jurado designado por el Consejo de la Escuela de Computación para examinar el Trabajo Especial de Grado, presentado por los Bachilleres Sebastian Ziegler Manzo C.I.:20.748.755 y Pedro Luis Valdivieso Gutierrez C.I.: 20.870.806 titulado "Expansión de la distribución Linux para ciencia de datos Live-Hadoop", a los fines de cumplir con el requisito legal para optar al título de Licenciado en Computación, dejan constancia de lo siguiente:

Leído el trabajo por cada uno de los miembros del jurado, se fijó el día 07 de Agosto de 2015, a la 1:00 pm, para que sus autores lo defendieran de forma pública, en el aula PBIII, lo cual estos realizaron mediante una exposición oral de su contenido, y luego respondieron satisfactoriamente a las preguntas que le fueron formuladas por el jurado, todo ello conforme a lo dispuesto en la Ley de Universidades y demás normativas vigentes de la Universidad Central de Venezuela. Finalizada la defensa pública del Trabajo Especial de Grado, el jurado decidió aprobarlo con una nota de 20 puntos.

En fe de lo cual se levanta la presente acta, en Caracas el 07 de Agosto de 2015, dejándose también constancia de que actuó como Coordinador del jurado el Profesor Tutor Jesús Lares.

Prof. Jesús Lares
Tutor

Prof. José R. Sosa
Co-Tutor

Prof. Fernando Crema
Jurado

Prof. Hector Navarro
Jurado

Resumen

Para hacer procesamientos con grandes volúmenes de datos (Big Data) es necesario tener un cluster. Montar un cluster no es una tarea sencilla, existe una distribución de Linux desarrollada en la Universidad Central de Venezuela por el profesor José R. Sosa que monta un cluster Beowulf listo para procesar y almacenar grandes volúmenes de datos. Este sistema operativo lleva por nombre Live Hadoop y presenta ciertas limitaciones que hacen que resuelva un conjunto limitado de problemas de Big Data. Por todo lo anterior se decidió expandir el sistema operativo Live Hadoop para integrar distintas herramientas del ecosistema Hadoop y así ampliar su funcionalidad.

La siguiente investigación tiene como fin crear y adaptar módulos que permitan la extracción, carga, almacenamiento y procesamiento de grandes volúmenes de datos en Live Hadoop. Para lograr lo anterior se integraron distintas herramientas al sistema operativo Live Hadoop en forma de módulos, estos fueron: Logstash, Elasticsearch, Kibana, Mahout, Fluentd, HBase, ZooKeeper y Spark.

Logstash y Fluentd sirven para hacer extracción, carga y limpieza de grandes volúmenes de datos. Elasticsearch permite indexar datos en una base de datos orientada a documentos y lo hace de forma distribuida. Kibana provee una visualización de los datos almacenados en Elasticsearch usando una interfaz web, permite hacer consultas y gráficas. Mahout incluye con una amplia biblioteca de algoritmos de minería de datos que pueden usarse desde distintos lenguajes de programación. Hbase permite indexar datos desde el sistema de archivos distribuido de Hadoop en una base de datos orientada a columnas usando distintos procesos que corren de manera distribuida. Zookeeper tiene toda una interfaz que provee funciones de coordinación y administración para procesos distribuidos. Finalmente Spark permite hacer procesamiento de grandes volúmenes de datos en memoria lo que disminuye el tiempo de procesamiento de manera drástica, tiene una amplia librería de algoritmos de minería de datos y tiene la capacidad escribir resultados en el sistema de archivos distribuidos de Hadoop o en bases de datos.

De todas estas herramientas se hicieron pruebas simples y además se realizó un caso de estudio complejo integrando algunas de las herramientas añadidas. El caso de estudio fue enfocado en el procesamiento de datos de bitácoras de servidores con el fin de detectar anomalías en las mismas y generar reportes y alertas que son enviados a un administrador.

Este trabajo dejo como producto final un sistema operativo con un conjunto de herramientas del ecosistema Hadoop que monta un cluster Beowulf listo para procesar grandes volúmenes de datos usando Live Hadoop. Un sistema de recolección y procesamiento de datos de bitácoras de servidores para detectar anomalías. También, se dejó una instalación del sistema operativo *XenServer* en una maquina de la sala de servidores del centro de computación donde se virtualizó un cluster de 15 nodos de la distribución de Hadoop Hortonworks y otros 5 nodos con la distribución Live Hadoop.

Agradecimientos

A nuestros padres (Alberto y Felix), madres (Linda y Milvia) y hermanos (Daniel) gracias por su apoyo incondicional, por tener tanta paciencia con nosotros, por darnos la oportunidad de poder formarnos en nuestra grandiosa Universidad Central de Venezuela y sobre todo por estar a nuestro lado en todo momento, tanto en las buenas como en las malas.

A nuestros familiares que siempre han estado presente.

A nuestros tutores Jesús Lares y José R. Sosa, gracias por su apoyo durante todo el proyecto y por siempre estar al tanto de este gran trabajo.

A todos nuestros amigos de la universidad (Andrés, Annerys, Amaro, Beto, Carlos, Elohina, Kimberly, Juan, Mary, Vivi) sin todos ustedes esto no hubiese sido posible, 5 años no es poco tiempo y esperamos que sean muchos más los que vengan trayéndonos éxito a todos.

A la Universidad Central de Venezuela, a la Escuela de Computación y a todos y cada uno de los profesores que contribuyeron en nuestra formación académica.

Por parte de Sebastian:

A Pedro V. por haber aceptado cumplir juntos este último reto juntos y que hayamos podido culminar todo con el mayor éxito.

A mis padres y hermano que siempre estuvieron a mi lado en todo momento y que gracias siempre han sido un apoyo incondicional.

A mi tía Silvia M. y mi primo Gianmarco P. gracias por haber colaborado tanto conmigo al momento de tener que fastidiarlos en su casa.

A Jesus B. y Alejandro P. aproximadamente 19 años conociéndonos y en todo momento han estado allí, gracias.

A María S. (Peke) muchísimas gracias por todo el apoyo incondicional, fuiste una gran base para poder sacar adelante todo este trabajo, gracias también por todos esos viernes junto a FJ que ayudaban muchísimo a despejar la mente.

A mi prima Samantha M. siempre estuviste allí y fuiste más que de gran ayuda en las malas, todo salió excelente gracias a ti.

Gracias a todos aquellos que durante esta carrera fueron de gran apoyo y que me ayudaron a crecer mucho más como persona.

Por parte de Pedro:

A Sebastian por ser mi compañero y amigo, se que a veces fui un poco flojo pero siempre estuviste ahí para darme el empujoncito y terminar con éxito esta etapa de nuestras vidas.

A mis padres que siempre estuvieron atentos y me apoyaron en todo. En especial a mi papá que me ayudó cada vez que tenía una duda o inconveniente, sin el aporte de sus conocimiento de Linux muchas de estas cosas no hubiesen sido posibles.

A mi novia Elohina por todo el apoyo que me dio en el desarrollo de este trabajo, gracias por ser mi bastoncito.

A mi abuela Mirian, por estar siempre presente aportando su buen humor y esa actitud positiva.

A mi familia en Mérida: abuela Aura, tía Heidy, Diego Andrés. Ustedes siempre estuvieron en mi corazón y desde allá me mandaban toda su buena vibra y buenos deseos.

A mi tía Chela en Colombia que cuando vino estuvo pendiente del trabajo y desde allá se que recibo tus buenos deseos, gracias por el apoyo.

A mi familia en Miami: tía Maria, Rafael, Nicole, Natalie. Se que no nos vemos mucho pero siempre los recuerdo con mucho cariño y les doy las gracias por estar siempre pendientes de mi trabajo.

A todas las personas que de alguna u otra manera influyeron en el desarrollo de este trabajo, gracias por todo su apoyo y dedicación, comparto este logro con ustedes.

Tabla de Contenidos

Índice de figuras

Índice de tablas

Introducción

En el mundo de Big Data es necesario tener un cluster para poder almacenar o procesar grandes volúmenes de datos. Hacer que un conjunto de computadoras estén unidas entre sí por una red y que se comporten como si fuesen una única computadora no es tarea fácil, requiere de fuertes conocimientos en redes y en el área de sistemas operativos en general.

Actualmente existen un conjunto de sistemas operativos que montan clusters Beowulf basándose en el concepto de Live-CD, es decir, clusters con la característica especial de ser una infraestructura de forma AdHoc o temporal, sobre un conjunto de computadoras que no necesariamente estén pre-configurados para ello y con características heterogéneas. Sobre estas computadoras se corre un sistema operativo totalmente en memoria, se configura el cluster con dicho sistema operativo, se ejecuta lo que se tenga que ejecutar y cuando se apague el computador ninguna de su información en disco se ve alterada. Este concepto es perfecto para laboratorios o salas donde exista tiempo ocioso de los computadores en la noche u otro momento del día. Sin embargo, los sistemas operativos existentes que montan cluster Beowulf basándose en un arranque por Live-CD no están preparados para Big Data.

Motivado por todo lo anterior surgió una iniciativa del profesor José R. Sosa de la Universidad Central de Venezuela de crear un sistema operativo que montara un cluster Beowulf basándose en el concepto de Live-CD y que este preparado para Big Data. Este sistema operativo se llama Live Hadoop, por el proyecto de la Fundación Apache que lleva por nombre Hadoop. Apache Hadoop es un proyecto de software libre, escrito en el lenguaje de programación Java y provee una serie de herramientas para almacenamiento y procesamiento distribuido de grandes volúmenes de datos.

Live Hadoop se encuentra todavía en una etapa de desarrollo y no ofrece ninguna herramienta de un ecosistema Hadoop fuera de los procesos convencionales de MapReduce, lo que hace que esté limitada a resolver ciertos problemas del área de Big Data. Es por ello que el propósito general de esta investigación sera expandir la distribución de Live Hadoop para integrar algunas herramientas del ecosistema Hadoop y así incrementar la gama de problemas que se pueden solventar con Live Hadoop.

La investigación esta dividida en cinco grandes partes. En la primera parte, se plantea el problema de investigación, su justificación, sus objetivos y el alcance de la misma. En la segunda parte, se describen todos los fundamentos teóricos en los cuales se baso la investigación para armar la solución final. En la tercera parte, se explica detalladamente la metodología de trabajo usada que fue una adaptación de la ya existente Programación Extrema. En la cuarta parte, se muestra detalladamente los distintos pasos y procedimientos que se hacen para llegar a la solución. Y finalmente en la quinta parte, donde se concluirá la investigación y además se harán recomendaciones y especificaciones para trabajos futuros sobre Live Hadoop.

Capítulo 1

El Problema

1.1. Planteamiento del Problema

Hoy en día, hay un único estándar de facto para poder procesar grandes volúmenes de datos, un cluster. Específicamente un cluster basado en Apache Hadoop. Sin embargo, desde el punto de vista de investigación, consiste en una tecnología muy costosa porque para realizar grandes cálculos se requiere de grandes infraestructuras.

Para la validación de algoritmos, o simplemente formación, se puede recurrir a instalaciones de prueba o "SandBox" apoyados en máquinas virtuales. El problema de estas soluciones es que no sirven para efectuar tareas analíticas de envergadura.

Se tiene otro tipo de solución al poseer un cluster o un conjunto de nodos. Se puede proceder a instalar alguna de las distribuciones ofrecidas por las 3 grandes compañías especializadas en el procesamiento de grandes volúmenes de datos las cuales son: Cloudera, Hortonworks y MapR. Al tener un cluster se acostumbra a ser utilizado únicamente para un fin en específico, lo que lleva a tener una infraestructura especializada para Hadoop, sin poder dar otro uso a este conjunto de nodos. Cabe destacar que se debe realizar una instalación y configuración previa de cualquiera de las distribuciones para poder iniciar a procesar grandes volúmenes de datos.

Fuera de las soluciones ofrecidas por grandes compañías, se puede instalar todo un ecosistema Apache Hadoop realizando una instalación en limpio. Lo que como usuario final puede llegar a ser un poco engorroso debido a la gran cantidad de configuraciones que se deben realizar.

Existe un sistema operativo iniciado en la Universidad Central de Venezuela por el profesor José R. Sosa donde se creó una distribución de Hadoop, llamada Live Hadoop en el formato LiveCD, que permite levantar una infraestructura de pruebas para grandes volúmenes de datos de forma AdHoc o temporal, sobre un conjunto de equipos en red, que no necesariamente estén pre-configurados para ello. Esta tecnología permite implementar un cluster Hadoop sobre un conjunto de máquinas que pudieran tener otro uso o fin, como por ejemplo una sala de formación, salas de navegación o laboratorios. Para esto, el sistema funciona enteramente en memoria, con lo cual, no requiere de ningún tipo de instalación en disco.

Este sistema nos permite configurar y poner en marcha un laboratorio para el procesamiento de grandes volúmenes de datos con nodos reales, y sobre el cual se pueden hacer cálculos analíticos sin la necesidad de contar con la costosa tecnología que requiere un cluster para computación de alto rendimiento.

Actualmente Live Hadoop brinda un ecosistema de Hadoop básico: Hadoop Common, Hadoop Distributed File System (HDFS), Hadoop Yet Another Resource Negotiator (YARN) y Hadoop MapReduce. Si se quiere realizar otro tipo de tareas que necesiten de otras herramientas del ecosistema Hadoop es necesario instalar esas herramientas y configurarlas antes de poder utilizarlas.

El proceso de instalación y configuración de las herramientas representa un proceso largo y hasta a veces tedioso. Por la característica de ser un LiveCD debe hacerse cada vez que el sistema operativo inicie y hay que tener buenos conocimientos de administrador de sistemas (redes, manejo de la interfaz de comandos, entre otros).

Por todo lo anterior, seria de gran utilidad que Live Hadoop fuese ampliada con distintas herramientas del ecosistema Hadoop, herramientas para la extracción, carga, almacenamiento y procesamiento de grandes volúmenes de datos y así facilitar el trabajo de instalación y configuración de las mismas.

1.2. Justificación

1.2.1. ¿Por qué es un problema?

Es un problema porque limita el uso del sistema operativo Live Hadoop a tareas que solo puedan ser resueltas con un conjunto de herramientas básicas de Hadoop donde solo se pueden aplicar modelos de procesamiento como Open MPI o MapReduce. Incluyendo el hecho que no se ofrecen herramientas que hagan tareas como la extracción y carga de los datos, no se ofrecen bases de datos distribuidas para el almacenamiento de datos o herramientas que permitan coordinar distintos procesos. Lo que lleva a otra de las problemáticas latentes en las instalaciones de un cluster Hadoop es la configuración funcional de las herramientas que complementan al ecosistema, realizar esta tarea es un trabajo arduo que no todos los usuarios finales de Hadoop pueden realizar, las grandes distribuciones solucionan este problema de configuración mediante interfaces amigables pero su instalación requiere de una serie de pasos que para algunos usuarios no sería del todo trivial.

1.2.2. ¿Para quién es un problema?

Es un problema para cualquier profesor o estudiante de la Universidad Central de Venezuela que necesite de funcionalidades adicionales de Live Hadoop para realizar tareas de extracción, carga, almacenamiento y procesamiento de grandes volúmenes de datos. Pero también se ve afectado cualquier usuario del sistema operativo en cuestión que necesite de funcionalidades adicionales para realizar tareas mucho más específicas.

1.2.2.1. Profesores

Es un problema para los profesores de la Universidad Central de Venezuela ya que uno de los objetivos que se tiene planteado es usar el sistema operativo Live Hadoop como herramienta de enseñanza para las cátedras de Ciencias de Datos. Como Live Hadoop carece de ciertas facilidades a la hora de realizar procesamiento de grandes volúmenes de datos los profesores se ven obligados a impartir una materia con mayor contenido teórico dejando de lado la parte practica.

1.2.2.2. Estudiantes

Es un problema para los estudiantes de la Universidad Central de Venezuela que deseen usar este sistema operativo como una herramienta adicional que los ayude a entender el funcionamiento de Hadoop y algunas de las herramientas del ecosistema. Si Live Hadoop no tiene algunas de las funcionalidades deseadas por algún estudiante entonces se verán en la obligación de instalar las herramientas que cumplan sus requerimientos y para la instalación de las mismas es necesario tener buenos conocimientos en el área de redes y de sistemas operativos en general, aparte esta instalación suele ser un proceso largo y tedioso.

1.2.3. ¿Desde cuándo es un problema?

El problema de la instalación de un cluster para procesamiento de grandes volúmenes de datos tiene sus raíces en 1997 [1] cuando se introdujo el concepto de Ciencia de los Datos por Chien-Fu Jeff Wu. Desde 1997 y hasta el día de hoy no se ofrecen soluciones de rápido despliegue y fáciles de instalar que incluyan un conjunto de herramientas lo suficientemente variado como para satisfacer las necesidades de los usuarios.

Más recientemente en el semestre 2-2014 en la escuela de Computación de la Facultad de Ciencias de la Universidad Central de Venezuela se empezó a impartir la cátedra de Introducción a la Ciencia de los Datos y fue necesario un sistema operativo con las características de Live Hadoop, para trabajar en los laboratorios de la materia. La solución más cercana al problema de instalación fácil de un cluster Hadoop, lo resuelve Live Hadoop con un rápido despliegue y fácil instalación, dejando de lado la inclusión de distintas herramientas.

En lineas generales es un problema desde que se desea realizar el procesamiento de grandes volúmenes de datos ya que no se tienen soluciones fáciles de instalar, de rápido despliegue y con un conjunto de herramientas bastante completo como para realizar distintas tareas.

1.3. Objetivos de la investigación

1.3.1. Objetivo General

Ampliación de la distribución Live Hadoop mediante la creación y adaptación de módulos que permitan la extracción, carga, almacenamiento y procesamiento de datos.

1.3.2. Objetivos Específicos

- Definir las herramientas a utilizar.
- Crear los módulos necesarios que permitan la extracción, carga, almacenamiento y procesamiento de datos.
- Integrar los módulos creados a Live Hadoop.
- Tomar como caso de estudio el análisis de las bitácoras de servidores.
- Visualizar los resultados obtenidos.

1.4. Antecedentes

Es frecuente en el campo de la computación paralela y distribuida encontrarse con el problema de crear toda una infraestructura con un conjunto de nodos que formen un cluster para hacer algún tipo de procesamiento o almacenamiento. Hay diversos sistemas operativos similares a Live Hadoop en el sentido de que permiten levantar una infraestructura de pruebas de forma AdHoc o temporal, sobre un conjunto de equipos en red, que no necesariamente estén pre-configurados para ello y con características heterogéneas.

Uno muy popular se llama PelicanHPC, el cual permite montar un cluster de alto rendimiento en pocos minutos. El modelo de procesamiento que se puede hacer sobre un cluster montado con PelicanHPC es OpenMPI [2], si se desea hacer otro tipo de procesamiento se deben realizar las configuraciones pertinentes.

Otro sistema operativo con características similares es MOLA, basado en PelicanHPC. El uso principal de MOLA es para hacer procesamiento paralelo y distribuido de virtual screening. Virtual screening es una técnica computacional usada en el descubrimiento de drogas donde básicamente se busca librerías de pequeñas moléculas para identificar la estructura de aquellas que tienen mas probabilidad de unirse a otro tipo de droga, generalmente un receptor de proteína o una enzima [3].

Entre las distribuciones para el procesamiento de grandes volúmenes de datos están los previamente mencionados de las 3 grandes compañías: Cloudera, Hortonworks y MapR.

PelicanHPC y MOLA al igual que Live Hadoop montan un cluster Beowulf pero hasta el momento de la creación de este trabajo no se tiene conocimiento alguno de distribuciones que permitan el procesamiento de grandes volúmenes de datos sin tener que realizar un arduo trabajo de instalaciones y configuraciones.

1.5. Alcance

El presente trabajo de investigación desarrollara una serie de módulos que permitan la extracción, carga, almacenamiento y procesamiento de grandes volúmenes de datos utilizando el sistema de generación de módulos para el sistema operativo Slax (ver 2.2.5.1).

La realización de los módulos implica instalar las herramientas necesarias al sistema operativo Live Hadoop, bien sea a través de la compilación de las mismas o en el caso que este disponible los archivos binarios.

Enfocado al caso de estudio de las bitácoras de servidores, los módulos integrados deben estar en la capacidad de hacer una extracción y carga de las bitácoras desde la fuente externa al sistema de archivos distribuido del cluster haciendo en el proceso una limpieza de los datos. Para esto se tiene pensado integrar Logstash (ver 2.4.5.3.1) a Live Hadoop.

Para el procesamiento de los datos de las bitácoras de servidores se tiene pensado utilizar el lenguaje de programación R (ver 2.3.1) apoyándose en la indexación de Elasticsearch (ver 2.4.5.2.4) y HDFS (ver 2.4.2) sobre los datos. De igual manera Elasticsearch debe ser integrada al sistema operativo.

Finalmente para la visualización de resultados del análisis de los datos de las bitácoras de servidores se tiene pensado integrar a Live Hadoop la herramienta Kibana (ver 2.4.5.2.5).

Capítulo 2

Marco Conceptual

2.1. Ciencias de datos

Ciencias de datos es la generación de conocimiento a partir de grandes volúmenes de datos, aplicando técnicas de procesamiento paralelo y distribuido, para implementar algoritmos que permitan predecir o detectar patrones sobre los datos almacenados. A partir de los resultados obtenidos se podrán construir herramientas que permitan analizar los resultados y realizar procesos de toma de decisiones.

2.1.1. Grandes volúmenes de datos (Big Data)

Término aplicado a conjuntos de datos que superan la capacidad del software y hardware habitual para ser capturados, gestionados y procesados en un tiempo razonable. También se puede definir como una tendencia en el avance de la tecnología la cual es utilizada para describir enormes cantidades de datos con distintos tipos de estructuras (estructurados, semi-estructurados y no estructurados) que tomaría demasiado tiempo y sería muy costoso cargarlos a una base de datos relacional para su análisis.

Al hablar del término Big Data, se tienen relacionados distintos fenómenos los cuales ayudan a explicar mejor el concepto de Big Data. Estos fenómenos están separados en 5 variables, conocidas como las 5 V de Big Data:

- Volumen: cantidad de data generada por unidad de tiempo. Dependiendo de las capacidades de procesamiento o almacenamiento, grandes volúmenes pueden ser Terabytes, como para otros grandes volúmenes pueden ser Zettabytes. Para el procesamiento de estas cantidades no se deben utilizar bases de datos tradicionales debido a que su rendimiento es deficiente y no proveen técnicas para el particionamiento de estas.
- Velocidad: velocidad a la cual es generada la data. Existen herramientas las cuales permiten el análisis de estos datos sin tener siquiera que almacenarlos.
- Variedad: distintos tipos de dato que se pueden utilizar. Los datos aparte de tener diferentes tipos,

también puede ser estructurados, semi-estructurados y no estructurados (para más información de la Organización de los datos observar el punto 2.1.2).

- Veracidad: credibilidad y correctitud de los datos. Algunas veces, dependiendo de la fuente de los datos, la calidad y certeza de estos no pueden ser controlados y estos casos para el momento de su análisis podrían entregar valores erróneos. Normalmente los datos erróneos son creados gracias al poco control que se puede tener sobre un humano, es decir, la fuente de los datos provienen de los humanos.
- Valor: valor oculto en los datos. Así se tengan grandes volúmenes de datos, si estos no poseen valor alguno, no se podría finalmente obtener información valiosa de estos.

Ciencia de datos incorpora diversos elementos que facilita la aplicación de esta a distintos campos como:

- Matemática
- Computación
- Estadística
- Ingeniería de datos
- Modelos probabilísticos
- Medicina

2.1.2. Organización de los datos

La organización de los datos se refiere a cómo organizar los datos usando un sistema manejador de bases de datos o cualquier otra tecnología para la administración de datos.

2.1.2.1. Estructurados

Organizan y estandarizan como los elementos de datos se relacionan unos con otros, esto se hace siguiendo un modelo de datos específico que implica una serie de reglas que deben ser aplicadas a los datos. Por ejemplo, las bases de datos relacionales utilizan este modelo para organizar los datos.

2.1.2.2. Semi-Estructurados

Son una forma de datos estructurados que no aplican a ningún modelo formal de estructura (generalmente asociados a las bases de datos relacionales), estos datos suelen contener etiquetas o algunas otras marcas para identificar y separar los elementos semánticos fortaleciendo una jerarquía dentro de los mismos datos.

Este concepto de datos semi-estructurados se genera de los populares lenguajes de marcado, como por ejemplo, eXtensible Markup Language (XML), muy usado en páginas web y en tecnologías orientadas a servicios web, o también JavaScript Object Notation (JSON), usado comúnmente por su facilidad de procesamiento y basado en el lenguaje de programación Javascript.

2.1.2.3. No Estructurados

Aquellos datos los cuales no siguen un modelo de datos predefinido o que no están organizados de una manera bien definida. También pueden ocurrir escenarios como que la data es estructurada aunque no está formalmente definida en un modelo de datos. Los datos no estructurados suelen ser pesados en texto, aunque pueden contener información como fechas, números y hechos. Esto resulta en ambigüedades que hacen que sea mucho más difícil el procesamiento utilizando algoritmos tradicionales, lo cual lleva también a utilizar mecanismos de almacenamiento más complejos.

2.1.3. Bases de datos

Un *sistema de base de datos* (DBS) es un registro computarizado con la finalidad de mantener información y tenerla disponible siempre que se requiera. Las bases de datos normalmente almacenan datos en computadores. Un *sistema manejador de bases de datos* (DMBS) es un conjunto de programas que permiten la administración de una base de datos [4].

2.1.3.1. Bases de datos relacionales

Una *base de datos relacional* es una colección de datos organizados en un grupo de tablas formalmente descritas a través de un esquema (Schema), estas tablas pueden ser accedidas o re-ensambladas. La manera estándar para acceder a una base de datos y que suele servir de interfaz para los usuarios de estas es el Structured Query Language (SQL). Las sentencias SQL suelen usarse para obtener información de las bases de datos, agregar información, borrarla o modificarla.

En terminología de bases de datos relacionales, cuando se refieren a las tablas, las llaman relaciones; a las columnas, atributos y a las filas, tuplas.

Una *relación* es un grupo de tuplas que contienen los mismos atributos. Una tupla suele representar un objeto y la información acerca de ese objeto. En las bases de datos relacionales, cada tupla tiene una clave primaria que es simple si la representa un atributo o es compuesta si mas de un atributo la representa. Además cada tupla puede tener una clave foránea que hace referencia a otra relación, donde en dicha relación la clave foránea se convierte en una clave primaria. Un ejemplo de algunas bases de datos relacionales: MySQL, Oracle, PostgreSQL o MariaDB.

2.1.3.2. Bases de datos no relacionales

Una *base de datos no relacional* (también llamadas popularmente No-SQL) provee un mecanismo de almacenamiento y obtención de datos que es modelado de forma distinta al de las bases de datos relacionales. Las motivaciones por las cuales fueron creadas estas bases de datos son por simplicidad del diseño, escalabilidad horizontal (agregar más nodos a un sistema distribuido aumenta la capacidad de almacenamiento y computo) y un fino control sobre la disponibilidad. Estas bases de datos no relacionales suelen clasificarse en varios tipos: orientadas a grafos, orientadas a documentos, pares nombre-valor, orientadas a columnas y las que combinan distintos tipos.

Ejemplos de algunas bases de datos no relacionales agrupadas por su clasificación:

- Orientada a columnas: Accumulo, Cassandra, Druid, HBase y Vertica.
- Orientada a documentos: Clusterpoint, CouchDB, Couchbase, MarkLogic, MongoDB y OrientDB.
- Orientada a pares nombre-valor: CouchDB, Dynamo, FoundationDB, MemcacheDB, Redis, Riak, Aerospike, OrientDB y MUMPS.
- Orientada a grafos: Allegro, Neo4J, InfiniteGraph, OrientDB, Virtuoso y Stardog.

2.2. Sistema Operativo

Existen distintos tipos de sistemas operativos y estos pueden tener distintas definiciones dependiendo de su uso. Un *Sistema Operativo* puede ser un programa el cual administra el hardware de un computador. También provee una base para poder ejecutar programas sobre este hardware, actuando como intermediario entre el hardware y los programas a ejecutar [5].

También un sistema operativo puede proporcionar a los programadores de aplicaciones un conjunto abstracto de recursos simples, en vez de los complejos conjuntos de hardware [6]. Es decir, nos ayuda a observar el hardware con el cual vamos a interactuar de una forma mucho más sencilla, ya que se realizan abstracciones para el acceso a este.

2.2.1. Tipos de Sistema Operativo

En esta sección se procederá a describir los distintos tipos de sistema operativo existentes, los cuales pueden ir desde aquellos que tienen una utilidad muy específica (Léase el punto 2.2.1.4) o aquellos que tienen múltiples propósitos (Léase el punto 2.2.1.2).

2.2.1.1. Tiempo Real

Son aquellos sistemas los cuales se caracterizan por tener el tiempo como un parámetro clave. Que un sistema sea de tiempo real no quiere decir que sea un sistema rápido. Se podría hablar de *sistema*

operativo de tiempo real como aquel que tiene una precisión alta al momento de ejecutar alguna acción debido a algún evento que haya ocurrido. Los sistemas de tiempo real tienden a reaccionar por eventos pero también existen sistemas de tiempo real de tiempo compartido [6]. En los sistemas de tiempo real se tienen 2 clases:

- Sistemas de tiempo real estrictos: son aquellos que tienen que cumplir alguna acción determinada siempre que se le solicite y dicha acción debe de tener un inicio y un fin bien marcado, este inicio y fin es llamado determinismo. Un ejemplo clásico de estos sistemas son los frenos ABS de los automóviles, estos deben responder siempre que se les sea solicitado y deben culminar de forma exitosa su ejecución, de lo contrario podría suceder algún accidente.
- Sistemas de tiempo real no estrictos: tienen la capacidad de proveer al usuario tiempo real (así como es el tiempo real estricto), pero permite que ciertas acciones a realizar puedan fallar. Estas fallas no deben de existir en todo momento, ya que en muchos casos se debería de tener una respuesta adecuada. Por ejemplo: los smartphones o los sistemas de audio digital.

2.2.1.2. Multiprocesador

Durante mucho tiempo se tuvo procesadores (CPU) los cuales mantenían un único núcleo de procesamiento, un tiempo después, alrededor del año 2000 se comenzaron a observar en el mercado los primeros procesadores multinúcleo. Estos procesadores necesitan de un sistema operativo el cual provea el soporte necesario para poder utilizar los distintos núcleos dentro de un mismo procesador, como también provea soporte para utilizar distintos procesadores conectados a un mismo hardware. Estos sistemas son llamados *multiprocesador*, también son conocidos como sistemas multinúcleo o sistemas paralelos. Son ampliamente utilizados hoy en día. Estos sistemas aparecieron primero en servidores y luego tendieron a migrar estos sistemas a los equipos de escritorio [5]. Los sistemas más utilizados hoy en día, como Windows, Linux y Mac OSx, proveen soporte desde hace algún tiempo atrás de múltiples procesadores. Hasta los sistemas operativos móviles tendieron a migrar a multiprocesamiento debido a que en el mercado existen soluciones multinúcleos para estos dispositivos.

2.2.1.3. Distribuido

También conocidos como sistemas operativos de cluster, los *sistemas operativos distribuidos* se soportan bajo el concepto de sistemas multiprocesadores con la diferencia que estos sistemas tienden a soportar múltiples CPU enlazados mediante conexiones de red, los cuales son llamados nodos. Estos sistemas tienden a compartir el almacenamiento y su poder de procesamiento para un único sistema. El sistema operativo distribuido observará una gran cantidad de hardware como recursos los cuales serán puestos a disposición del usuario para ser utilizados. La ventaja de estos sistemas es que permiten proveer al usuario una gran potencia mediante el uso de distintos nodos [5].

En un cluster se busca proveer alta disponibilidad de servicios, esto se realiza gracias a la redundancia de hardware que se encuentra entre nodos. Actualmente un cluster puede ser heterogéneo, lo cual significa que puede contener distintos tipos de nodos con distintas especificaciones, no tienen que ser necesariamente iguales. Uno de los frameworks más populares (Más información en el punto 2.4) hoy

en día para el procesamiento de grandes volúmenes de datos, trabaja sobre un sistema operativo el cual funciona de forma distribuida.

2.2.1.4. Embebido

Son sistemas los cuales suelen operar sobre el hardware el cual tiene funciones muy específicas. Estos sistemas tienden a ofrecer solo un conjunto de operaciones, no suelen ocupar mucho espacio en memoria debido a lo específicos que son. Normalmente no permiten la ejecución de aplicaciones de terceros sobre estos sistemas debido a que son escritos en memorias de tipo ROM [6].

2.2.2. Componentes de un Sistema Operativo

Los sistemas operativos actuales se basan en módulos. Estos módulos tienen funciones determinadas, de no estar construidos de esta manera, el sistema operativo sería un único programa el cual manejaría todos los recursos disponibles y no se podría apoyar en otros módulos para realizar distintas funciones lo cual lleva también a un mantenimiento mucho más engorroso.

2.2.2.1. Kernel

El *Kernel* de un sistema operativo es un programa el cual maneja las solicitudes de entrada y salida provenientes de los programas que se ejecutan sobre el sistema operativo, traduce estas solicitudes a instrucciones para ser procesadas por el CPU y cualquier otra pieza de hardware que lo requiera. Es el elemento fundamental de un sistema operativo [7] ya que posee todas las funciones principales de este.

2.2.2.1.1. Programas en ejecución Un sistema operativo para poder ejecutar un programa que se encuentre en disco, primero debe de ser cargado en memoria. Estos programas en ejecución son llamados *procesos*. Específicamente un proceso es una instancia de un programa en ejecución, incluyendo los valores actuales del contador de programa, registros y variables [6].

2.2.2.1.2. Módulos Los sistemas operativos actuales tratan de ser muy sencillos en su núcleo y también tratan de proveer soporte de nuevas funciones, estas funciones utilizan las interfaces que provee el Kernel para poder comunicarse con el hardware.

Como se puede observar en la figura 2.1, la arquitectura de Unix/Linux es un ejemplo perfecto para hablar de módulos. Ya que el Kernel de Linux provee funciones básicas como las que ya se ha hablado más atrás. Luego las distintas aplicaciones se conectan por las interfaces provistas por el Kernel para hacer uso del hardware. Estas aplicaciones proveen funcionalidades extra al sistema operativo y hacen transparente el uso del hardware.

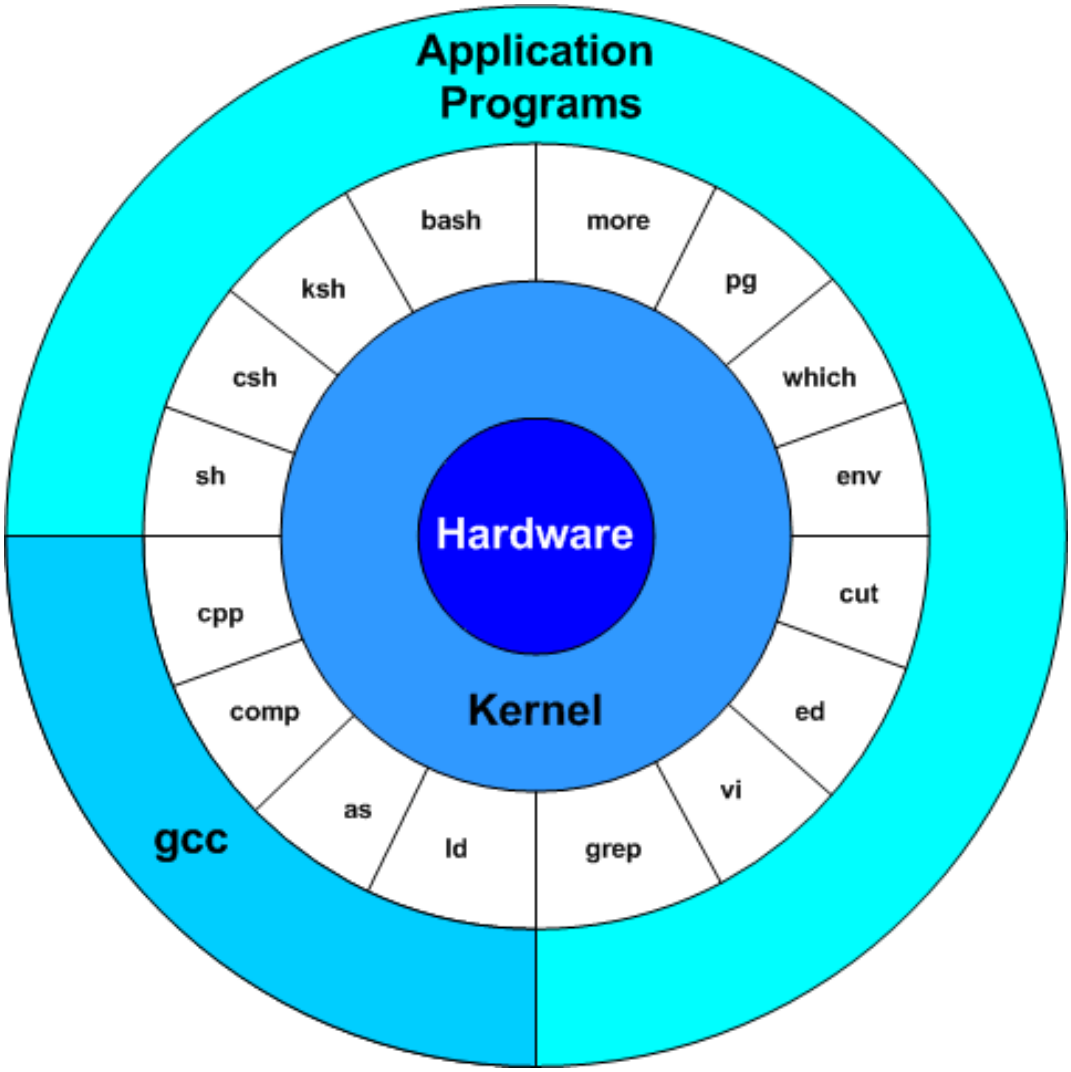


Figura 2.1: Arquitectura de Unix/Linux

2.2.2.1.3. Multi-Tarea Es la capacidad de poder manejar múltiples procesos en muy poco tiempo o de forma simultánea, esto según el soporte del sistema operativo a los procesadores multinúcleo, para así poder demostrar al usuario final que se están corriendo distintos procesos en un mismo momento [6][7]. Los sistemas operativos contienen distintas políticas de ejecución de procesos ya que todos estos deben compartir la CPU en un momento determinado. El proceso que posea el control de la CPU en cierto momento tiene la capacidad de acceder a memoria y también de ejecutar instrucciones. Este control se mantiene por un tiempo determinado por el planificador del sistema operativo el cual procederá a devolverle eventualmente el control al Kernel y finalmente poder proveer a otro programa la capacidad de ejecutarse. Este cambio entre procesos y programas es llamado *cambio de contexto*.

2.2.2.1.4. Manejo de memoria Como se habló anteriormente, un sistema operativo administra los recursos a ser utilizados. Uno de estos recursos y uno de los más valiosos es la memoria RAM (Random Access Memory). Los sistemas operativos deben encargarse del manejo de memoria que no es más que la asignación de espacios de memoria a procesos para que puedan ser ejecutados de forma correcta. Es-

tos espacios de memoria son asignados gracias a que la memoria está dividida en segmentos los cuales ayudan a distribuir el espacio de memoria total entre distintos usuarios y así poder hacer un mejor uso de esta. Actualmente los sistemas operativos modernos proveen soporte de manejo de memoria y protección, para así poder asegurar a los distintos procesos que los espacios de memoria serán únicos y no serán accedidos por otros programas. Ya que en un sistema operativo se le provee una abstracción a los programas haciéndole creer a este que tiene el control total de los recursos.

2.2.2.2. Comunicación enfocada a sistemas distribuidos

Un *sistema distribuido* se define como aquellos en los cuales se tienen múltiples piezas de hardware redundante en distintos computadores (nodos) y estos utilizan las redes de interconexión para comunicarse entre sí y así aprovechar la capacidad de procesamiento de todos estos computadores. Para poder hacer uso de esta capacidad de procesamiento se tiene que disponer de un sistema operativo el cual permita la comunicación entre dichos nodos para poder separar un gran proceso en varios pequeños para ser resueltos por distintos nodos. Como también se tienen que resolver distintos problemas como por ejemplo: el uso de un mismo dato en distintos nodos, uso de datos actualizados en un nodo, sincronización entre procesos, entre otros.

La comunicación entre los distintos nodos de un sistema distribuido se puede realizar mediante redes LAN (Redes de área local), MAN (Redes de área metropolitana) o WAN (Redes de área global), el tipo de alcance viene dado por la necesidad de comunicar un cluster que se encuentra en un espacio físico con otro. Estas comunicaciones suelen hacerse, de ser en una LAN, con cables Ethernet debido a su velocidad y bajo costo. Como también podrían utilizarse cables de fibra óptica los cuales son utilizados a nivel de redes MAN o WAN.

Algunos de los problemas que suelen surgir al momento de tener un sistema distribuido son:

2.2.2.2.1. Transparencia Proveer transparencia al momento de acceder a los datos, indicar su localización para evitar problemas al momento de necesitar datos en un nodo y que puedan estar almacenados en otro nodo. Existen casos donde la transparencia en la replicación de archivos es clave para así poder solventar el caso explicado anteriormente. Los nodos deberían de poder completar sus tareas sin problema alguno.

2.2.2.2.2. Comunicaciones Los problemas de comunicaciones están atados a 2 factores clave, la infraestructura utilizada, topología y fallos en la red; y la comunicación entre nodos la cual debe facilitar primitivas para la comunicación entre estos.

2.2.2.2.3. Escalabilidad y rendimiento El rendimiento de un sistema distribuido viene acotado por el rendimiento de cada nodo por separado, la velocidad de comunicación de la red y la tolerancia a fallos soportada por cada nodo en particular. La escalabilidad viene acotada por el costo de añadir nuevos recursos y el rendimiento total al momento de añadir más nodos (este factor es dependiente de la velocidad de la red).

2.2.2.2.4. Heterogeneidad Existen problemas relacionados a tener hardware de distintos fabricantes o hasta distintos modelos. Este problema se extiende también al software, a los dispositivos de comunicación y protocolos de comunicación. Normalmente se trata de tener un middleware el cual debe proveer un manejo bastante amplio de hardware el cual su funcionalidad final es hacer transparente para el software el uso de este.

2.2.2.2.5. Tolerancia a fallos y confiabilidad La confiabilidad de un sistema viene dada por su capacidad para resolver errores, su disponibilidad y su minimización de puntos críticos. En el caso de la tolerancia a fallos se acostumbra a tener 2 casos básicos, se enmascaran los errores y se intentan resolver o se muestra un fallo pero de manera tal que no se pierda información alguna.

2.2.2.2.6. Seguridad Existen 3 puntos clave en el apartado de seguridad, se debe tener protección ante personal no autorizado (confidencialidad), se debe tener protección ante alteración y corrupción de datos (integridad) y finalmente se deben mantener los recursos accesibles (disponibilidad).

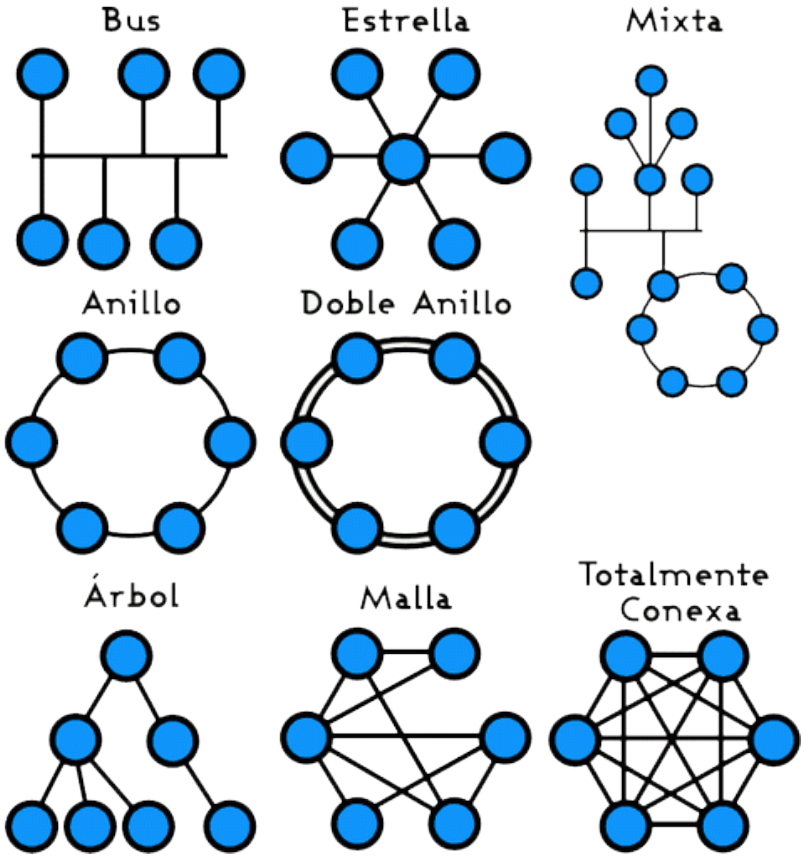


Figura 2.2: Topologías de red

2.2.2.2.7. Topologías de red Como se puede observar en la figura 2.2, existen distintos tipos de topología de red las cuales pueden proveer una comunicación confiable y permiten realizar combinaciones para entregar un alto desempeño. Entre las conexiones más utilizadas en sistemas distribuidos tenemos:

- Malla: se interconectan distintos nodos pero existen casos en los cuales no existen caminos directos entre un par de nodos y para poder comunicarse toman un camino utilizando nodos intermedios.
- Árbol: pueden tener un buen desempeño en casos donde los nodos a comunicar estén cercanos, pero de estar lejos, puede que exista una ralentización.
- Estrella: es una de las topologías más utilizadas, permite la conexión de distintos dispositivos y se tiene una cantidad de saltos entre un par de nodos muy pequeña. El punto negativo en esta topología es que depende de un nodo central.
- Mixta: permite la conexión de distintos dispositivos y es una combinación de cualquier tipo de red, se suelen combinar los nodos centrales de las topologías estrella y hacer topologías de malla o árbol con estos nodos centrales.

2.2.3. Ejecución de un Sistema Operativo

Los sistemas operativos pueden ser cargados de 2 formas distintas las cuales se explicarán a continuación.

2.2.3.1. Instalaciones

El sistema se instala en uno de los dispositivos de almacenamiento secundario, luego al encender el computador se carga la BIOS, la cual viene integrada en la tarjeta madre del equipo, luego en la BIOS se procede a ejecutar POST (Power-On Self Test) para chequear que todos los dispositivos conectados al computador funcionen correctamente. Se lee la configuración de la BIOS en la cual se tiene un orden de arranque de dispositivos de almacenamiento. Luego en caso de utilizar MBR (Master Boot Record) en el dispositivo de entrada/salida, se procede a leer un sector específico del dispositivo en cuestión y se carga el bootloader primario el cual nos llevará a un bootloader secundario el cual permitirá seleccionar el sistema operativo a cargar.

2.2.3.2. Live-CD

Existe también una forma de correr un sistema operativo sin tener que realizar una instalación, al iniciar el computador, se puede tener desde un dispositivo de entrada/salida (USB, unidad de CD/DVD) una imagen de un sistema operativo la cual se puede cargar al momento de seleccionar el dispositivo a cargar en el bootloader, este sistema operativo no tiene la necesidad de ser instalado, para poder ejecutarlo sencillamente al momento de cargar el 2do bootloader, se carga desde el dispositivo de entrada/salida a utilizar y este es leído y copiado en memoria. Lo cual hace que su ejecución sea muy veloz (ya que la ejecución es directamente desde la memoria). Otra de sus ventajas es que puede ser utilizado sin problema alguno y luego, una vez apagado el computador, el sistema operativo será eliminado de la RAM y de estar instalado algún otro sistema operativo en disco, este puede iniciar sin problema alguno.

2.2.4. Correo Electrónico

Existen diversas formas de comunicación y aunque las redes sociales y los mensajes SMS han empujado al correo electrónico a ser una tecnología vieja aun sigue siendo el estándar para la comunicación en linea a través de la red de redes conocida como Internet.

La infraestructura que hay por debajo de un sistema de correo y que hace el envió de correos electrónicos posible es compleja. Un sistema de correo consiste de distintos componentes (ver figura 2.3 [8]):

- Un “mail user agent” (MUA) que permite a los usuarios escribir el correo.
- Un “mail submission agent” (MSA) acepta el correo que sale de un MUA, lo transforma, y lo manda al sistema de transporte.
- Un “mail transport agent” (MTA) que encamina los mensajes entre las maquinas.
- Un “delivery agent” (DA) que coloca el mensaje en un repositorio de almacenamiento (por ejemplo una base de datos).
- Opcionalmente un “access agent” (AA) que conecta al MUA con el repositorio de almacenamiento de correos (a través de el protocolo IMAP o POP).

En conjunto con estos sistemas también hay herramientas para reconocer spam, virus informáticos, entre otros. El corazón de un sistema de correo electrónico es el MTA, sendmail es el MTA original de los sistemas operativos UNIX, escrito por Eric Allman en la Universidad de Berkeley. Desde entonces muchos MTAs han sido desarrollados, algunos de ellos son productos comenrciales y algunos son implementaciones de software libre. Los mas populares aparte de sendmail son Exim y Postfix (mencionado mas adelante 2.2.4.1) [8].

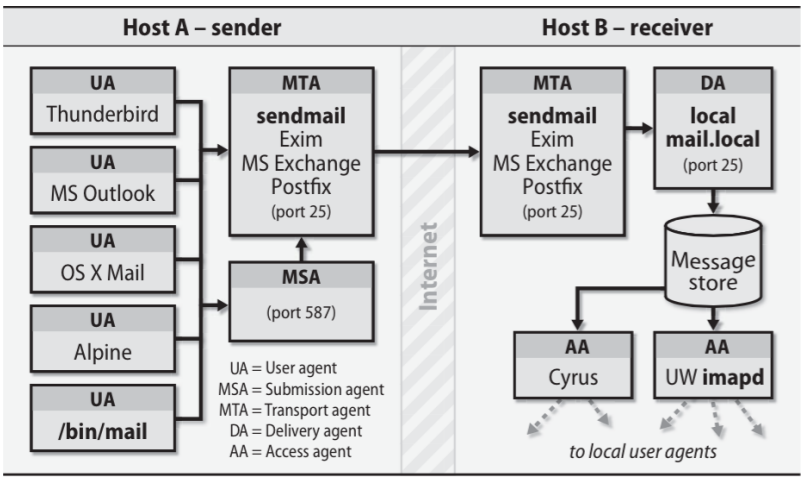


Figura 2.3: Componentes de un sistema de correo electrónico

2.2.4.1. Postfix

Postfix es una muy popular alternativa a sendmail. Escrito por Wietse Venema de IBM en el año 1996, los objetivos de diseño de Postfix se enfocaron mayormente en la seguridad pero también en una política de distribución de software libre, buen rendimiento, robustez y flexibilidad. Todas las distribuciones grandes de Linux incluyen Postfix [8].

Una de las cosas mas importantes de Postfix y la razón por la que es tan popular es porque funciona con muy pocas configuraciones y además implementa de manera eficiente expresiones regulares para filtrar el correo electrónico.

Postfix habla ESMTP. Soporta dominios virtuales en las direcciones de correo y filtros de spam.

Postfix esta compuesto de varios programas pequeños que envian mensajes de red, reciben mensajes, entregan el correo local, etc. La comunicación entre ellos se hace a través de sockets de dominio local (tambien conocidos como FIFOs). Postfix trabaja bajo el paradigma maestro-esclavo, donde un programa maestro se encarga de arrancar y administrar todos los demás procesos. El programa maestro tiene un archivo de configuración único llamado “master.cf” donde se listan configuraciones generales y especificas acerca de como deben ser arrancados el resto de los programas [8].

2.2.5. Linux

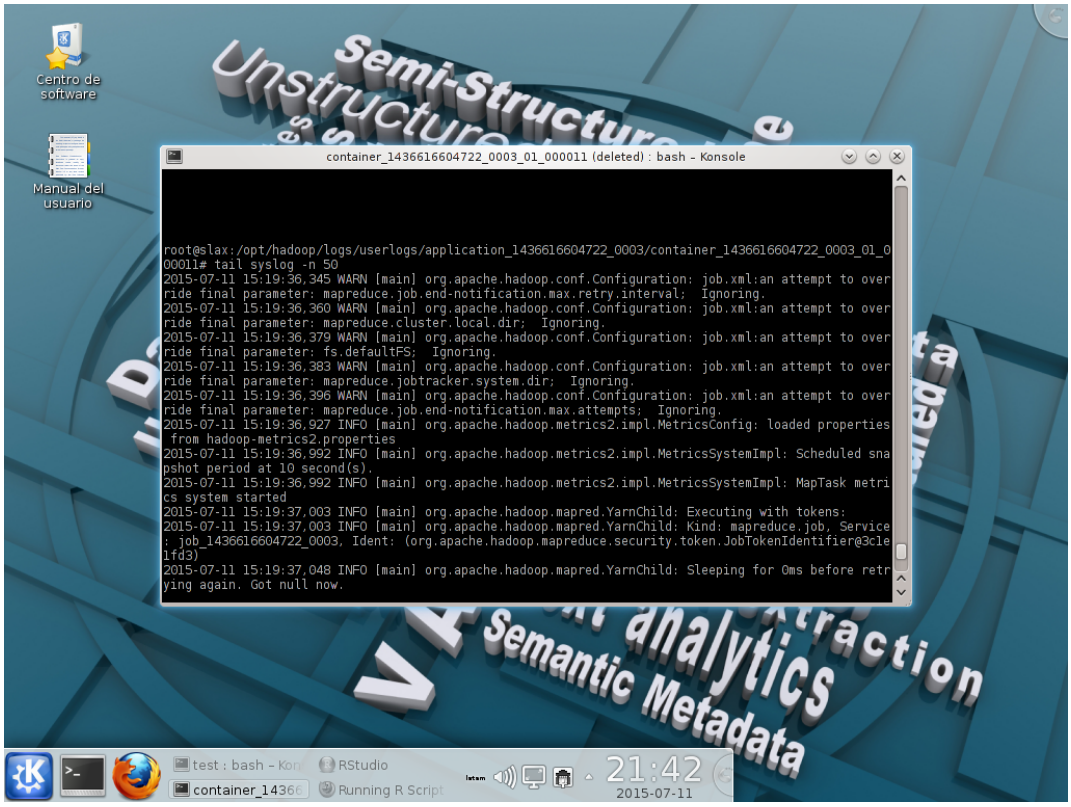


Figura 2.4: Linux con interfaz gráfica KDE

Linux es un sistema operativo al estilo Unix y compatible con POSIX. Fue programado en distintos lenguajes, mayormente en C y ensamblador. Posee soporte para una variedad muy amplia de arquitecturas, notablemente se tiene x86-64, ARM, PowerPC, Itanium. Utiliza un Kernel de tipo monolítico, como se puede observar en la imagen 2.1. Actualmente se encuentra en su versión 4.1.2 [9]. Linux no tiene un mercado principal, al ser un sistema operativo libre, su código fuente está disponible para cualquier persona que desee descargarlo, es utilizado en una gran cantidad de servidores web en Internet y también es utilizado en mucho teléfonos móviles, también se acostumbra a ser utilizado en computadores personales, sistemas embebidos, clusteres, entre otros. El manejo de la interfaz gráfica de Linux se realiza sobre un sistema llamado X Window System, provee una interfaz entre los comandos de consola y el diseño visual el cual se quiera recrear. Existen distintas interfaces gráficas, Gnome, Cinnamon, Unity, LXDE, KDE (Véase la figura 2.4), entre otras. También se acostumbra a utilizar Linux mediante una Shell o consola, sin interfaz gráfica. Uno de los sistemas operativos móviles (Android) con mayor mercado está basado en su Kernel. Se acostumbra a realizar distintas distribuciones tomando Linux como base.

2.2.5.1. Slax

En el mundo de los sistemas operativos existen distintas distribuciones basadas en Linux como se mencionó en el punto 2.2.5. *Slax* es una de estas distribuciones, es un sistema operativo al estilo Live-CD (léase el punto 2.2.3.2) el cual utiliza KDE4 como su entorno gráfico, viene con soporte de zram activado por defecto (tecnología para compresión automática de memoria RAM). Soporta un amplio rango de sistemas de archivos, EXT, btrfs, FAT y NTFS. [10]. Uno de los puntos clave de Slax es su soporte modular, permite añadir cualquier programa compatible con Linux en forma de módulo y su activación es totalmente en tiempo real, los módulos son comprimidos en formato sb con SquashFS y luego para ser montados en el sistema de archivos se utiliza AuFS, estos dos sistemas de archivos serán explicados más adelante.

2.2.5.1.1. SquashFS Es un sistema de archivos comprimido de solo-lectura para Linux. Fue inventado para sistemas de archivos normales que necesiten de la opción de solo-lectura, para almacenar archivos o para dispositivos que tengan muy poca capacidad donde se necesite todo el espacio posible [11]. SquashFS juega un papel importante en Slax ya que los módulos utilizados son comprimidos con SquashFS y llevados a un archivo de formato sb. Al momento de activar nuevamente un módulo, este es montado en el sistema de archivos utilizando AuFS explicado más adelante.

2.2.5.1.2. AuFS Advance multilayered unification File System o AuFS es un sistema de archivos que implementa union mount para otros sistemas de archivos. Un union mount permite montar distintas partes de un sistema de archivos (conocidas como ramas) en otro lugar totalmente distinto, haciéndole creer al usuario que todos los archivos pertenecen a un mismo sistema de archivos [12]. AuFS está basado en UnionFS, otro sistema de archivos que implementa union mount. Entre las características de AuFS se tiene:

- Unir distintos directorios en un solo sistema de archivos virtual
- Especificar banderas de permisología de escritura a cada una de las ramas

- Manipulación dinámica de ramas

2.3. Lenguajes de Programación

Un *lenguaje de programación* es un conjunto de palabras diseñadas para comunicar instrucciones a un computador. También se podría definir como aquel lenguaje que permite especificar de manera precisa sobre qué datos debe operar una computadora, como deben ser almacenados o transmitidos y que acciones debe tomar bajo una variada gama de circunstancias. La descripción de un lenguaje de programación usualmente se divide en dos componentes: la sintaxis (forma en la que se escribe) y la semántica (lo que significa).

Esos lenguajes suelen clasificarse en lenguajes interpretados y compilados. Los *lenguajes interpretados* tienen un intérprete específico que obtiene como entrada un programa y ejecuta las acciones escritas a medida que las va procesando; mientras que los lenguajes compilados son llevados a un programa ejecutable utilizando un compilador, este obtiene como entrada un programa y traduce las instrucciones las cuales pueden servir de entrada para otro interprete o compilador.

2.3.1. R

El lenguaje de programación *R* está especializado para probabilidades y estadística. Es un proyecto de GNU el cual es similar al lenguaje S. R puede ser considerado como una implementación distinta de S, con algunas diferencias importantes, una de las ventajas es que gran parte del código escrito en lenguaje S puede correr en R.

R provee una gran variedad de técnicas estadísticas y gráficas, es un lenguaje altamente extensible mediante la inclusión de paquetes o librerías hechas por terceros, muchos de estos paquetes sirven para resolver problemas específicos de una manera sencilla. Una de las fortalezas de R es la facilidad con la que gráficos bien diseñados y de alta calidad pueden ser realizados, incluyendo símbolos matemáticos y formulas donde sea necesario. R posee un conjunto de herramientas de software para facilitar la manipulación, cálculo y despliegue de gráficos. Incluye:

- Formas efectivas de hacer manejos de datos y almacenamiento.
- Suite de operadores para cálculos con arreglos, en particular matrices.
- Una gran y coherente colección de herramientas para análisis de datos.
- Facilidades gráficas para análisis de datos y despliegue por pantalla o impresora.

Para programar en R existe un IDE llamado RStudio, el cual es gratis y software libre. Este está disponible en 2 versiones: RStudio de escritorio, donde el programa corre localmente; y RStudio Server, que permite acceder a RStudio usando un navegador mientras corre en un servidor remoto donde todos los cálculos serán realizados. RStudio está escrito en el lenguaje de programación C++ y usa el framework

Qt para su interfaz gráfica de usuario. Para más información sobre la instalación de RStudio, verificar el punto .4.

R tiene una potencia increíble la cual puede ser aprovechada en el área de Big Data, una vez más entran los problemas explicados anteriormente (punto 2.1.1), donde se tiene una gran cantidad de datos la cual no puede ser procesada de la forma tradicional. Para esto se puede unir el poder y la facilidad de uso de R con la potencia que provee Hadoop (para ver Apache Hadoop adelantar hasta el punto 2.4). Mediante el uso del sistema de paquetes de R, se puede hacer uso de librerías que permitan conectarse con Hadoop para así realizar el procesamiento a estos grandes volúmenes de datos sin mayor problema.

2.3.2. Java

El lenguaje *Java* es de propósito general el cual es orientado a objetos, concurrente, basado en clases, portable y especialmente diseñado para tener pocas dependencias en su implementación como sea posible. Se basa en principios como WORA (write once, run everywhere) donde básicamente cualquier programa compilado en este lenguaje puede correr en cualquier equipo sin necesidad de recompilar (portabilidad). Todo esto es posible ya que Java corre en una máquina virtual llamada Java Virtual Machine (JVM) lo cual lo hace independiente de la arquitectura de la computadora donde esté corriendo. Este lenguaje de programación deriva mucho de lenguajes como C y C++, pero con menos funcionalidades de bajo nivel. Una de las características más importantes de Java (aparte de su gran portabilidad) es su manejo automático de la memoria. Java tiene una implementación de recolección de basura automática la cual se encarga de manejar la memoria en el ciclo de vida de un objeto.

Uno de los puntos importantes de Java en el mundo de Big Data es que Hadoop, el framework por excelencia de este mundo, es totalmente dependiente de Java debido a que una gran parte de este framework fue desarrollado sobre Java.

2.4. Apache Hadoop

Apache Hadoop es un proyecto software libre de la comunidad Apache. Es un framework que facilita el procesamiento de grandes volúmenes de datos de forma distribuida a través de clusteres usando modelos sencillos de programación. Está diseñado para escalar desde un servidor sencillo hasta miles de nodos los cuales pueden ser heterogéneos. Hadoop trata de ser confiable, proveer alta disponibilidad y manejo de fallos. Actualmente se encuentra en su versión 2.7.1 (06 Julio, 2014)[13].

2.4.1. Common

Es un conjunto de utilidades que permiten el soporte a otros proyectos de Hadoop, estas utilidades son consideradas como el núcleo del framework que provee servicios esenciales para otras aplicaciones basadas en Hadoop. Hadoop fue desarrollado casi en su totalidad en Java, este provee un conjunto de librerías que permiten la interacción con el cluster.

Hadoop posee distintas implementaciones de ciertos componentes de forma nativa, esto por cuestiones de rendimiento o por la no-disponibilidad de Java en el cluster. Esos componentes están disponibles en un único archivo el cual es llamado *librería nativa de Hadoop*.

El framework promete ser muy estable, este puede ayudar a solventar distintos problemas, esos problemas normalmente tienen un punto en común y es la cantidad de datos a utilizar, pueden ser enormes cantidades. Algunos de los problemas que se pueden encontrar son:

- Manejo de grandes volúmenes de datos
- Datos sociales, conseguir relaciones entre personas o buscar los temas actuales en distintas sociedades
- Obtener datos de clientes en páginas web y observar su comportamiento en dicha página
- Analizar logs para poder tener una mejor seguridad, se pueden tomar acciones estudiando comportamientos pasados o reaccionar en tiempo real según comportamientos analizados
- Procesar datos de un gran grupo de sensores los cuales pueden encontrarse en un área específica
- Mediante el análisis de datos de geolocalización se pueden optimizar rutas para compañías de entregas y tener una reducción de costos
- Análisis de grandes volúmenes de datos en forma de imágenes descargadas desde satélites
- Manejo de servidores de correo, organización y clasificación de estos
- Utilizar datos públicos relacionados con el área de la salud para planificación estratégica en casos específicos
- Buscar patrones de comportamiento de ciertas enfermedades para ser diagnosticadas de una manera rápida

Como se puede observar, Hadoop puede ayudar a resolver estos problemas y más. En cualquier área de investigación la cual requiera de análisis de datos y que este análisis pueda ser realizado mediante algoritmos.

2.4.2. Hadoop Distributed File System (HDFS)

Existen sistemas los cuales necesitan almacenar una cantidad de datos muy grande, esta cantidad de datos puede que no pueda ser almacenada dentro de un solo nodo físico, por lo tanto, se ha recurrido al almacenamiento en sistemas de archivos distribuidos, estos permiten mediante una conexión de red y distintos computadores compartir archivos de manera tal que para los nodos es transparente el lugar donde se almacenan dichos datos.

Hadoop Distributed File System, o HDFS como también es conocido, es un sistema de archivos distribuido el cual busca solucionar el problema del espacio distribuyendo los datos almacenados en distintos nodos, esta distribución se realiza mediante la replicación de datos en distintos nodos, esto para poder asegurar una mejor disponibilidad de los datos.

2.4.2.1. Características

HDFS posee las siguientes características las cuales lo hacen relucir frente a otros sistemas de archivos:

- Manejo de grandes archivos: cuando se hace referencia a grandes archivos, son aquellos que pueden pesar cientos de Mega-Bytes, Giga-Bytes, Tera-Bytes, Peta-Bytes, etc.
- Compatibilidad con hardware: fue diseñado para aceptar hardware común, que se puede encontrar en cualquier tipo de servidores. No se necesitan marcas específicas, ni modelos específicos de discos para poder funcionar.
- Acceso a datos en flujos: la construcción de HDFS fue pensada para manejar datos de manera eficiente, se maneja sobre un patrón el cual se escribe una vez un dato, pero se leen múltiples veces. HDFS permite ir leyendo datos bajo demanda lo cual es una ayuda para reducir el rendimiento debido a que no se debe esperar una copia de todo un conjunto de datos para funcionar.
- Tolerancia a fallos: para poder tener siempre disponible los datos en caso de ser requeridos, utiliza la replicación de los datos en distintos nodos (por defecto 3).

Pero, también tiene algunas desventajas:

- Acceso a datos con baja latencia: fue creado para manejar grandes volúmenes de datos, por lo tanto, está optimizado para tener altas tasas de transferencia.
- Escritura múltiple: los archivos deben ser escritos por un único proceso, las escrituras siempre son realizadas al final de los archivos. No Existe soporte para escribir en un mismo archivo por múltiples procesos al mismo tiempo.
- Pequeños archivos: HDFS tiene un límite para almacenar archivos debido a que crea metadata de los archivos almacenados en memoria, directorios y bloques, lo cual limita la cantidad de datos a tener almacenados dependiendo de la cantidad de memoria que posea el nodo.

2.4.2.2. Arquitectura

Antes de hablar de la arquitectura básica de HDFS se debe hablar sobre la manera la cual HDFS almacena los datos. Estos datos son almacenados de una forma muy parecida a los sistemas de archivos convencionales donde se separan los discos en bloques de un tamaño predeterminado. HDFS adopta este concepto de bloques y toma una unidad mucho más grande que los discos convencionales, 64MB, mientras que los discos convencionales poseen bloques de 512B, dependiendo de su configuración. De tener un dato mayor de 64MB este es cortado en distintos trozos de 64MB (por defecto) los cuales permitirán almacenar y distribuir todos los trozos en un cluster y así poder realizar el almacenamiento de forma distribuidas.

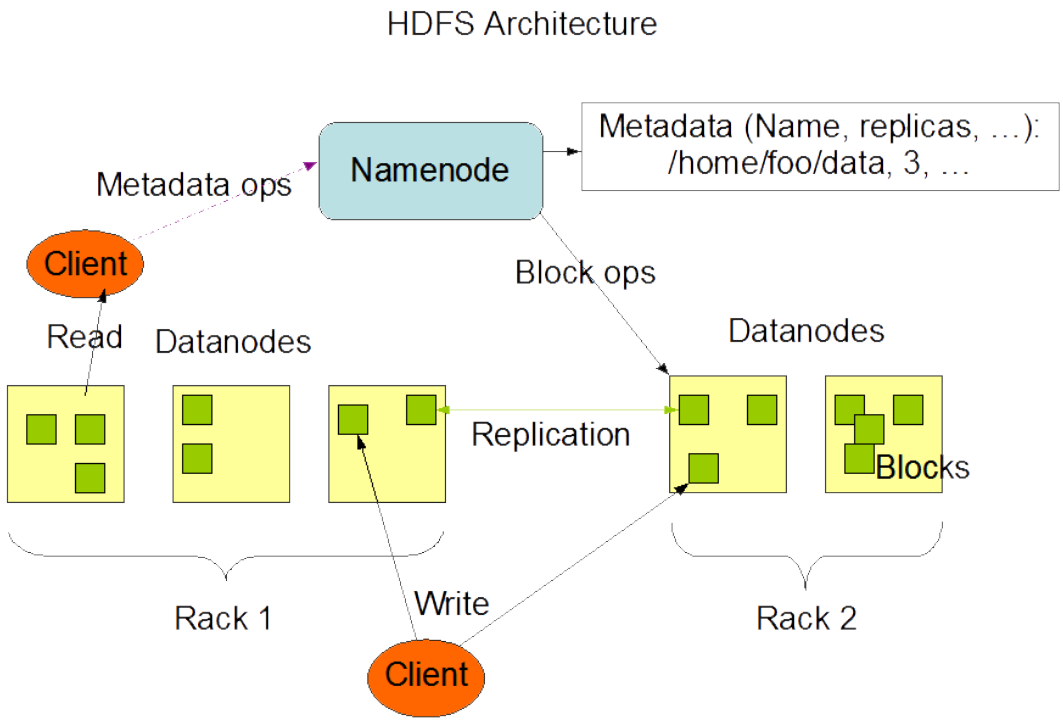


Figura 2.5: Arquitectura de HDFS

HDFS tiene 2 tipos de nodos para realizar sus operaciones, los nodos maestros (Namenode) y los nodos esclavos (Datanode):

- Namenode: se encarga de la administración y el mantenimiento del sistema de archivos y la metadata asociada a todos estos. El mantenimiento consiste en mantener dentro de un árbol todos los archivos y directorios para poder ser accedidos con mayor facilidad. Toda esta información es guardada en un medio de almacenamiento persistente en forma de log y de imagen para luego ser leída. Almacena los datos relacionados con la posición de un bloque para un archivo en específico durante un periodo de tiempo, estos datos son actualizados al iniciar el sistema y cada cierto tiempo. Como se puede observar en la imagen 2.5[14] el Namenode puede ser un punto de fallo simple debido a que todas las operaciones a ser realizadas están relacionadas con este.
- Datanode: almacena y distribuye los bloques entre los distintos nodos. La distribución de los bloques es realizada cuando reciben un aviso desde un namenode o algún cliente (explicado en el siguiente punto) solicita una distribución. Una vez realizada la distribución, estos realizan un reporte al namenode, este reporte se realiza periódicamente y contiene los bloques los cuales están siendo almacenados en ese nodo en específico
- Clientes: son los usuarios del sistema de archivos

Como se mencionó anteriormente, existe una dependencia enorme con el namenode debido a que en este se encuentra toda la metadata necesaria para poder rearmar un dato, ya que estos están divididos en distintos bloques. Por eso, es muy importante tener mecanismos para poder mantener estos nodos siempre activos, Hadoop provee dos mecanismos clave para poder lograrlo:

1. Se realiza un respaldo de la metadata actual. Este respaldo normalmente es realizado en distintos sistemas de archivos, uno de los utilizados es el sistema de archivos local del namenode y otro de los utilizados es en algún nodo secundario el cual provea almacenamiento por red mediante NFS

2. Se podría mantener un namenode secundario el cual se activaría al fallar el namenode principal. Mientras el namenode principal esté en funcionamiento, el secundario no se deja mostrar en el cluster como un namenode. Este se encarga de realizar las mismas funciones que el namenode principal, es decir, mantener la metadata de todos los directorios y archivos

2.4.2.3. Escalabilidad

La manera la cual fue construido HDFS permite que sea un sistema de archivos escalable horizontalmente, el problema se puede presentar al momento de ingresar nuevos nodos al sistema y estos lo satures, en este apartado se hablará sobre las limitaciones de escalabilidad y a que están asociadas.

Uno de los problemas principales viene dado por el namenode, el cual mantiene la metadata del cluster en memoria RAM, lo cual es un problema debido a que los servidores tienen un límite de memoria RAM el cual puede ser instalado. Este problema se presenta debido a que cada directorio, archivo o bloque almacenado ocupa alrededor de 150B, lo cual hace que al tener grandes volúmenes de datos, este problema se vea presente en un cluster Hadoop.

Una de las fallas existentes en un cluster Hadoop es que durante el uso de este, la relación bloque-archivo puede afectar al namenode, debido a que el tamaño de los bloques tiende a disminuir lo cual hace que se tenga más metadata y eso perjudica el sistema ya que se invierte mucho tiempo actualizando las tablas para mantener el sistema en orden.

Un problema relacionado con las tablas las cuales deben ser actualizadas es la cantidad de mensajes que deben ser enviados desde los datanodes hacia el namenode, al tener que enviar muchos mensajes, la red del cluster podría colapsar como también puede colapsar el namenode debido a que este tiene una capacidad de procesamiento limitada. La carga relacionada con los datanodes la cual es sometida el namenode es proporcional a la cantidad de datanodes que se encuentren en el cluster. Se puede decir que el cuello de botella principal de un cluster Hadoop es el namenode debido a que tanto clientes como datanodes dependen de este.

2.4.3. YARN

YARN (Yet Another Resource Negotiator) es el administrador de recursos de un cluster Apache Hadoop, también es denominado MapReduce 2.0 o MRv2. La llegada de YARN represento un cambio significativo para Hadoop (de su versión 1.0 a la 2.0) en general porque permitió aplicar distintos modelos de procesamiento sobre los clusters (ver figura 2.6 [15]), antes con MRv1 los nodos del clúster debían dedicarse única y exclusivamente a un modelo de procesamiento orientado a MapReduce. Entre los modelos de procesamiento que soporta YARN se puede mencionar: Apache Hadoop MapReduce, Apache Spark, Apache HAMA, Apache Giraph y Open MPI [16].

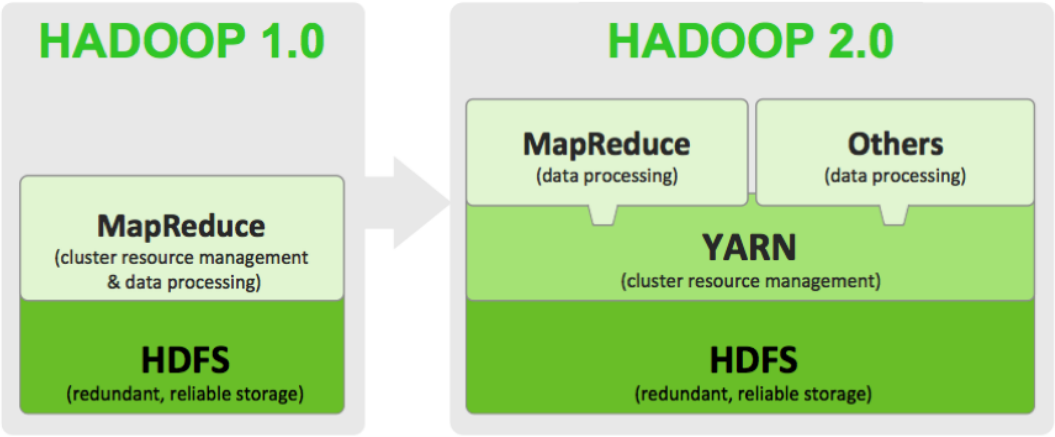


Figura 2.6: Arquitectura Apache Hadoop antes y despues de YARN

2.4.3.1. Características

- Compatibilidad: las aplicaciones existentes de MapReduce hechas para correr en Hadoop 1 (con MRv1) pueden correr en YARN sin ningún problema.
- Escalabilidad: ciertos problemas de cuello de botella en grandes clústers usando MRv1 fueron arreglados con YARN. Con MRv1 los clústers tenían un límite de escalabilidad de alrededor de 4000 nodos.
- Procesamiento: YARN soporta distintos modelos de procesamiento y puede acoplarse para soportar muchos más.

2.4.3.2. Arquitectura

YARN se basa fundamentalmente en el paradigma maestro/esclavo, estando en el tope el ResourceManager el cual es el encargado de manejar todos los recursos del cluster (ver figura 2.6 [17]).

Por nodo hay un NodeManager encargado de manejar recursos específicos a una tarea corriendo en un nodo (el único recurso que administra por ahora es la memoria) [18].

Por aplicación hay un ApplicationMaster que puede ser alguna librería específica de algún framework que es la encargada de negociar recursos con el ResourceManager y coordinar las tareas con el NodeManager.

El ResourceManager asigna los recursos del clúster de manera abstracta a través de contenedores (Containers) los cuales incorporan elementos como memoria, CPU, disco, red, entre otros.

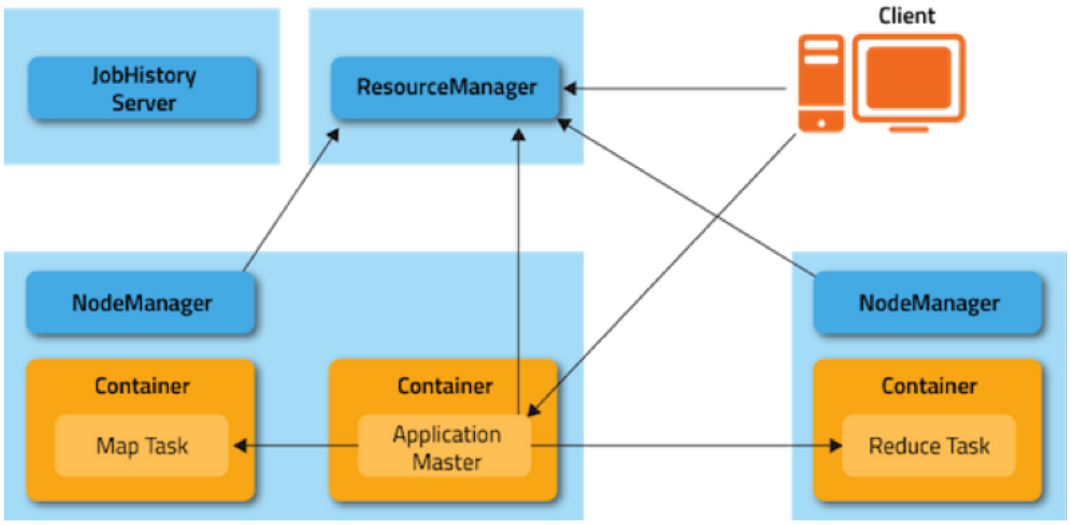


Figura 2.7: Arquitectura de YARN

2.4.4. MapReduce

MapReduce es un modelo de programación con una implementación asociada al procesamiento y generación de grandes cantidades de datos. Los usuarios especifican una función de map que procesa pares clave/valor para generar un grupo intermedio de pares clave/valor y una función de reduce que combina todos los valores intermedios.

Programas escritos de esta manera son automáticamente paralelizados y ejecutados en un clúster Apache Hadoop. El sistema en tiempo de ejecución se encarga de cosas como los detalles de particionar los datos de entrada, la planificación del programa en ejecución a través de todas las maquinas, encargarse de manejar fallos y administrar la comunicación necesaria entre maquinas. Esto permite a programadores sin experiencia con sistemas paralelos y distribuidos a utilizar fácilmente los recursos del sistema.

MapReduce fue desarrollado por Google y expuesto al resto del mundo en una publicación hecha por ellos mismos en diciembre del 2004, Google usa MapReduce para indexar páginas web [19].

Específicamente la implementación de Apache Hadoop MapReduce es un framework con una serie de librerías para facilitar escribir aplicaciones usando este esquema.

2.4.4.1. Conceptos básicos de Map Reduce

Los programas que funcionan bajo este modelo están compuestos de dos procedimientos esenciales llamados map y reduce, básicamente entre esos dos procedimientos se puede resumir todo el flujo de datos que hace el programa. MapReduce también incorpora un esquema de tolerancia a fallos que permite responder de una forma eficaz a casi cualquier problema que se pueda presentar al momento de la ejecución del programa. Cada uno de estos conceptos será desarrollado más adelante.

2.4.4.1.1. Mapping Lists La primera fase de un programa que corre bajo un esquema de MapReduce es llamada mapping. Una lista con elementos de datos es dada en un instante de tiempo a una función llamada Mapper (Mapping function), la cual transforma cada elemento individualmente a un elemento de datos de salida (ver figura 2.8 [20]) [9].

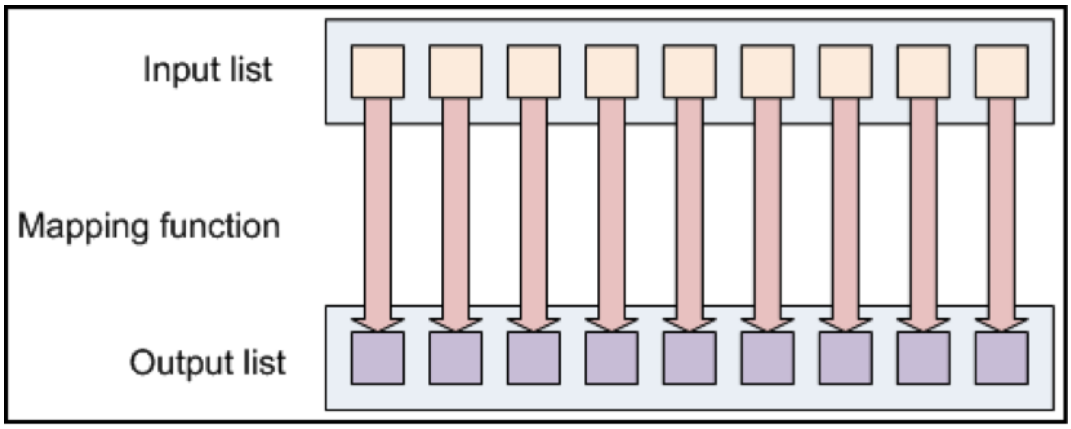


Figura 2.8: Mapping Lists: cada elemento de entrada genera un elemento de salida

2.4.4.1.2. Reducing Lists Reducing permite juntar todos los valores. Una función reducer recibe un iterador de valores de entradas de una lista de entrada. Entonces combina estos valores y retorna un único valor de salida. Reducing es usado frecuentemente para producir un resumen de los datos, cambiando un gran volumen de datos a una pequeña cantidad de los mismos (ver figura 2.9 [21]) [9].

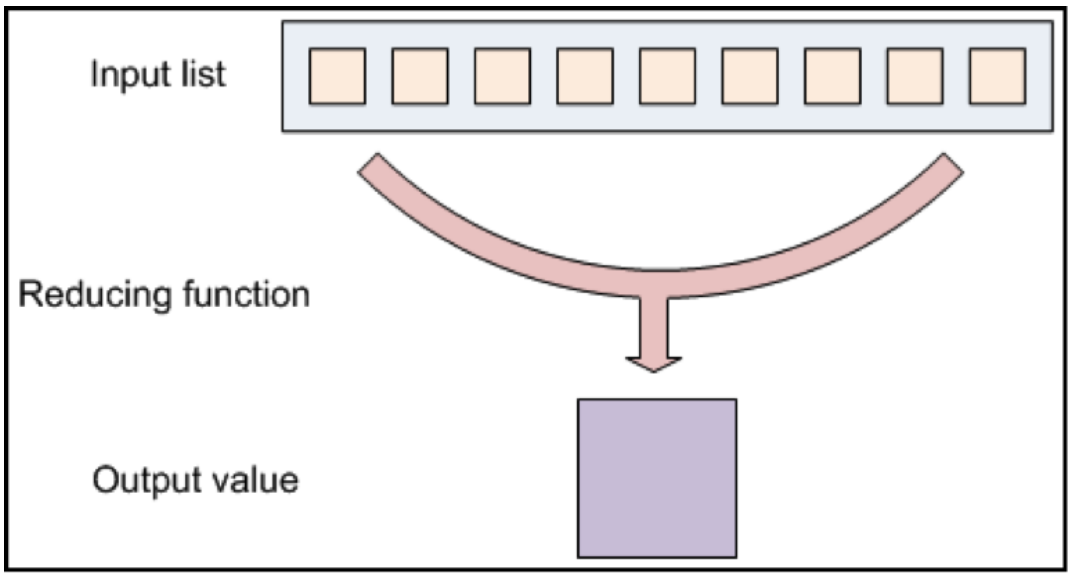


Figura 2.9: Reducing Lists: genera una única salida de una lista de entrada

2.4.4.1.3. Flujo de Datos La entrada de datos para MapReduce viene típicamente de archivos cargados en el clúster en HDFS (ver 2.4.2). Estos archivos son igualmente distribuidos a través de todos los nodos del clúster. Un programa que corre con un esquema de MapReduce involucra correr tareas de

mapping en varios o todos los nodos del clúster. Cada una de estas tareas de mapping es equivalente, es decir, no tienen identificadores particulares asociados. De esta manera, cualquier mapper puede procesar cualquier archivo de entrada. Cada mapper carga un conjunto de archivos locales a la máquina que los está procesando.

Cuando la fase de mapping haya completado, los pares (clave/valor) intermedios deben ser cambiados entre máquinas para mandar todos los valores de la misma clave a un reducer. Las tareas de reduce se esparcen a través de los mismos nodos que los mappers. Este es el único paso de comunicación en MapReduce. Las tareas de map individuales no cambian información una con otra, y ni están conscientes de la existencia de las otras tareas. Similarmente, tareas diferentes de reduce no se comunican una con otra. El usuario nunca tiene control de cómo es la transferencia de datos, toda la transferencia es manejada por la plataforma de Hadoop MapReduce; guiada implícitamente por las diferentes claves asociadas a los valores. Si los nodos del clúster fallan, las tareas deben poder ser reiniciadas.

Finalmente los componentes que están presentes en el flujo de datos de un programa que corre con un esquema de MapReduce son los siguientes [9]:

- Input reader: se encarga de dividir los datos de entrada al tamaño apropiado (suele ser 64 MB hasta 128MB) y el framework asignará cada pedazo a una función de map. Estos datos generalmente son leídos del HDFS y generan pares clave/valor.
- Función de Map: la función de Map agarra una serie de pares clave/valor, procesa cada uno, y genera cero o más salidas de pares clave/valor.
- Función de Partición: cada salida de la función de map es colocada en un reducer en particular por la función de partición de la aplicación. A la función de partición se le da la clave y el número de reducers y retorna un índice para el reducer deseado.
- Función de Comparación: la entrada de cada función de reduce es obtenida de la maquina donde el map corrió y son ordenadas usando la función de comparación de la aplicación.
- Función de Reduce: el framework llama a la función de reduce de la aplicación una vez por cada clave única ordenada. Reduce itera por los distintos valores que están asociados con esa clave y puede producir cero o más salidas.
- Escritor de salida: la salida de la función de reduce es escrita en algún repositorio, usualmente el mismo HDFS.

2.4.4.1.4. Tolerancia a fallos La forma en la que MapReduce o Hadoop en general logra la tolerancia a fallos es a través del reinicio de las tareas. Tareas individuales corriendo en los nodos (TaskTrackers) están en constante comunicación con la cabeza del nodo del sistema, llamado JobTracker. Si un TaskTracker falla al comunicarse con el JobTracker por un período de tiempo (por defecto en Hadoop, un minuto), el JobTracker va a asumir que el Tasktracker en cuestión fallo. El JobTracker conoce que tareas de map y reduce fueron asignadas a cada TaskTracker [9].

Si el trabajo todavía está en la fase de mapping, entonces otros TaskTrackers serán asignados de volver a ejecutar todas las tareas de map del TaskTracker en específico que fallo. Si el trabajo está en su

fase de reduce, entonces los otros TaskTrackers van a volver a ejecutar automáticamente las tareas del TaskTracker que fallo [9].

Las tareas de reduce una vez completadas, deben ser escritas de regreso al HDFS. Sin embargo, si un TaskTracker ya ha completado dos de tres tareas de reduce asignadas a él, simplemente la tercera debe ser ejecutada por otro. Las tareas de map son un poco más complicadas: incluso si un nodo ha completado diez tareas de map, es posible que las tareas de reduce no hayan copiado todos sus datos de entrada de esas tareas map. Si ese nodo falla la salida de las tareas map queda inaccesible. Por lo tanto cada tarea de map ya completada debe volverse a ejecutar para hacer que sus resultados estén disponibles al resto de las tareas reduce. Todo esto es manejado automáticamente por la plataforma de Hadoop [9].

Esta tolerancia a fallos necesita que los programas en ejecución sean los más individuales posible, es decir, si cada tarea map o reduce tuviese su identidad propia y se comunicara con el mundo exterior (a través de la red por ejemplo) o con otras tareas, entonces reiniciar esas tareas se vuelve un poco más complicado porque hay que tomar en cuenta el estado que tenía en el momento que fallo. Este proceso es realmente complicado y propenso a errores. MapReduce simplifica este problema de cierta forma evitando las identidades o que las tareas se comuniquen entre ellas. Una tarea individual solo puede trabajar con sus propios datos y conoce solo sus propias salidas, para hacer este fallo y proceso de reinicio limpio y no dependiente [9].

2.4.5. Herramientas del Ecosistema Hadoop

Existen múltiples herramientas creadas para funcionar bajo el ecosistema Hadoop, se tienen distintos subconjuntos de herramientas las cuales tienen utilidades similares pero siempre ofreciendo características únicas.

2.4.5.1. Administración de datos

Realizan el manejo de datos y el almacenamiento de estos en disco. En este apartado se especifican 2 herramientas de las cuales se habló anteriormente: Hadoop Distributed File System (HDFS) y Yet Another Resource Manager (YARN).

2.4.5.2. Acceso a datos

Se encargan de permitir a los usuarios interactuar con los datos, es decir, leer datos (bajo demanda o por lotes), escribirlos, procesarlos.

2.4.5.2.1. Apache HBase Es una base de datos distribuida orientada a columnas que funciona sobre HDFS, es ideal si se requiere de acceso en tiempo real a grandes cantidades de datos. Distintas compañías ofrecen soluciones para almacenar datos las cuales no fueron ideadas desde un principio para escalar masivamente y mucho menos para ser distribuidas. Otras compañías trabajan sobre sistemas manejadores de bases de datos relacionales y ofrecen soluciones para replicar datos y particionar estos, lo que trae

como consecuencia el manejo de una mayor cantidad de herramientas. En el caso de HBase, es una base de datos NoSQL, se puede decir que HBase es un almacén de datos en vez de una base de datos, esto debido a la gran cantidad de funcionalidades que deja de prestar, que a diferencia de los sistemas manejadores de bases de datos relacionales si incluyen. Algunas de estas funciones pueden ser: triggers, índices secundarios para columnas.

Entre las características más resaltantes de HBase se tiene:

- Escalabilidad linear y modular
- Velocidad de lectura y escritura consistente: HBase trata siempre de proveer al usuario una velocidad de respuesta consistente
- Replicación automática y configurable de tables: las tablas son distribuidas en todo el cluster y son automáticamente separadas y distribuidas en caso de que se tenga un nuevo ingreso de datos
- Soporte para caídas de nodos
- Compatibilidad con trabajos MapReduce
- Facilidad de uso con el API para Java
- Web Service de tipo REST-ful y Gateway Thrift con soporte de XML y datos binarios para su interacción fuera del ecosistema

Gracias a la compatibilidad con trabajos MapReduce, las tablas de HBase pueden servir tanto de entrada como salida de datos para trabajos MapReduce. Tres puntos clave en el uso de HBase relacionado con el Hardware y el software:

- Memoria RAM: es necesaria una gran cantidad de memoria RAM (a mayor cantidad de datos a procesar, mayor cantidad de memoria), esto relacionado con el 3er punto explicado a continuación.
- Sistema Operativo: se recomienda ser instalado en un sistema operativo de 64-bits debido a que pueden existir datos que realicen overflow al computador de ser ejecutados en un sistema operativo de 32-bits. Esta recomendación esta relacionada con todo el entorno Hadoop.
- Paginación: a menor cantidad de memoria RAM, mayor la cantidad de paginación que se realiza con dispositivos de almacenamiento masivo lo cual ralentiza el sistema en grandes proporciones.

Uno de los puntos negativos de HBase es que no ofrece un buen desempeño con pocas cantidades de datos, por lo que es recomendable utilizar sistemas manejadores de bases de datos relacionales, los cuales entregan un desempeño mucho mejor con pocas cantidades de datos [22]. También se debe tener en cuenta que HBase, como se mencionó anteriormente, no posee índices secundarios, especificaciones fuertes con respecto a los datos en las columnas, transacciones o triggers. Los fallos de rendimiento que se pueden percibir en HBase suelen suceder gracias a un hardware deficiente para su ejecución.

HBase provee distintos APIs para poder explotar sus funciones al 100 %, estos APIs fueron realizados para poder utilizar HBase desde otros lenguajes distintos a Java, el cual es el lenguaje nativo aceptado por HBase. Entre sus APIs se tiene compatibilidad con: Stargate, Thrift y Avro.

2.4.5.2.2. Apache Mahout Es una librería de los algoritmos más famosos de machine-learning, fue implementado sobre Hadoop utilizando el paradigma de MapReduce. Una vez que se tengan datos almacenados en HDFS, Mahout provee las herramientas necesarias para poder aplicar ciencia de datos sobre dichos datos. Mahout fue creado desde un principio para trabajar utilizando el paradigma MapReduce que es usado ampliamente en Hadoop, pero ahora está comenzando a tomar otro rumbo el cual provee compatibilidad con Spark (para información de Spark observar el punto 2.4.5.2.3) y H2O.

Mahout provee los siguientes algoritmos:

- Filtrado colaborativo: realiza recomendaciones de objetos según el comportamiento observado en grupos de datos
  - Filtrado colaborativo
  - Factorización de matrices ALS
  - Factorización de matrices con pesos
- Clasificación: aprende de grupos ya existentes y asigna objetos sin grupos a los que mejor concuerden
  - Regresión logística
  - Métodos bayesianos
  - Random Forest
  - Modelos ocultos de Markov
- Clusterización: toma objetos en una clase particular y los organiza según otros objetos de la misma clase o de una clase distinta
  - K-Medias
  - K-Medias disperso
  - K-Medias por Streaming
  - Clusterización espectral
- Reducción de dimensionalidad: permite reducir la cantidad de variables en un conjunto de datos
  - Descomposición de valores singulares
  - Análisis de Componentes Principales
  - Descomposición QR

2.4.5.2.3. Apache Spark Es un motor el cual permite procesar grandes volúmenes de datos mediante distintos algoritmos iterativos los cuales tienen una naturaleza de reutilizar datos constantemente, como los algoritmos de Machine-Learning. Por lo tanto, Spark trata de mantener la mayor cantidad de datos posibles a utilizar en memoria RAM para poder procesarlos de manera veloz. Provee una serie de APIs de alto nivel en Java, Python y Scala, también incluye una serie de herramientas para monitoreo general del sistema. Puede ser ejecutado sobre distintas plataformas, es 100 % compatible con Hadoop, también

es ejecutable como una aplicación standalone. Las fuentes de información pueden ser múltiples, como por ejemplo: HDFS, HBase, Cassandra.

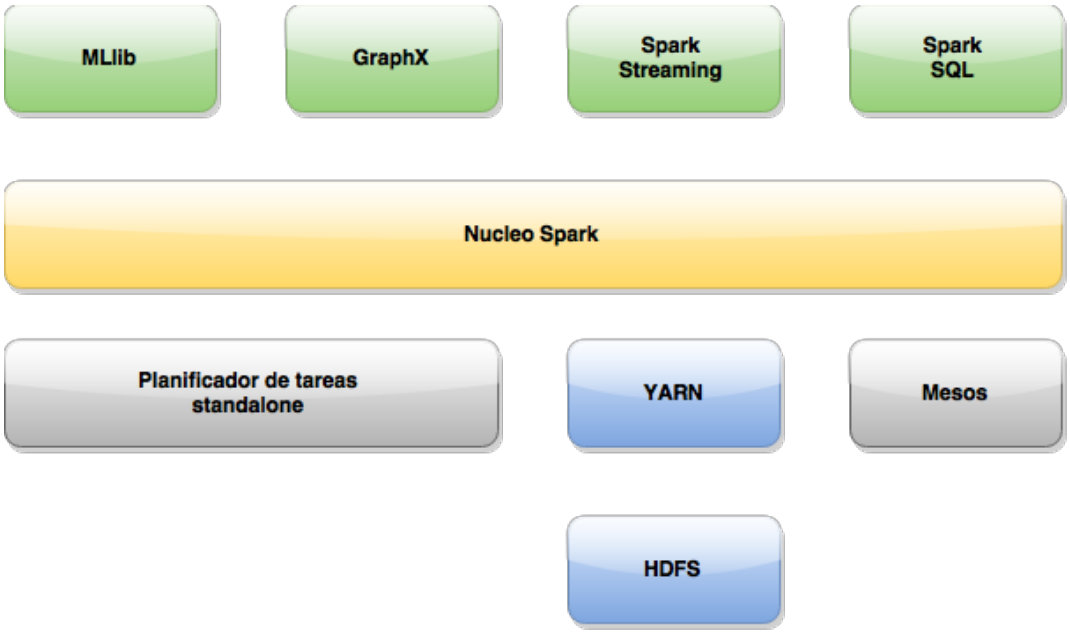


Figura 2.10: Arquitectura de Spark sobre Hadoop y otras herramientas

Como se puede observar en la figura 2.10, Spark provee una arquitectura la incluye distintas herramientas. A continuación la descripción de cada una de ellas:

- Spark SQL: unifica el acceso a data de forma estructurada, dicha data puede venir de distintas fuentes. Provee compatibilidad con Hive y conexiones JDBC
- MLlib: librería escalable facilitada por Spark que contiene algoritmos de Machine-Learning
- GraphX: API facilitado por Spark para realizar gráficos
- Spark Streaming: permite la construcción de aplicaciones las cuales soliciten datos a manera de flujo las cuales deben ser escalables y tolerante a fallos. Provee a los usuarios recuperación de tareas que no pudieron ser finalizadas.

La relación entre Hadoop y Spark es destacable ya que este último es compatible con distintas herramientas desarrolladas para un ecosistema Hadoop. Lo más importante es el hecho que Spark puede acceder a todos los datos almacenados en Hadoop, mediante HDFS, HBase, MongoDB, Cassandra o Hive. La ventaja principal de Spark es el manejo de datos en memoria RAM, pero la memoria es un recurso limitante, esto hace surgir la siguiente pregunta ¿Qué ocurre si algún dato no puede ser cargado en memoria? Spark permite manipular datos directamente en disco. Otro de los puntos positivos es que Spark trata de reducir la cantidad de código a ser desarrollado mediante distintas APIs de altos nivel.

En el núcleo de Spark se encuentra una abstracción de programación llamada RDD (Resilient Distributed Datasets), esta abstracción representa una colección de objetos distribuidos mediante distintos nodos de cómputo, estos objetos pueden ser manipulados en paralelo.

2.4.5.2.4. Elasticsearch Es un motor de análisis de datos altamente escalable, basado en búsquedas de texto y totalmente software libre. Permite almacenar, buscar y analizar grandes volúmenes de datos de forma rápida muy cercana al tiempo real. Es usado generalmente como una poderosa capa que permite proveer todas sus funciones relacionadas con búsquedas de texto a otras aplicaciones que necesiten realizar acciones de búsqueda complejas. Elasticsearch está construido sobre Apache Lucene que no es más que un motor de búsqueda de alto rendimiento basado en texto que está escrito totalmente en Java.

Entre los casos de uso más famosos de elastic se tienen a las siguientes compañías:

- Tango: utiliza el stack ELK para monitorear logs y medir el rendimiento de su compañía en tiempo real. Todo esto viene de querer asegurarse que sus servicios están siendo bien prestados a 250 millones de usuarios
- Soundcloud: se trataba de crear un motor de búsqueda que fuese flexible y de fácil utilización, para poder entregar a los usuarios del sistema (30 millones aproximadamente) búsquedas rápidas
- Mc Graw Hill: crear un sistema auto-adaptable de aprendizaje. Se utilizó Elasticsearch para buscar, indexar y calificar contenido
- Github: se utiliza Elasticsearch para indexar más de 8 millones de repositorios
- Mercadolibre: todas las búsquedas realizadas en Mercadolibre son hechas a través de Elasticsearch

Algunos de los conceptos que se deben entender para poder comprender totalmente Elasticsearch son los siguientes:

- Cercano a tiempo real: es una plataforma de búsqueda que ofrece resultados cercanos al tiempo real, lo que significa que existe una pequeña latencia (normalmente un segundo) desde el momento que se indexa un documento hasta el momento que se vuelve accesible
- Índices: es una colección de documentos que tienen características similares. Un índice es identificado con un nombre (en el caso de Elasticsearch, todo en minúsculas) y este nombre es usado para hacer referencia al índice al momento de realizar operaciones
- Tipo: dentro de un índice, se pueden definir uno o más tipos. Un tipo es una categoría o partición lógica del índice cuya semántica es totalmente configurable por el usuario. En general, un tipo es definido para documentos que tienen campos en común
- Documento: es la unidad básica de información que puede ser indexada. Este documento está expresado en formato JSON
- Fragmentos: un índice puede almacenar una gran cantidad de datos que pueden exceder los límites de almacenamiento de un único nodo. Para poder solucionar esto, Elasticsearch crea subdivisiones de los índices para luego ser almacenadas en otros nodos. Cada fragmento es en sí un índice independiente. Los fragmentos son importantes ya que permiten escalar horizontalmente un cluster y aparte permite distribuir y paralelizar las operaciones

- Réplicas: en una red en la cual pueden existir muchas fallas es muy importante crear réplicas de los fragmentos almacenados ya que esto permite tener una alta disponibilidad del recurso a buscar en caso de que un nodo o un fragmento se encuentre dañado, también permite que las búsquedas puedan ser realizadas en distintos nodos lo cual aumenta la tasa de salida del servidor

2.4.5.2.5. Kibana Es una plataforma software libre que permite la visualización y análisis de datos almacenados en Elasticsearch. *Kibana* puede ser utilizado para buscar, observar e interactuar con los datos almacenados en los índices de Elasticsearch. Esta herramienta hace muy sencillo el entendimiento de grandes volúmenes de información mediante gráficas, tablas y mapas. Todos los datos desplegados en Kibana son cargados en tiempo real desde Elasticsearch.

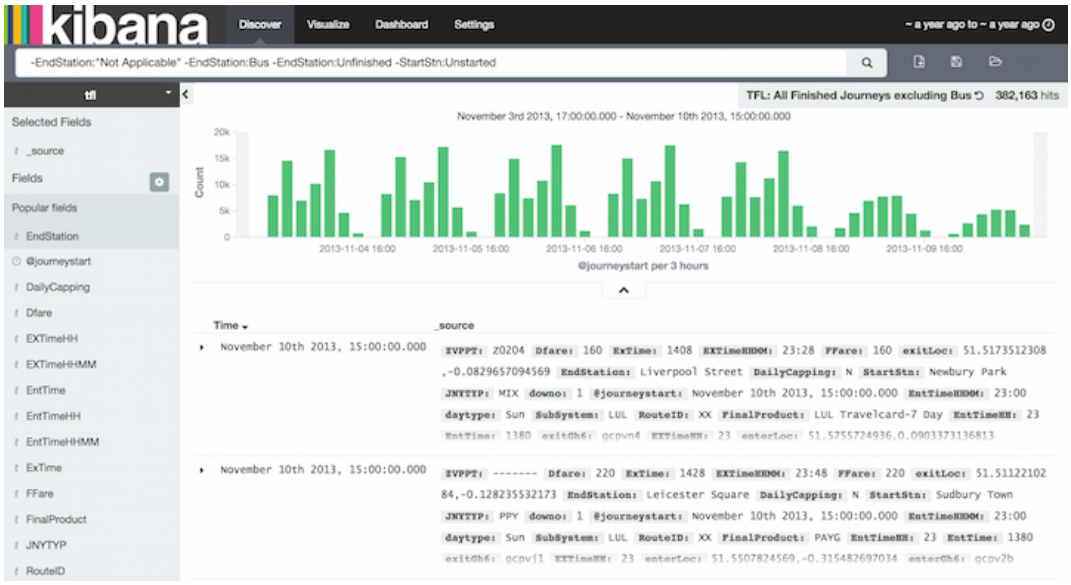


Figura 2.11: Interfaz de Kibana

Como se puede observar en la figura 2.11 [23] la interfaz de Kibana es totalmente web, para poder hacer uso de Kibana solo es necesario un navegador para poder acceder a la herramienta.

2.4.5.3. Integración

Permiten hacer una abstracción para facilitar tareas como la extracción de los datos, transformación, retención o replicación. Para facilitar este tipo de tareas se pensó más allá de proveer un SDK o una nueva librería para mejorar la eficiencia al momento de escribir aplicaciones usando un modelo MapReduce, sino que se proveen herramientas que hacen fácil la ejecución y automatización de estas tareas.

2.4.5.3.1. Logstash Es una herramienta utilizada para recibir, procesar y crear logs. *Logstash* provee un poderoso mecanismo para almacenar, consultar y analizar los logs que le sean provistos como entrada. El lenguaje de programación utilizado para crear Logstash fue Java. Toda entrada que capture Logstash puede ser modificada mediante filtros que permiten realizar múltiples cambios a estas entradas, desde

cambiar un campo en específico hasta cambiar todo el formato de un log. Logstash admite como entrada de datos: archivos, lecturas directas desde servidores (HTTP, S3), entradas desde la consola del sistema, websockets, etc. Como también provee un conjunto de formatos de salida (JSON, logs al estilo Ruby) y métodos donde almacenar esta salida. Logstash ofrece un sistema de plugins que permite al usuario añadirle funcionalidades extra a la herramienta, por ejemplo, Logstash no provee un soporte inicial de HDFS pero gracias a algunos plugins se puede tener salida hacia este.

Básicamente el funcionamiento de Logstash consta de 3 pasos:

1. Entradas: obtienen los datos de distintas fuentes
2. Filtros: modifican los datos
3. Salidas: entrega los datos ya modificados a sus destinatarios, los datos modificados pueden ser formateados mediante lo que Logstash llama como Codecs

2.4.5.3.2. Fluentd Es un coleccionista de datos de código abierto que permite la unificación de datos para luego esparcirlos hacia otras herramientas. Uno de los puntos clave de Fluentd es que este estructura todo los datos de salida en formato JSON. La herramienta sirve como una capa de unificación de datos para capturarlos, procesarlos y luego entregarlos a las herramientas que van a hacer uso de estos datos. Fue realizado en una combinación de C y Ruby, por lo que requiere de muy pocos recursos. El funcionamiento de Fluentd es muy parecido al explicado en Logstash, a partir de una entrada se realiza un procesamiento sobre esta que permite modificar esos datos de entrada para unificar la salida, esta salida también es totalmente configurable.

2.4.5.4. Operaciones

Son las herramientas que permiten administrar, monitorear y realizar operaciones sobre un cluster Hadoop. En este conjunto de herramientas podemos encontrar funciones como la planificación de tareas o la comunicación de las configuraciones de distintos nodos para realizar tareas automatizadas.

2.4.5.4.1. Apache ZooKeeper *ZooKeeper* es un servicio de coordinación distribuida el cual provee una serie de herramientas que ayudan a construir aplicaciones distribuidas para comunicar distintos procesos. Entre sus características encontramos:

- Simple: su núcleo es básicamente un sistema de archivos el cual expone ciertas operaciones a sus usuarios incluyendo algunas abstracciones como lo es el ordenamiento de los nodos y las notificaciones
- Facilita la interacción entre procesos: provee mecanismos los cuales permite a distintas aplicaciones comunicarse sin tener que ser necesariamente en tiempo real
- Alta disponibilidad: es ejecutado en distintos nodos o mejor llamados colecciones. Su diseño permite la alta disponibilidad del servicio, también ayuda a evitar puntos propensos a fallos debido a la replicación del servicio

- Expresivo: posee primitivas que pueden ser usadas para construir mecanismos complejos al momento de una aplicación comunicarse con otras.
- Facilidad para programar: provee una librería de código abierto la cual permite la utilización de todas las funciones de ZooKeeper, también existen comunidades que comparten distintos protocolos creados con ZooKeeper para poder realizar tareas específicas.

La forma de trabajar de ZooKeeper es parecida a como trabaja un sistema de archivos, se tienen carpetas las cuales son agrupaciones de nodos llamados ZNodes, cada nodo individual es llamado ZNode. En cada ZNode se puede almacenar un máximo de 1MB, ya que ZooKeeper está diseñado para coordinar distintas aplicaciones y no para transmitir datos. El acceso a los datos es atómico, al momento de leer la data, es leída completa. Al momento de realizar alguna escritura, esta es realizada de la misma manera, se sustituye todo el archivo y no solo una parte de este. Para acceder a un nodo específico se utilizan rutas absolutas como si se accediera a un sistema de archivos Unix. Por ejemplo: /grupoA/subgrupo1/nodo1.

ZooKeeper facilita la coordinación entre aplicaciones distribuidas las cuales pueden necesitar en momentos clave que alguna otra aplicación sea ejecutada para poder obtener datos (por ejemplo). La forma más sencilla para poder coordinar distintas aplicaciones es su configuración y ZooKeeper provee una gran ayuda al momento de realizar configuraciones debido a que estas pueden realizarse en grupos e individualmente para cada nodo. Al momento de solicitar realizar alguna acción ZooKeeper se encarga de organizar cuando se ejecutaría dicha acción debido a que pueden existir otros eventos. Para realizar alguna actualización de archivos, esta actualización pasa a través de un servidor maestro el cual decide si dicha actualización puede o no realizarse, la decisión es tomada por un conjunto de servidores maestros más el servidor maestro-líder.

2.4.6. Distribuciones Hadoop

Para facilitar la instalación de un entorno Hadoop y la configuración de este, distintas empresas han creado grandes distribuciones las cuales incluyen una variedad de las herramientas explicadas anteriormente, pasa así tener un ecosistema preparado para solo ser configurado y ejecutado. Cada una de estas distribuciones incluye herramientas del ecosistema Hadoop el cual provee Apache y otras desarrolladas por las empresas para ser totalmente compatibles con el ecosistema. La mayoría de las distribuciones mostradas a continuación están orientadas al sector empresarial, estas tienen un nivel de robustez muy alto.

2.4.6.1. MapR

Esta distribución ofrece distintas soluciones cuando se habla de desarrollo sobre un ecosistema Hadoop. Como bien se sabe, las bases de datos relacionales no escalan del todo bien cuando se habla de grandes volúmenes de datos debido a que estas tienen un mal rendimiento al ingresar estos datos a la base de datos. Pero, MapR ofrece soluciones para trabajar con bases de datos SQL sobre Hadoop, como también ofrece soluciones NoSQL.

Una de las grandes diferencias con las otras 2 grandes distribuciones (las cuales serán explicadas más adelante), es el uso de un sistema de archivos, una base de datos y un sistema de administración del cluster

creados especialmente para esta distribución cuyos nombres son: MapR-FS, MapR-DB y MapR-Control System respectivamente. Excluyendo estas 3 herramientas, MapR sería solo un conjunto de herramientas del ecosistema Hadoop de Apache.

2.4.6.1.1. MapR-FS Es un sistema de archivos distribuido basado en el funcionamiento de HDFS pero mejorando ciertos puntos de fallo que este tiene. A diferencia de HDFS, MapR-FS no es estructurad, no depende de una arquitectura en específico, este diseño busca solventar la problemática del punto de fallo simple que tiene HDFS relacionado con el NameNode.

MapR-FS posee 2 características clave:

- NFS: HDFS provee soporte para utilizar NFS, pero para ser utilizado primero se deben escribir los datos en un sistema de archivos temporal antes de escribir luego en HDFS, lo cual ralentiza el sistema, pueden ocurrir escrituras fuera de rden y se debe de tener un espacio de almacenamiento apartado para realizar estas operaciones. En el caso de MapR-FS, el soporte de NFS es real, se permite la lectura y escritura desde NFS y también hacia MapR-FS
- NameNode: como se mencionó anteriormente, el NameNode en HDFS es un punto de fallo crítico, ya que si este cae, todo el sistema de archivos distribuido también caerá. La metadata en MapR-FS es distribuida a través de todos los nodos, este enfoque no depende de un nodo el cual deba realizar el mantenimiento de toda la metadata

2.4.6.1.2. MapR-DB Es una base de datos de calidad empresarial, fue creada basándose en el enfoque NoSQL. Es totalmente compatible con el ecosistema Hadoop, incluye soporte para distintas características como: modelo de datos basado en grandes columnas (al estilo Apache HBase), escalabilidad al estilo de Hadoop, localidad de datos con trabajos MapReduce, alta consistencia de datos, transacciones ACID a nivel de filas, entre otras.

Una de las grandes características que posee MapR-DB es la localidad de datos con trabajos MapReduce. Al momento de ser ejecutado algún trabajo MapReduce esta base de datos al recibir una solicitud, le entrega los datos al solicitante, pero estos datos a utilizar son tomados del mismo nodo desde donde están siendo solicitados, esto para evitar retrasos de procesamiento.

2.4.6.1.3. MapR-Control System Es un sistema para poder controlar el ecosistema de la distribución, es equiparable a Apache Ambari. Provee sistemas de alarmas, una serie de medidores para observar el estado del cluster de una forma rápida. Este también funciona mediante una interfaz web que se comunica mediante un API el cual realiza todas las funcionalidades de esta herramienta.

2.4.6.2. Cloudera

La empresa más grande en lo que respecta a distribuciones de Hadoop, su estrategia de mercado está basada en el entrenamiento y el soporte de aquellos que instalan su distribución, como también proveen

distribuciones pagas. Estos ofrecen una distribución gratuita y 3 opciones para empresas con distintas herramientas incluidas en cada una de las versiones.

Entre las herramientas desarrolladas por Cloudera se tienen:

- Cloudera Impala: un motor de consultas distribuido que funciona en tiempo real
- Hue: es un conjunto de aplicaciones web que permiten la gestión de las distintas herramientas instaladas en un entorno Hadoop. Permite la ejecución, edición y visualización de distintas tareas para múltiples herramientas
- Sentry: añade una mayr seguridad al cluster Hadoop gestionando el acceso a las herramientas que requieran la lectura de datos. Permite aplicar políticas de acceso a las aplicaciones basadas en roles
- Cloudera Search: basado en una herramienta llamada Solr la cual permite consultas orientadas a texto, muy cercanas a tiempo real. Esta herramienta funciona sobre HDFS y HBase

2.4.6.3. Hortonworks

Uno de los mayores colaboradores en lo que respecta al ecosistema Hadoop. Hortonworks provee un ecosistema llamado Hortonworks Data Platform (HDP), actualmente se encuentra en su versión 2.3. Hortonworks basa su estrategia de mercado en tener HDP de forma libre, puede ser descargado e instalado por cualquiera, pero sus ingresos vienen a partir del soporte provisto a aquellos que lo necesiten como también disponen de una serie de cursos de capacitación en el área.

En la última versión de HDP se busca proveer a los desarrolladores una serie de herramientas que facilitan la edición de queries SQL sobre una herramienta incluida en Hortonworks llamada Hive, también incluye un nuevo navegador de archivos para HDFS mediante una interfaz web. También para los operadores del cluster se tienen nuevas formas de configurar los nodos.

Hortonworks utiliza Apache Ambari como administrador del sistema y HDFS como sistema de archivos, entre las herramientas desarrolladas por Hortonworks se tienen:

- Tez: framework diseñado para trabajar sobre YARN, permite la manipulación de datos producidos en lotes
- Slider: framework el cual permite la gestión de recursos en tiempo real, según los requerimientos de las herramientas ejecutadas. Permite administrar de una mejor manera el cluster deibdo a que permite asignar mayor o menor cantidad de recursos según los requerimientos de las herramientas, esto anteriormente no era posible dependiendo solo de YARN
- Phoenix: provee una interfaz SQL sobre HBase para así poder tratar este almacén de datos con un lenguaje muy parecido a SQL. Este realiza una optimización sobre distintas consultas a realizar lo cual provee un mayor rendimiento
- YARN: administrador de recursos de Hadoop (para más información consultar el punto 2.4.3)

2.4.6.4. Live Hadoop

Big Data es una de las áreas que más se encuentra en desarrollo a nivel mundial debido a sus proyecciones de uso y su impacto en el mundo de las TIC en los próximos años. Sin embargo, desde el punto de vista de investigación, consiste en una tecnología muy costosa porque requiere de grandes infraestructuras. Para la validación de algoritmos, o simplemente formación, se puede recurrir a instalaciones de prueba o "SandBox.ªpoyados en máquinas virtuales. El problemas de estas soluciones es que no sirven para efectuar tareas analíticas de envergadura.

El objetivo fundamental del proyecto Live Hadoop fue la creación de una distribución de Hadoop instalada sobre Slax en el formato de LiveCD, que permite levantar una infraestructura de pruebas para Big Data de forma AdHoc o temporal sobre un conjunto de equipos en red, que no necesariamente estén pre-configurados para ello. Esta tecnología permite implementar un cluster Hadoop sobre un conjunto de máquinas que pudieran tener otro uso o fin, como por ejemplo una sala de formación, salas de navegación o laboratorios. Para esto, el sistema funciona enteramente en memoria, con lo cual, no requiere de ningún tipo de instalación en disco.

Este sistema nos permite configurar y poner en marcha un laboratorio de Big Data con nodos reales, y sobre el cual se pueden hacer cálculos analíticos grandes (esto dependiendo de la potencia de los computadores), todo esto sin la necesidad de contar con la costosa tecnología que requiere un cluster para computación de alto rendimiento. Este LiveCD contiene una instalación de Hadoop y una distribución de R con el componente RHadoop, los cuales nos permiten la ejecución de algoritmos analíticos sobre infraestructuras de Big Data. Al igual que el SandBox, el LiveCD está pre-configurado en modo mononodo cuando se ejecuta en un único equipo.

Una de las ventajas más importantes de este sistema operativo es que posee la opción de bootear mediante un servidor PXE el cual es creado en cualquier nodo que se desee (normalmente el nodo maestro), para luego en las opciones de boot de los distintos nodos poder cargar el sistema operativo mediante la red.

2.5. Data Mining

La minería de datos o *Data Mining* es un proceso analítico diseñado para explorar datos (normalmente una gran cantidad de datos) para lograr conseguir patrones consistentes o relaciones sistemáticas entre las distintas variables a analizar para luego validar nuevos hallazgos aplicando los patrones detectados a nuevos conjuntos de datos. La meta final de la minería de datos es crear predicciones [24]. Data Mining también puede ser considerado como el proceso de analizar datos desde diferentes perspectivas y resumir estos datos en información útil [25].

El proceso de la minería de datos de datos consiste de 3 pasos clave:

1. Exploración inicial de los datos
2. Construcción del modelo e identificación de patrones (Clustering)

3. Aplicación del modelo a nuevos datos (Clasificación)

2.5.1. Clustering

El término clusterización o *Clustering* se refiere a una cantidad variada de algoritmos y métodos que son útiles para agrupar objetos que tengan propiedades similares. En otras palabras, clustering es la herramienta de análisis exploratorio de los datos que apunta a organizar distintos objetos en grupos en una forma tal que el grado de asociación entre 2 objetos es máximo si estos pertenecen al mismo grupo y mínimo en el caso contrario [24].

2.5.1.1. Fuzzy K-Means

El algoritmo K-Means es un algoritmo de Clustering basado en distancias que particiona los datos de entrada en una cantidad determinada de grupos, la cantidad de grupos es asignada al inicio del algoritmo. Los algoritmos basados en distancias son dependientes de una métrica de distancia para poder medir la similitud entre distintos datos [26]. Es importante acotar que los grupos o cluster generados con K-Means pueden ser llamados grupos fuertes, esto debido a que un elemento solo pertenece a un único grupo, como también existe el caso de los grupos débiles que sus elementos pueden pertenecer a uno o más grupos.

En el caso de Fuzzy K-Means, es un algoritmo de Clustering al igual que K-Means el cual está basado en los grupos débiles. La forma como trabaja el algoritmo de K-Means es la siguiente:

1. Escoger centros de los K-Grupos al azar
2. Medir distancias de los objetos con respecto a los centros
3. Mover centros hacia los grupos calculando las distancias
4. Repetir pasos 2 y 3 hasta que los centros no cambien

Los pasos para poder crear un algoritmo Fuzzy K-Means son los siguientes:

Algorithm 1 Fuzzy K-Means, extraído de [27]

Hacer nuevos centros al azar $m_1, m_2, ..., m_k$;
while Existan cambios en los centros **do**
  Encontrar el grado de membresía
  $a(j,i) \leftarrow exp(-||x_j - m_i||^2)$
  $u(j,i) \leftarrow a(j,i)/sum_j a(j,i)$
  for i = 1 **to** k **do**
    $v1 \leftarrow sum(u(j,i)^2 * x_j)$
    $v2 \leftarrow sum(u(j,i)^2)$
    $m_i \leftarrow v1/v2$
  end for
end while

2.5.2. Clasificación

Al hablar de *Clasificación* en lo que respecta a Data Mining es una función que asigna objetos en una colección a una categoría o clase en específico, la meta de la clasificación es la predicción precisa de cada elemento a clasificar [26], normalmente esta categoría es calculada en el paso de Clustering.

2.5.3. Dynamic Time Warping

Antes de hablar de Dynamic Time Warping es necesario saber que es una serie de tiempo. Una *serie de tiempo* es una secuencia de valores medidos en momentos específicos los cuales son ordenados cronológicamente.

Cuando se habla de análisis de series de tiempo existe un algoritmo llamado *Dynamic Time Warping* o DTW, el cual es utilizado para medir la similitud entre 2 secuencias las cuales pueden variar en el tiempo o en su velocidad, este algoritmo calcula la concordancia óptima entre 2 series de tiempo. En general, las series de tiempo son transformadas de forma no-lineal en el tiempo para determinar una medida la cual sirva para obtener la similitud entre estas 2 series [28].

Capítulo 3

Método de Desarrollo

Para llegar a la solución de un problema primero se debe de seguir un proceso, en este contexto está el desarrollo de software como el proceso y el producto de software como la solución. El proceso para el desarrollo de software es un conjunto de actividades en una forma deliberada, estructurada y metódica, requiriendo cada etapa del desarrollo desde la formación de la idea hasta la entrega del producto final. El proceso de creación de software puede llegar a ser muy complejo, por esta razón se debe utilizar un método de desarrollo que implica identificar los requerimientos, planificar, diseñar, documentar, implementar y probar el software.

Es importante acotar que aunque este trabajo no desarrolla un producto de software en si, el proceso de creación de estas metodologías puede fácilmente extrapolarse o adaptarse al desarrollo de un sistema operativo o en el caso de este trabajo a la expansión de la distribución Live Hadoop.

Específicamente para este trabajo de investigación se decidió utilizar una metodología Ad Hoc donde se define una serie de iteraciones y donde cada una de ellas representa un incremento en el desarrollo de la distribución Live Hadoop.

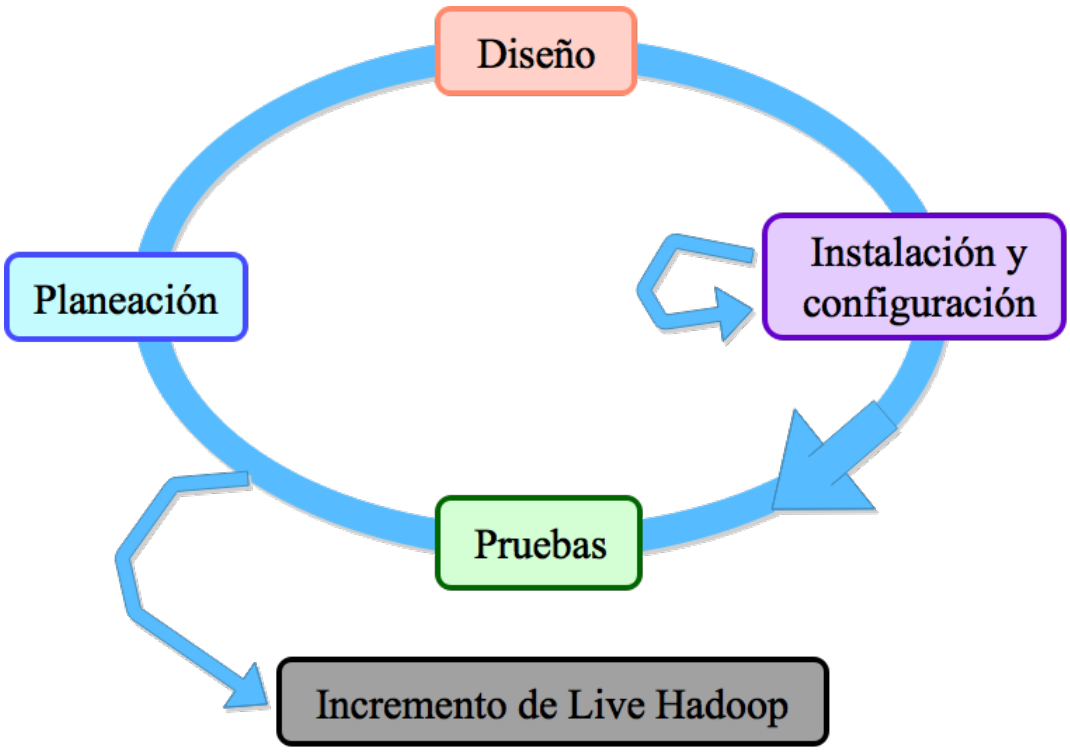


Figura 3.1: Ciclo de desarrollo de la distribución Live Hadoop

En la figura 3.1 se presenta el ciclo de desarrollo de la distribución Live Hadoop. Este ciclo esta compuesto de varias fases descritas a continuación:

- Para la **planeación** se definió cual seria la herramienta a integrar, el tiempo que podría tomar realizar esta tarea de integración y finalmente se acordaron los criterios para las pruebas de aceptación. Las pruebas de aceptación se realizaron con los profesores Jesús Lares y José Sosa.
- Con el **diseño** se definieron estándares para la instalación y configuración.
- Seguidamente en la **instalación y configuración** se realizaba el proceso de instalación e integración de la herramienta a Live Hadoop. Se definieron unos pasos de instalación que se cumplían con la mayoría de las herramientas:

  1. Descarga del software
  2. Descarga y/o actualización de dependencias
  3. Descompresión/Instalación del software
  4. Ubicación del software (mover a carpeta /opt)
  5. Configurar software (archivos por defecto/crear configuración)
  6. Ejecución y prueba del software
  7. Empaquetamiento/Modularización del software

- Finalmente para las **pruebas**, se realizaron pruebas unitarias y de aceptación con los profesores Jesús Lares y José Sosa para comprobar el correcto funcionamiento de la herramienta.

Capítulo 4

Desarrollo de la Solución

Para el desarrollo de la solución se uso una metodología de Ad Hoc descrita en el capitulo 3 donde se siguió un modelo de iteraciones, cada iteración represento una actividad hablada y aprobada por los profesores Jesús Lares y José R. Sosa. Las iteraciones de la metodología de desarrollo se definieron en una hoja de calculo y puede verse en el anexo .7.

4.1. Cluster

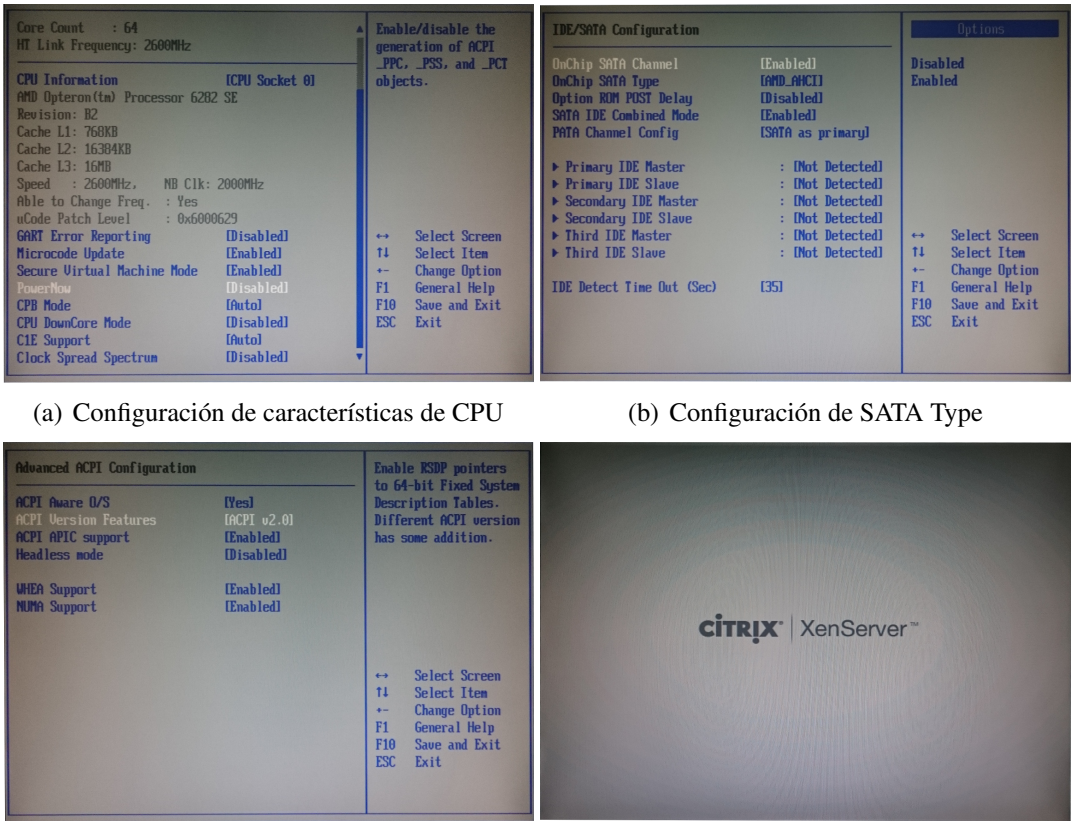
Para el desarrollo del trabajo se utilizó un equipo marca *Supermicro* con las especificaciones mostradas en la tabla 4.1.

Componente	Modelo
Tarjeta Madre	Supermicro H8QG6
Procesador	4 x AMD Opteron 6282 SE
Memorias	524288MB, 32 x 16GB
Disco principal	Western Digital 1 x 500GB
Disco secundario	Western Digital 2 x 2TB
Tarjeta Gráfica	2 x NVidia Tesla C2050
Sistema Operativo	XenServer 6.5

Tabla 4.1: Especificaciones del cluster

4.1.1. Instalación

Para poder realizar las pruebas, se utilizó el equipo que se encuentra en la facultad de ciencias de la Universidad Central de Venezuela. Dicho equipo cuenta con las especificaciones mostradas en la tabla 4.1. Para poder convertirlo en un cluster funcional se realizó una instalación de XenServer 6.5 [29] con el fin de virtualizar. Como se puede observar en el grupo de imágenes 4.1 se tuvo que realizar dicha configuración para poder instalar e iniciar exitosamente XenServer, esto debido a que el equipo utilizado



(a) Configuración de características de CPU

(b) Configuración de SATA Type

(c) Configuración de ACPI

(d) Pantalla de Inicio de XenServer

Figura 4.1: Configuraciones de BIOS para XenServer

no se encontraba en la lista de hardware compatible con XenServer. Sin realizar esta configuración al intentar iniciar XenServer este muestra una pantalla de error lo cual no permite su utilización.

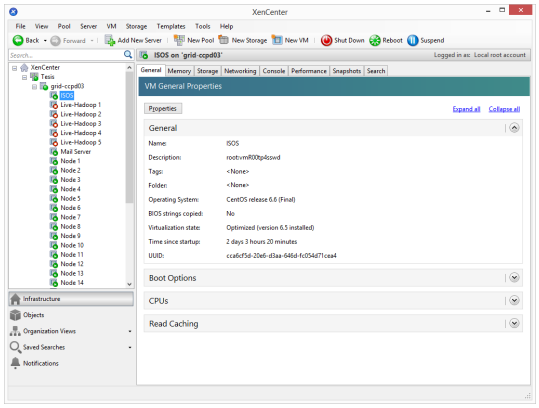


Figura 4.2: Interfaz de XenCenter

Para poder administrar el cluster, XenServer provee una herramienta llamada XenCenter la cual funciona sobre Windows XP - 8.1, su interfaz puede ser vista en la figura 4.2. La herramienta facilita al administrador del cluster la realización de sus tareas, como puede ser crear, pausar y detener maquinas virtuales, modificar plantillas para maquinas virtuales pre-configuradas, crear un cluster de XenServer lo

que permite tener un servicio de alta disponibilidad, entre otros.

4.1.2. Configuraciones

Para realizar pruebas de funcionamiento en el cluster se crearon un conjunto de 15 nodos en los cuales se instaló la distribución Hortonworks siguiendo los pasos explicados en el anexo .1, para más información de las características de los nodos observar la tabla 4.2.

Componente	Modelo
Procesador	3vCPU
Memoria	24GB
Disco duro	100GB
Sistema Operativo	CentOS 6.6

Tabla 4.2: Especificaciones de los nodos de Hortonworks

Luego de realizar una instalación exitosa de Hortonworks se aprovechó este pequeño cluster para la generación de bitácoras en el caso de prueba mostrado en el punto 4.3. Se tiene una máquina virtual la cual funciona como almacenamiento se archivos para el momento de instalar nuevas maquinas virtuales y a su vez como servidor DHCP para los distintos nodos, este servidor fue utilizado para poder asignarle direcciones IP a los distintos nodos que se encuentran en el cluster. Se tiene otra maquina creada con el fin de ser utilizada como servidor de correos para el caso de prueba explicado más adelante.

Para configurar el servidor DHCP se siguieron los siguientes pasos:

1. Ejecutar *yum install dhcp*
2. En */etc/sysconfig/dhcpd* añadir la siguiente línea:
   2.1. *DHCPDARGS=eth0*
3. En */etc/dhcp/dhcpd.conf* añadir las siguientes líneas:
   3.1. *option domain-name “hdp.cluster”;*
   3.2. *option domain-name-servers 8.8.8.8 8.8.4.4;*
   3.3. *default-lease-time 600;*
   3.4. *max-lease-time 7200;*
   3.5. *authoritative;*
   3.6. *subnet 192.168.1.0 netmask 255.255.255.0 {*
   3.7. *option routers 192.168.1.1;*
   3.8. *option subnet-mask 255.255.255.0;*
   3.9. *range 192.168.1.133 192.168.1.164*
   3.10. *}*

4. Iniciar el servidor utilizando */etc/init.d/dhcpd start*

Para realizar las pruebas al sistema operativo Live Hadoop se reservaron 5 nodos en XenServer, estos nodos siguen las características indicadas en la tabla a continuación:

Componente	Modelo
Procesador	3vCPU
Memoria	24GB
Disco duro	60GB
Sistema Operativo	Live Hadoop

Tabla 4.3: Especificaciones de los nodos de Live Hadoop

4.2. Modificaciones realizadas a Live Hadoop

Todos los módulos añadidos a Live Hadoop se encuentran en la ruta */opt* donde pueden ser activados mediante el comando *slax activate Ruta_Absoluta_Al_Módulo*.

El sistema operativo es compilado como un archivo .iso el cual puede ser utilizado para ser montado en un pendrive o grabado en un DVD-ROM, la estructura del archivo .iso es la siguiente:

```
slax
   boot -> Archivos necesarios para que el sistema operativo pueda
      ser iniciado
   changes
   modules -> Se encuentran todos los modulos activados al iniciar
      el sistema operativo
   optional
   rootcopy -> Todos los archivos y carpetas dentro de esta son
      copiados a la raiz del sistema de archivos
      etc -> Archivos de configuracion
         rc.d -> Configuraciones al momento de iniciar el sistema
            operativo
         ssh -> Configuraciones del SSH
      opt -> Modulos creados (no son activados con el sistema
         operativo)
      root -> Configuracion de variables de entorno
      sbin -> Configuracion de PXE (boot por red)
      var
         certs -> Certificados SSL
```

4.2.1. Modificación de rutinas de arranque de Apache Hadoop

Para montar un cluster usando Live Hadoop existen una serie de scripts creados por el profesor Jose R. Sosa que automatizan todo el proceso de configuración e instalación del cluster. Los scripts funcionan muy bien en su mayoría pero tienen ciertas limitaciones que trataron de mitigarse un poco.

Hay un script llamado "prepare-hadoop-multinode.sh" el cual se encarga de hacer toda la configuración básica de Hadoop para un conjunto de nodos, con configuración básica de Hadoop se hace referencia a la configuración necesaria para correr un cluster con Hadoop Common, Hadoop Distributed File System (HDFS), Hadoop Yet Another Resource Negotiator (YARN) y Hadoop MapReduce.

Entre las limitaciones de "prepare-hadoop-multinode.sh" se puede mencionar que siempre que se quería configurar un cluster nuevo se debía modificar el funcionamiento del script indicándole las direcciones IP del cluster, no permite configurar herramientas adicionales del ecosistema de Hadoop y no permite especificar cual sera el maestro del cluster y sus respaldos, es decir especificar NameNode, SecondaryNameNode y ResourceManager (para mas información ver HDFS 2.4.2 y YARN 2.4.3).

Lo que se hizo fue modificar este script para que recibiera un archivo de entrada usando el formato de serialización de datos JSON donde se especifican todos los parámetros de configuración del cluster, este archivo sigue la siguiente estructura:

```
{
  "cluster_ips": [
    "XXX.XXX.XXX.XXX",
    "XXX.XXX.XXX.XXX",
    ...
  ],
  "cluster_modules": {
    "zookeeper": {
      "servers": [
        "XXX.XXX.XXX.XXX",
        "XXX.XXX.XXX.XXX",
        "XXX.XXX.XXX.XXX"
        ...
      ]
    },
    "hbase": {
      "master": "XXX.XXX.XXX.XXX",
      "master_backup": "XXX.XXX.XXX.XXX",
      "region_servers": [
        "XXX.XXX.XXX.XXX",
        "XXX.XXX.XXX.XXX",
        ...
      ]
    },
      "spark": {
        "master": "XXX.XXX.XXX.XXX",
```

```
      "slaves": [
        "XXX.XXX.XXX.XXX",
        "XXX.XXX.XXX.XXX",
        ...
      ]
    }
  }
}
```

Las cadenas de caracteres de la forma "XXX.XXX.XXX.XXX" hacen referencia a una dirección IP. A continuación se explican cada uno de estos parámetros:

- **cluster_ips** hace referencia a todas las direcciones IP del cluster.
- **cluster_modules** indica que módulos se van a activar en el cluster. Actualmente solo se soporta Zookeeper, HBase y Spark:
  - Para el parámetro **zookeeper** se debe especificar un arreglo de **servers** que hace referencia a los nodos que van a formar el servicio de Zookeeper.
  - Con respecto a **hbase** se deben especificar 3 cosas:
    - **master** indica cual va a ser el nodo maestro de HBase.
    - **master_backup** indica cual va a ser el nodo que va a funcionar como maestro de respaldo para HBase.
    - **region_servers** indica cuales van a ser los nodos que van a correr como servidores de regiones de HBase.
  - Finalmente para el parámetro **spark** se deben especificar 2 cosas:
    - **master** indica cual va a ser el nodo maestro de Spark.
    - **slaves** indica cuales van a ser los nodos esclavos de Spark.

Para poder procesar el archivo de entrada se uso como apoyo el comando *jq* el cual es un procesador de archivos JSON de la interfaz de comandos.

Con esto se lograron solventar 2 de 3 limitaciones, pero aun queda una que por la naturaleza de la implementación del script "prepare-hadoop-multinode.sh" no se pudo cambiar y esa es que no permite especificar cual sera el maestro del cluster y sus respaldos, esto trae como consecuencia que el maestro del cluster Hadoop siempre sera el nodo especificado en la primera dirección IP de **cluster_ips** y los maestros de HBase y Spark deberán ser ese mismo nodo.

4.2.1.1. Pruebas

Para probar el funcionamiento de las rutinas de arranque se uso el conjunto de nodos especificado en la tabla 4.3 y se uso el siguiente archivo de configuración:

```
{
  "cluster_ips": [
    "192.168.1.152",
    "192.168.1.153",
    "192.168.1.154",
    "192.168.1.155",
    "192.168.1.156"
  ],
  "cluster_modules": {
    "zookeeper": {
      "servers": [
        "192.168.1.152",
        "192.168.1.153",
        "192.168.1.154"
      ]
    },
    "hbase": {
      "master": "192.168.1.152",
      "master_backup": "192.168.1.153",
      "region_servers": [
        "192.168.1.153",
        "192.168.1.154",
        "192.168.1.155",
        "192.168.1.156"
      ]
    },
      "spark": {
        "master": "192.168.1.152",
        "slaves": [
          "192.168.1.153",
          "192.168.1.154",
          "192.168.1.155",
          "192.168.1.156"
        ]
      }
    }
}
```

Después de correr el script “prepare-hadoop-multinode.sh” con el archivo de configuración mostrado y después de haber iniciado los servicios se obtuvo un cluster con los siguientes nodos:

Nodo	IP	Nombre de dominio
1	192.168.1.152	master
2	192.168.1.153	slave-153
3	192.168.1.154	slave-154
4	192.168.1.155	slave-155
5	192.168.1.156	slave-156

Tabla 4.4: Especificaciones de los nodos de Live Hadoop con sus nombres de dominio

Las características básicas de Hadoop en los nodos:

Nodo	NameNode	SecondaryNameNode	DataNode	ResourceManager	NodeManager
1	Si	Si	Si	Si	Si
2	No	No	Si	No	No
3	No	No	Si	No	No
4	No	No	Si	No	No
5	No	No	Si	No	No

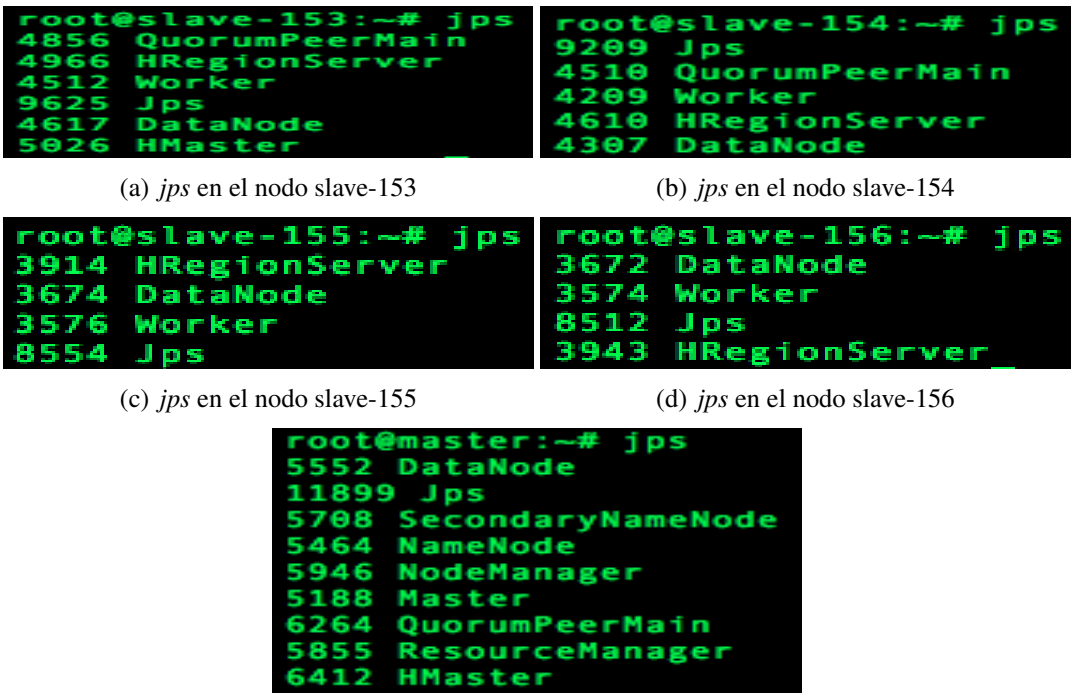
Tabla 4.5: Especificaciones de los nodos de Live Hadoop con las funcionalidades básicas de Hadoop

Las características de los módulos de Zookeeper, HBase y Spark en los nodos:

Nodo	Zookeeper	Maestro Hbase	Servidor de región HBase	Maestro Spark	Esclavo Spark
1	Si	Si	No	Si	No
2	Si	Si	Si	No	Si
3	Si	No	Si	No	Si
4	No	No	Si	No	Si
5	No	No	Si	No	Si

Tabla 4.6: Especificaciones de los nodos de Live Hadoop con los módulos adicionales

En las figuras 4.3(e), 4.3(a), 4.3(b), 4.3(c) y 4.3(d) puede verse el resultado de hacer el comando *jps* en cada uno de los nodos, con esto listamos todos los procesos de Java y confirmamos que se este ejecutando todo como y donde debe ser:



(a) *jps* en el nodo slave-153

(b) *jps* en el nodo slave-154

(c) *jps* en el nodo slave-155

(d) *jps* en el nodo slave-156

(e) *jps* en el nodo master

Figura 4.3: Resultados del comando *jps* en cada nodo

Los procesos que llevan por nombre “NameNode”, “SecondaryNameNode” y “DataNode” son de HDFS. Los procesos “ResourceManager” y “NodeManager” son de YARN. Los procesos “QuorumPeerMain” son de Zookeeper. Los procesos “HMaster” y “HRegionServer” son de HBase. Finalmente los procesos “Master” y “Worker” son de Spark.

Mas adelante se explica detalladamente como se realizo la configuración de los servicios de Zookeeper (4.2.6.1), HBase (4.2.7.1) y Spark (4.2.8.1).

4.2.2. Integración de Elasticsearch

Se incluyó la opción de ejecutar Elasticsearch 1.6.0 el cual fue previamente configurado para poder ser ejecutado en modo mononodo y multinodo.

Para su instalación se procedió a descargar Elasticsearch desde su página web oficial [30], una vez descargada la herramienta es extraída y su archivo de configuración es modificado, este se encuentra en */opt/elasticsearch-1.6.0/config/elasticsearch.yml*. Fueron añadidas las siguientes líneas:

- *cluster.name: forthewatch*
- *path.data: /mnt/cluster/elastic*

En lo que fue añadido al archivo de configuración *cluster.name* hace que todas las instancias de Elasticsearch que se encuentren corriendo en una misma red que tengan el nombre *forthewatch* van a

pertenecer a un cluster de Elasticsearch y *path.data* indica la ruta absoluta donde serán almacenados todos los datos creados por Elasticsearch

Una vez finalizada la configuración, la herramienta fue llevada al formato *.sb*, utilizando el comando *dir2sb* para convertir la herramienta en un módulo, *.sb* es el formato de los módulos utilizados en Slax (según se explicó en el punto 2.2.5.1). Para más información de Elasticsearch se creó una guía la cual se puede encontrar en .2

4.2.2.1. Pruebas

Las pruebas con Elasticsearch fueron realizadas de 2 formas distintas. Una de ellas ejecutando la herramienta de forma independiente */opt/elasticsearch-1.6.0/bin/elasticsearch*. Una vez corriendo la herramienta, se ejecutó el siguiente comando *curl -XPUT 'http://localhost:9200/evilempire/stormtrooper/1' -d '{"name":"evilchewy","join_date":"2015-06-15T15:27:10"}'*, al ejecutar el comando se recibe desde Elasticsearch lo siguiente:

```
{
    "_index":"evilempire",
    "_type":"stormtrooper",
    "_id":"1",
    "_version":1,
    "created":true
}
```

Lo que quiere decir que el comando fue ejecutado exitosamente. Para realizar una prueba adicional al primer comando ejecutado se realizo *curl -XGET 'http://localhost:9200/evilempire/stormtrooper/1?pretty'* lo que retorna toda la información asociada a la entrada creada.

Como segunda prueba se tomó Elasticsearch como almacenamiento de datos para un sistema de detección de anomalías en un servidor, las pruebas realizadas son explicadas más adelante en el punto 4.3.

4.2.3. Integración de Fluentd

Una de las dependencias de Fluentd es un interprete de Ruby por lo que fue necesaria su instalación como módulo. Una vez activado el módulo de Ruby ahora es necesario instalar las Gemas de Ruby las cuales contienen a Fluentd, por lo que se creó un shell script para que sea instalado fácilmente. Al activar el módulo de Fluentd con *slax activate /opt/fluentd.sb* se creará una ruta la cual contiene el script a ejecutar *sh /opt/fluentd/installFluentd.sh* luego de instalar el script, simplemente se debe correr la herramienta escribiendo en consola *fluentd*.

4.2.3.1. Pruebas

Para las pruebas de Fluentd, adelantar al punto 4.3.

4.2.4. Integración de Logstash

Para su instalación se descargó la versión 1.5.0 de Logstash desde su página web oficial [30], una vez descargada la herramienta es extraída para luego ser convertida en un módulo con el comando *dir2sb*. Para más información de Logstash consultar la guía en la sección .3.

4.2.4.1. Pruebas

Para probar Logstash se creó una instancia de Elasticsearch, se hizo la ejecución del script el cual se mencionó en la sección anterior utilizando *logstash -f /opt/logstash-1.5.0/samplescript.conf* el cual carga todos los archivos de bitácoras que se encuentren en la carpeta */var/log* y los almacena en Elasticsearch con el índice *_logs*. Luego para observar si fueron creados correctamente las entradas se observa el índice de Elasticsearch creado con *curl -XGET http://localhost:9200/logs?pretty* y la herramienta retorna todas las entradas que fueron almacenadas mediante Logstash.

También se puede observar las pruebas realizadas en el sistema de detección de anomalías en el punto 4.3.

4.2.5. Integración de Kibana

Para su instalación se descargó la versión 4.1.0 de Kibana desde su página web oficial [30], una vez descargada la herramienta es extraída y para modificar su archivo de configuración el cual se encuentra en <*ruta_kibana*>*/config/kibana.yml* y fueron añadidas las siguientes líneas al archivo:

```
port: 5601
host: "0.0.0.0"
elasticsearch_url: "http://localhost:9200"
kibana_index: ".darklord"
```

- port: puerto por el cual será servida la aplicación mediante el protocolo HTTP
- host: la dirección IP la cual será asociada al servidor
- elasticsearch_url: Dirección del servidor Elasticsearch
- kibana_index: nombre del índice utilizado por Kibana para almacenar tablas

Ya configurada la herramienta, se colocó en */modules* en formato *.sb* utilizando el comando *dir2sb*.

4.2.5.1. Pruebas

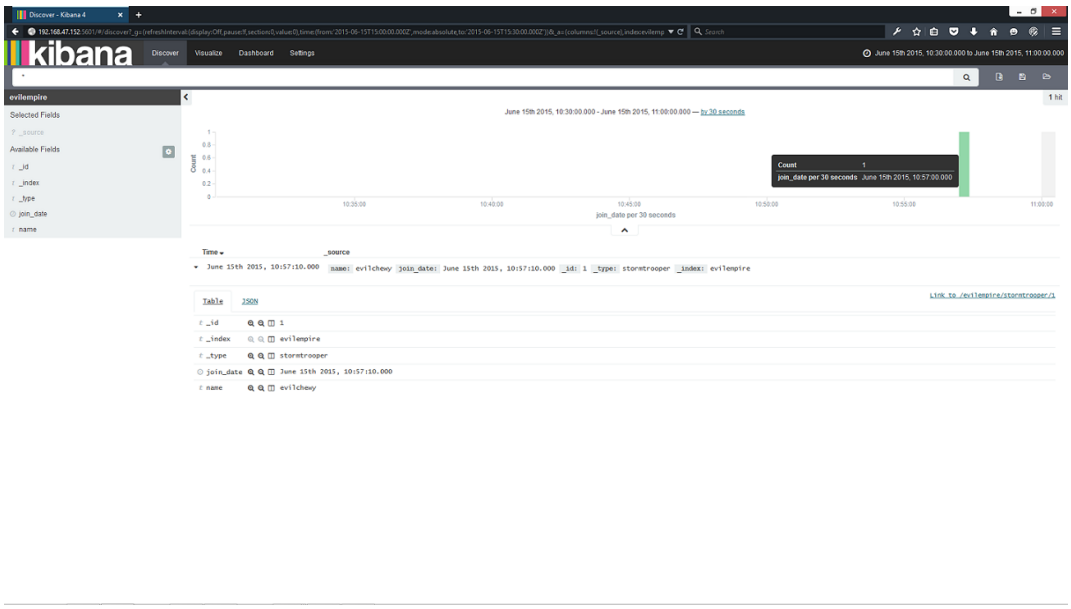


Figura 4.4: Interfaz de Kibana con las pruebas en tiempo real

Como se puede observar en la figura 4.4 se muestran las distintos bitácoras cargadas con Logstash en su respectiva prueba. Para poder ejecutar Kibana */opt/kibana-4.1.0-linux-x64/bin/kibana*, como ya fue previamente configurado, solo es necesario acceder el índice el cual vamos a utilizar que en este caso se utilizó *_logs*.

Otro caso de prueba es demostrado en el punto 4.3 en el cual se utiliza Kibana como una pantalla de monitorización de un servidor.

4.2.6. Integración de ZooKeeper

Para la integración de Zookeeper se descargo la ultima versión disponible (3.1.2) de la pagina oficial [31]. Zookeeper trae distintos scripts para la inicialización y administración, todos disponibles en la carpeta *ZOOKEEPER_PATH/bin*. Después se procedió a volver la herramienta un modulo de Slax con el comando *dir2sb* y se coloco en la carpeta */rootcopy/opt*.

4.2.6.1. Pruebas

Para probar el correcto funcionamiento del modulo se monto un cluster en Live Hadoop con 5 nodos (master, slave-153, slave-154, slave-155 y slave-156) con 3 servidores de Zookeeper (master, slave-153 y slave-154).

El archivo de configuración principal de Zookeeper se llama *zoo.cfg* y se encuentra en la carpeta

ZOOKEEPER_PATH/conf, adicionalmente Zookeeper trae un archivo de configuración de ejemplo el cual es excelente para guiarse, se encuentra en la misma carpeta y lleva por nombre *zoo_sample.cfg*.

Lo que se coloco en el archivo de configuración fue lo siguiente:

```
tickTime=2000
dataDir=/var/zookeeper
clientPort=2181
initLimit=5
syncLimit=2
server.1=master:2888:3888
server.2=slave-153:2888:3888
server.3=slave-154:2888:3888
```

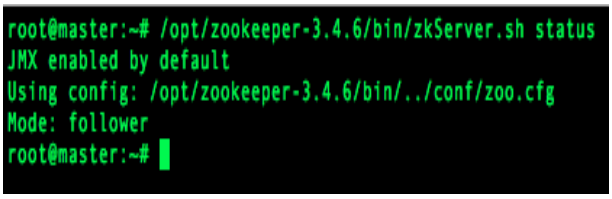
A continuación se explican cada uno de estos parámetros:

- **tickTime** es la unidad de tiempo en milisegundos usada por Zookeeeper. Durante ese instante de tiempo el cliente debe hacer chequeos constantes con el servidor de Zookeeper. El tiempo de expiración de una sesión es el doble de lo especificado en tickTime.
- **dataDir** es el directorio donde Zookeeper mantiene los estados de su base de datos en memoria y al menos de que se especifique lo contrario es el lugar donde guarda las bitácoras de transacciones.
- **clientPort** es el puerto por el cual Zookeeper escucha las conexiones de los clientes.
- **initLimit** es el tiempo limite que tiene el servidor de Zookeeper para conectarse a un líder. Este numero se multiplica por el tickTime para obtener el tiempo en milisegundos.
- **syncLimit** es el tiempo limite que tiene un servidor de Zookeeper para estar desactualizado del líder. Este numero se multiplica por el tickTime para obtener el tiempo en milisegundos.
- **server.X:P1:P2** representa todos los servidores que componen el servicio de Zookeeper. Cuando los servidores inician saben que servidor son revisando un archivo llamado *myid* ubicado en el directorio especificado en dataDir y hace referencia a "X". En lo que refiere a "P1" es el puerto usado para la comunicación entre servidores de Zookeeper y "P2" es el puerto usado para la elección del servidor líder del servicio de Zookeeper.

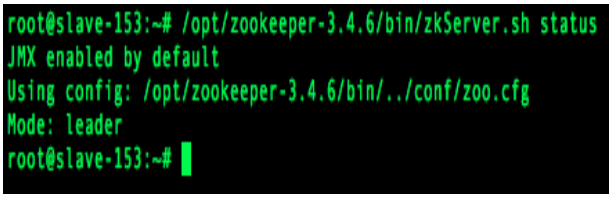
Finalmente en cada uno de los nodos se inicio el servicio de Zookeeper y seguidamente se verifico el estado de cada uno de ellos, para eso se usaron los siguientes comandos:

```
/opt/zookeeper-3.4.6/bin/zkServer.sh start
/opt/zookeeper-3.4.6/bin/zkServer.sh status
```

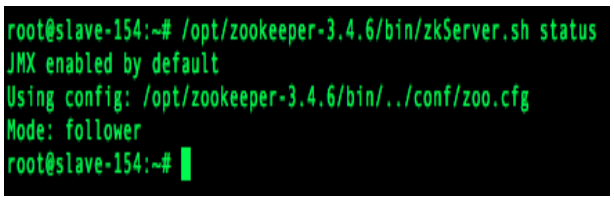
Verificando el estado de los servidores de Zookeeper (ver figuras 4.5(a), 4.5(b), 4.5(c)) se puede observar que el servidor slave-153 es el líder. Mas adelante en la sección 4.2.7.1 se muestran otras pruebas de el servicio de Zookeeper.



(a) El nodo master en modo de seguidor



(b) El nodo slave-153 en modo de líder



(c) El nodo slave-154 en modo de seguidor

Figura 4.5: Resultados del comando para ver el estado de Zookeeper

4.2.7. Integración de HBase

Para la integración de HBase se descargo la ultima versión disponible (1.0.1.1) de la pagina oficial [32]. HBase trae distintos scripts para la inicialización y administración, todos disponibles en la carpeta *HBASE_PATH/bin*. Después se procedió a volver la herramienta un modulo de Slax con el comando *dir2sb* y se coloco en la carpeta */rootcopy/opt*.

4.2.7.1. Pruebas

Para probar el correcto funcionamiento del modulo se monto un cluster en Live Hadoop con 5 nodos (master, slave-153, slave-154, slave-155 y slave-156) con la siguiente configuración de HBase:

Nodo	Maestro	Zookeeper	Servidor de Region
master	Si	Si	No
slave-153	Maestro de respaldo	Si	Si
slave-154	No	Si	Si
slave-155	No	No	Si
slave-156	No	No	Si

Tabla 4.7: Configuracion de prueba de Hbase

De acuerdo con la documentación oficial de HBase [32], hay 3 formas de correr el servicio:

1. De manera local, donde un solo nodo corre como el maestro, servidor de región y los datos se escriben en el sistema de archivos convencional.
2. De manera pseudo distribuida, donde un solo nodo corre como el maestro, servidor de región y los datos se escriben en el HDFS.

3. De manera distribuida, donde un solo nodo corre como el maestro con otros nodos maestro de respaldo, varios nodos como servidores de regiones y los datos se escriben en el HDFS.

La importancia de Zookeeper para Hbase es porque ZooKeeper coordina, comunica y cambia estados entre los maestros y los servidores de regiones. Zookeeper es una parte fundamemntal de HBase. Las operaciones de coordinación son fundamentales como por ejemplo asignación de regiones, fallos en el maestro, replicación, guardar estados.

Para esta prueba se estará usando el servicio de Zookeeper armado en 4.2.6.1. Además debe haber un acceso remoto por SSH entre cada uno de los nodos sin clave.

Todos los archivos de configuracion de HBase se encuentran en el directorio *HBASE_PATH/conf*, los archivos a configurar son: *HBASE_PATH/conf/hbase-site.xml*, *HBASE_PATH/conf/regionservers* y *HBASE_PATH/conf/hbase-env.sh*. Adicionalmente se creara uno nuevo en el mismo directorio llamado *backup-masters*.

El archivo *HBASE_PATH/conf/hbase-site.xml* es el principal archivo de configuración para HBase, a continuación se muestra lo que se coloco:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>

  <property>
    <name>hbase.rootdir</name>
    <value>hdfs://master:54310/hbase</value>
  </property>

  <property>
    <name>hbase.regionserver.port</name>
    <value>16021</value>
  </property>

  <property>
    <name>hbase.cluster.distributed</name>
    <value>true</value>
  </property>

  <property>
     <name>hbase.zookeeper.property.clientPort</name>
     <value>2181</value>
  </property>

  <property>
    <name>hbase.zookeeper.property.dataDir</name>
    <value>/var/zookeeper</value>
  </property>
```

```
  <property>
     <name>hbase.zookeeper.quorum</name>
     <value>master,slave-153,slave-154</value>
  </property>

</configuration>
```

A continuación se explican cada una de estas propiedades:

- **hbase.rootdir** es el directorio donde HBase va a escribir sus datos, se puede colocar un sistema de archivos convencional o HDFS. En este caso se coloco HDFS y el directorio lleva por nombre "hbase" (ver figura 4.8).
- **hbase.regionserver.port** es el puerto por el cual van a escuchar todos los servidores de región, por defecto tiene el valor 16020 pero es importante cambiarlo si algún nodo va a ser servidor de región y maestro al mismo tiempo, ya que ambos intentaran reservar el mismo puerto (16020) y puede ocasionar un error.
- **hbase.cluster.distributed** esta propiedad le indica a Hbase si debe correr de manera distribuida o no.
- **hbase.zookeeper.property.clientPort** es el puerto por el cual Zookeeper acepta conexiones de clientes, debe ser el mismo que se especifique en el archivo de configuración principal de los servidores de Zookeeper (*ZOOKEEPER_PATH/conf/zoo.cfg*).
- **hbase.zookeeper.property.dataDir** hace referencia al directorio donde Zookeeper mantiene los estados de su base de datos en memoria y al menos de que se especifique lo contrario es el lugar donde Zookeeper guarda las bitácoras de transacciones. Debe ser el mismo que se especifica en el archivo de configuración principal de Zookeeper (*ZOOKEEPER_PATH/conf/zoo.cfg*).
- **hbase.zookeeper.quorum** es una lista de nombres de dominio separados por coma que hacen referencia al conjunto de servidores que forman el servicio de Zookeeper.

El archivo *HBASE_PATH/conf/regionservers* indica cuales serán los servidores de región, en forma de nombres de dominio y separados por una linea. Esto fue lo que se coloco:

```
slave-153
slave-154
slave-155
slave-156
```

En el archivo *HBASE_PATH/conf/hbase-env.sh* se colocan todas las variables de entorno de HBase, las variables que no se especifiquen en este archivo tomaran valores por defectos, hay una lista completa de estas variables y posibles valores en la documentación oficial [32]. A continuación se muestra lo que se coloco:

```
export JAVA_HOME=/usr/lib64/jdk7/
export HBASE_OPTS="-XX:+UseConcMarkSweepGC"
export HBASE_MANAGES_ZK=false
```

A continuación se explican cada una de estas variables:

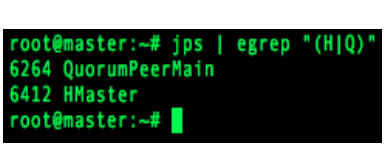
- **JAVA_HOME** hace referencia al directorio donde esta instalado Java.
- **HBASE_OPTS** hace referencia a distintas opciones de Java, "-XX:+UseConcMarkSweepGC" se refiere a cual recolector de basura deberá usarse.
- **HBASE_MANAGES_ZK** indica si el servidor de HBase maneja el servicio de Zookeeper o no. Si lo maneja esto quiere decir que cada vez que se inicie, detenga o reinicie HBase el servicio de Zookeeper hará lo mismo.

Finalmente el archivo *backup-masters* debe crearse para especificar cuales serán los respaldos del servidor maestro de HBase. Se colocan en forma de nombres de dominio y separados por una linea. A continuación se muestra lo que se coloco:

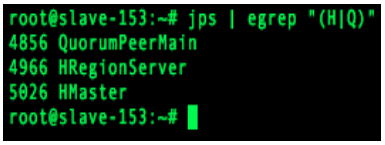
```
slave-153
```

Una vez configurado HBase se procedió a iniciarlo con el comando */opt/hbase-1.0.1.1/bin/start-hbase.sh*.

A continuación en las figuras 4.6(a), 4.6(b), 4.6(c), 4.6(d) y 4.6(e) se muestra la salida del comando *jps | egrep "(H|Q)"*, este comando muestra todos los procesos de Java y aplicamos un filtro con *egrep* para ver solo los procesos que empiecen por "H" (Hbase) y por "Q" (Zookeeper):



(a) El nodo master corre el maestro de HBase y Zookeeper



(b) El nodo slave-153 corre el respaldo del maestro de HBase, un servidor de región y Zookeeper

(c) El nodo slave-154 corre un servidor de región y Zookeeper

(d) El nodo slave-155 corre un servidor de región

(e) El nodo slave-156 corre un servidor de región

Figura 4.6: Resultados del comando *jps | egrep "(H|Q)"*

Adicionalmente se puede realizar una conexión al nodo que corre el maestro de HBase por un explorador web en el puerto 16010 y podemos ver algo similar a lo mostrado por el comando *jps | egrep "(H|Q)"*:

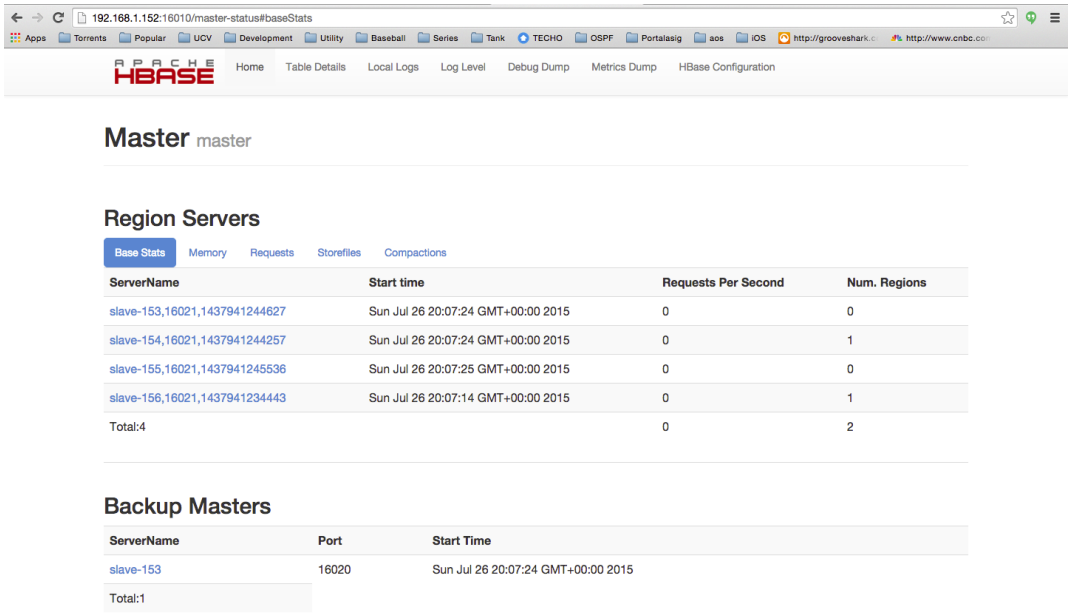


Figura 4.7: Interfaz web del maestro de HBase

A continuación se muestra el resultado de listar directorios en el HDFS y se puede ver el nuevo directorio creado por HBase:

```
root@master:~# /opt/hadoop/bin/hadoop fs -ls /
Found 1 items
drwxr-xr-x   - root supergroup          0 2015-07-26 20:07 /hbase
root@master:~# /opt/hadoop/bin/hadoop fs -ls /hbase
Found 6 items
drwxr-xr-x   - root supergroup          0 2015-07-26 20:07 /hbase/.tmp
drwxr-xr-x   - root supergroup          0 2015-07-26 20:07 /hbase/WALs
drwxr-xr-x   - root supergroup          0 2015-07-26 20:07 /hbase/data
-rw-r--r--   3 root supergroup         42 2015-07-26 20:07 /hbase/hbase.id
-rw-r--r--   3 root supergroup          7 2015-07-26 20:07 /hbase/hbase.version
drwxr-xr-x   - root supergroup          0 2015-07-26 22:18 /hbase/oldWALs
root@master:~#
```

Figura 4.8: Resultados de los comandos */opt/hadoop/bin/hadoop fs -ls /* y */opt/hadoop/bin/hadoop fs -ls /hbase*

También se puede realizar una conexión a un servidor de Zookeeper con el cliente ubicado en */opt/zookeeper-3.4.6/bin/zkCli.sh* y listar los znodes, podemos observar el znode creado por HBase para la coordinación de sus procesos distribuidos:

```
[zk: localhost:2181(CONNECTED) 0] ls /
[hbase, zookeeper]
[zk: localhost:2181(CONNECTED) 1] ls /hbase
[meta-region-server, backup-masters, table, draining, region-in-transition, table-lock, r
unning, master, namespace, hbaseid, online-snapshot, replication, splitWAL, recovering-re
gions, rs, flush-table-proc]
[zk: localhost:2181(CONNECTED) 2]
```

Figura 4.9: Resultados de los comandos *ls /* y *ls /hbase* desde el cliente de Zookeeper

Agregado a todo lo anterior, se realizo una prueba simple para verificar el funcionamiento de HBase

con Zookeeper donde se creo una tabla, se lleno con unos datos, se listaron los datos y después se borro la tabla:

1. Primero se realizo una conexión a la interfaz de comandos de HBase usando el comando */opt/hbase-1.0.1.1/bin/hbase shell*.

2. Seguidamente se creo una nueva tabla, para esto se especifico el nombre de la tabla y el nombre de la familia de columnas a la cual pertenece:

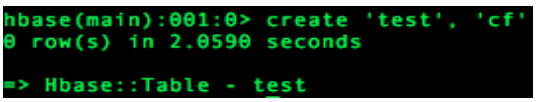


Figura 4.10: Resultado del comando *create 'test', 'cf'*

3. Después se lleno la tabla con algunos datos de prueba usando el comando *put*:

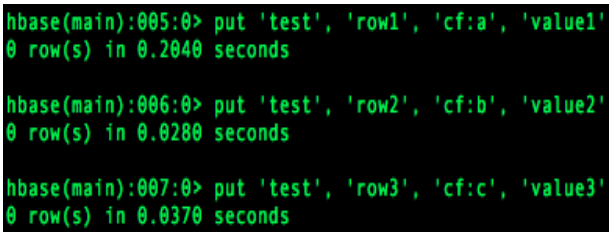


Figura 4.11: Resultados del comando *put*

4. Para listar el contenido de la tabla se uso el comando *scan*:

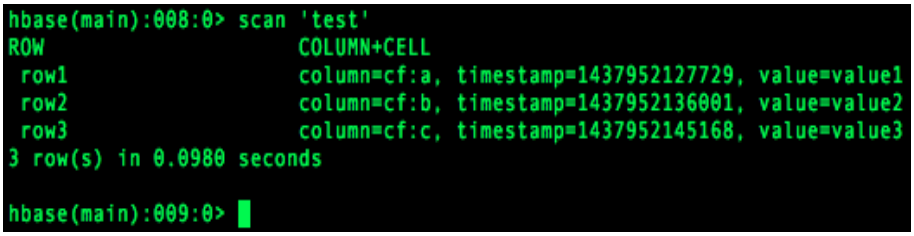


Figura 4.12: Resultado del comando *scan 'test'*

5. Finalmente para borrar la tabla, primero se debe desactivar usando el comando *disable* y después se puede borrar usando el comando *drop*:

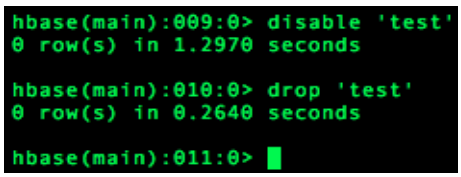


Figura 4.13: Resultado de lo comandos *disable 'test'* y *drop 'test'*

4.2.8. Integración de Spark

Para la integración de Spark se descargo la ultima versión disponible (1.4.0) de la pagina oficial [33]. Spark trae distintos scripts para la inicialización y administración, todos disponibles en las carpetas *SPARK_PATH/bin* y *SPARK_PATH/sbin*. Después se procedió a volver la herramienta un modulo de Slax con el comando *dir2sb* y se coloco en la carpeta */rootcopy/opt*.

4.2.8.1. Pruebas

Para probar el correcto funcionamiento del modulo se monto un cluster en Live Hadoop con 5 nodos (master, slave-153, slave-154, slave-155 y slave-156) donde el nodo con nombre de dominio master sera el maestro de Spark mientras que los nodos restantes son los esclavos.

Para esto solo se tuvo que crear un archivo de configuración especificándole a Spark la variable de entorno JAVA_HOME, ese archivo se encuentra en la capeta *SPARK_PATH/conf* y lleva por nombre *spark-env.sh*. Las variables que no se especifiquen en este archivo tomaran valores por defectos, hay una lista completa de estas variables y posibles valores en la documentación oficial [33]. A continuación se muestra lo que se coloco:

```
export JAVA_HOME=/usr/lib64/jdk7/
```

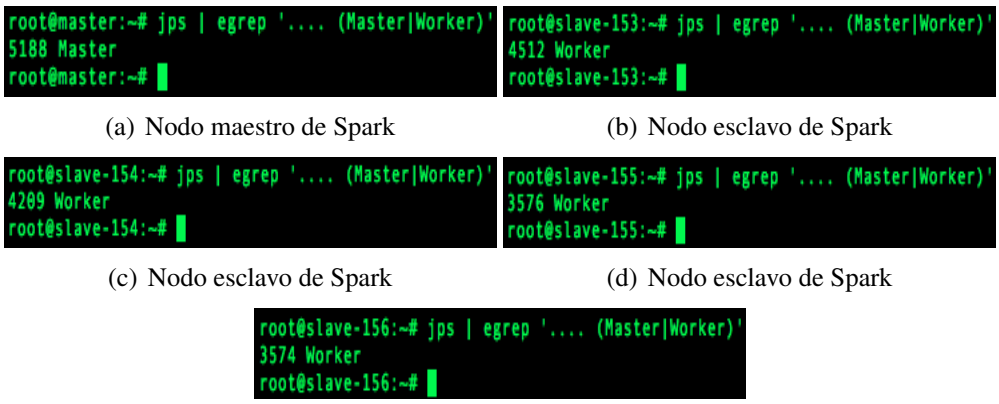
Después se inicio el maestro de Spark en el nodo master con el comando:

```
/opt/spark-1.4.0-bin-hadoop2.4/sbin/start-master.sh
```

Seguidamente se iniciaron los esclavos indicándoles cual es el maestro, con el siguiente comando:

```
/opt/spark-1.4.0-bin-hadoop2.4/sbin/start-slave.sh spark://master:7077
```

A continuación en las figuras 4.14(a), 4.14(b), 4.14(c), 4.14(d) y 4.14(e) se muestra la salida del comando *jps | egrep ".... (Master|Worker)"*. Este comando muestra todos los procesos de Java y aplicamos un filtro con *egrep* para ver solo los procesos que sean "Master" (nodo maestro de Spark) y "Worker" (nodo esclavo de Spark):



(a) Nodo maestro de Spark (b) Nodo esclavo de Spark

(c) Nodo esclavo de Spark (d) Nodo esclavo de Spark

(e) Nodo esclavo de Spark

Figura 4.14: Resultados del comando *jps | egrep ".... (Master|Worker)"*

Como una alternativa tambien se puede ingresar desde un browser al nodo maestro de Spark por el puerto 8080 y se vera algo similar a lo mostrado por el comando *jps | egrep ".... (Master|Worker)"*:

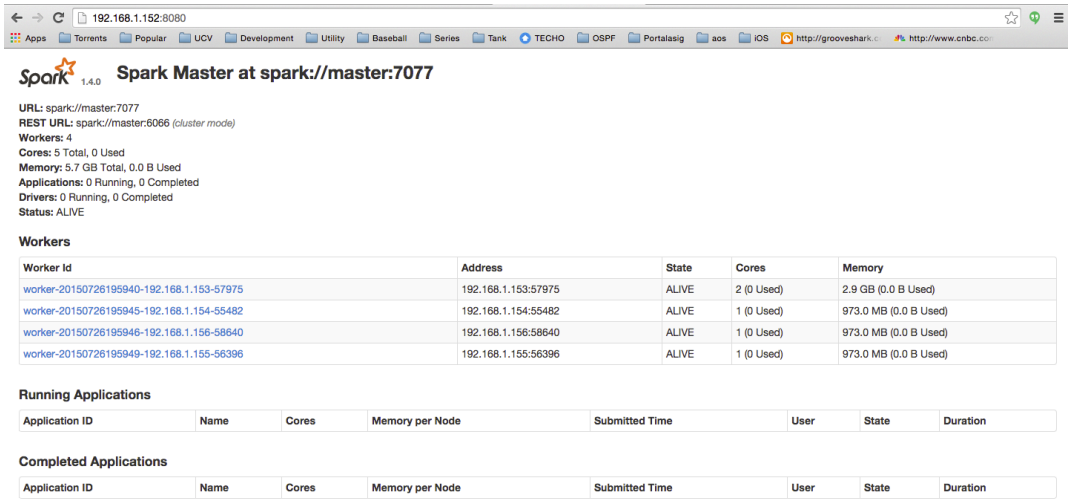


Figura 4.15: Interfaz web del maestro de Spark

Adicionalmente se realizo una pequeña prueba de rendimiento entre MapReduce y Spark. Se corrió el algoritmo de "WordCount" en Java y se uso para contar las palabras de la Biblia en su versión de King James en ingles. Las implementaciones de "WordCount" usadas fueron las tradicionales, la de MapReduce se encuentra en la pagina oficial de Hadoop [13] y la de Spark se puede conseguir en el repositorio publico de git [34].

Las implementaciones de "WordCount" también están disponibles en los anexos .5 y .6.

Para medir el tiempo se uso el comando *time* de UNIX y se tomo el tiempo total desde que se ejecuto el comando hasta que se devolvió el control de la interfaz de comandos al usuario. Para el caso de MapReduce se crearon dos directorios en HDFS, uno llamado "input" donde se encuentra el archivo de la Biblia y uno llamado "output" donde se guardara el resultado del conteo de palabras. Para el caso de

Spark simplemente se corre el algoritmo y se pasa por parámetro el archivo de la Biblia, Spark devuelve el resultado del conteo de palabras por la interfaz de comandos.

Los comandos ejecutados para correr el "WordCount" en MapReduce y Spark fueron:

```
# Para MapReduce
time hadoop jar wc.jar WordCount /input /output >> /dev/null

# Para Spark
time /opt/spark-1.4.0-bin-hadoop2.4/bin/run-example JavaWordCount bible.txt
   >> /dev/null
```

Los resultados son los siguientes:

WordCount	Tiempo en Segundos
MapReduce	9.945s
Spark	4.626s

Tabla 4.8: Tiempos del algoritmo de WordCount contando las palabras de la Biblia

Como se puede observar en la tabla 4.8 el tiempo de Spark fue mucho menor que el de MapReduce, esto gracias a la característica de que no tiene que escribir los resultados en un sistema de archivos como lo es el HDFS y porque corre enteramente en memoria (muchas y otras de estas características son mencionadas en 2.4.5.2.3).

4.2.9. Integración de Mahout

Se incluyó la opción de ejecutar Mahout en su versión más reciente hasta la fecha (0.10.1). Al igual que todas las herramientas mencionadas anteriormente, esta fue convertida en un módulo con el comando *dir2sb*.

4.2.9.1. Pruebas

Para probar Mahout se ejecutaron los scripts que son incluidos en la herramienta al momento de ser descargada. Estos scripts se encuentran en la carpeta */opt/apache-mahout-distribution-0.10.1/examples/bin*. Para ser ejecutados es necesario tener una conexión a Internet debido a que los algoritmos ejecutados necesitan de datos que son descargados al momento de la ejecución de los scripts.

4.2.10. Instalación y actualización de paquetes de R

Se realizaron las instalaciones de los paquetes *elastic* [35] y *curl* en R y fueron actualizados los paquetes *shiny*, *mime* y *jsonlite* para ser utilizados en el caso de prueba explicado en el punto 4.3.

4.3. Integración del ecosistema y caso de prueba

A continuación se demostrará como este pequeño ecosistema se unificó de tal manera que se pudo crear un sistema de detección de anomalías utilizando algunas de las herramientas antes mencionadas.

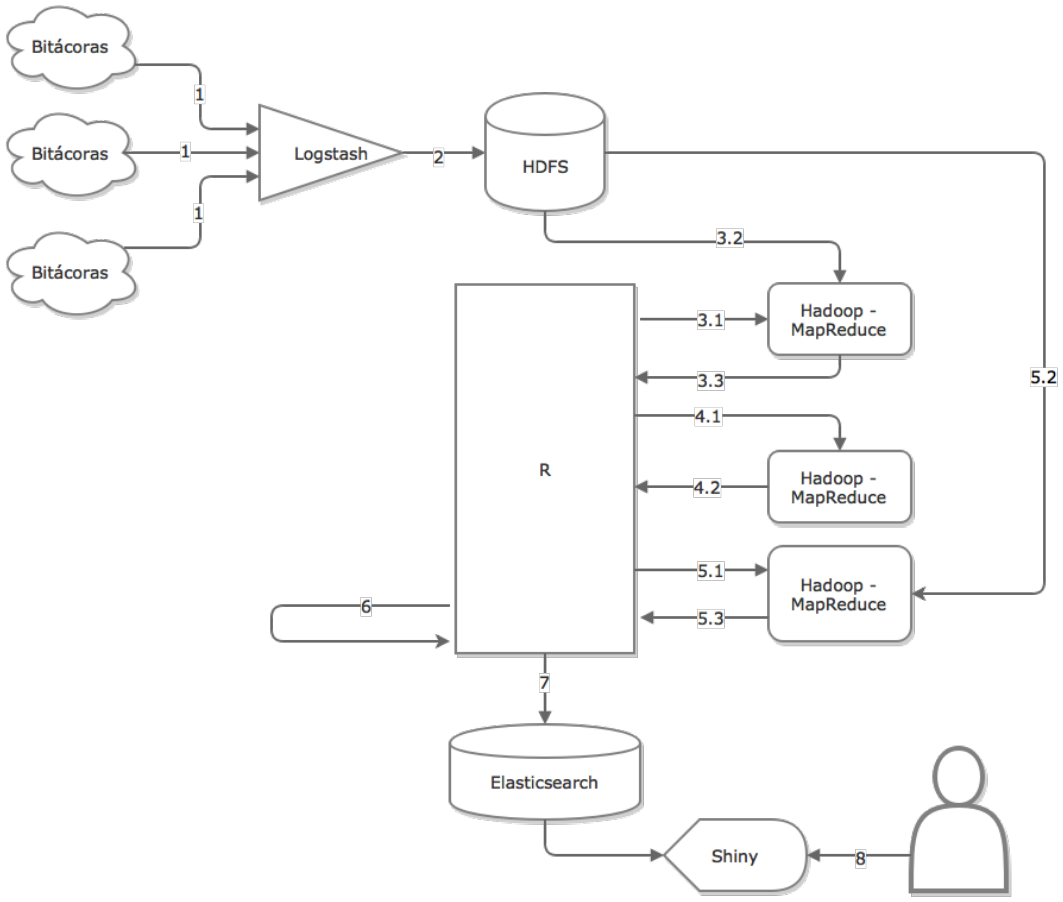


Figura 4.16: Arquitectura de la solución

Como se puede observar en la figura 4.16 los pasos a seguir para poder realizar el caso de prueba fueron los siguientes:

1. Las bitácoras son cargadas con Logstash desde uno o varios archivos generados por el servidor
2. Utilizando la salida de Logstash para escribir en HDFS [36] se cambia el formato de las bitácoras a JSON y estas son almacenadas en el sistema de archivos distribuido
3. Se inicia la creación de series de tiempo a partir de R
    3.1. Ejecución de una tarea MapReduce indicando donde se encuentran los datos en HDFS para obtener la cantidad de visitas por hora en las bitácoras
    3.2. Los archivos de entrada son cargados desde HDFS
    3.3. Al terminar, la tarea MapReduce retorna la cantidad de visitas por hora al servidor

4. Se inicia desde R una tarea en MapReduce para realizar el proceso de Clustering con K-Medias
   4.1. Se indica a MapReduce cuales van a ser los archivos de entrada
   4.2. Se obtiene la entrada retornada por MapReduce de todos los grupos asociados a las bitácoras
5. Se inicia otra tarea MapReduce para crear las series de tiempo a partir de nuevas bitácoras almacenadas por Logstash para poder ser clasificadas
   5.1. Ejecución de una tarea MapReduce indicando donde se encuentran los datos en HDFS para obtener la cantidad de visitas por hora en las bitácoras
   5.2. Los archivos de entrada son cargados desde HDFS
   5.3. Al terminar, la tarea MapReduce retorna la cantidad de visitas por hora al servidor
6. Correr en R el algoritmo de clasificación utilizando el método de distancia euclidiana entre las series de tiempo y los centros calculados en el paso 4
7. Desde R son almacenados en Elasticsearch los grupos encontrados, las series de tiempo y los grupos clasificados
8. Los resultados pueden ser visualizados desde Shiny que obtiene los datos desde Elasticsearch

4.3.1. Bitácoras de acceso

Como no se disponía de datos reales se generaron bitácoras de acceso de tipo Apache. Estas bitácoras fueron generadas con un programa el cual corre en Hadoop MapReduce [37]. Este generador a partir de un archivo de entrada, genera una salida de un archivo por día solicitado.

Se generaron 5 grupos los cuales tienen un comportamiento asignado. Todos los grupos fueron generados a partir de un comportamiento preliminar el cual fue creado para generar los distintos grupos a utilizar, se definió una cantidad de visitas tomado como un comportamiento estable en el tiempo:

Día	Cantidad de Visitas por día (Rango)
Lunes	1400 - 1900
Martes	1500 - 2100
Miércoles	1800 - 2400
Jueves	1100 - 2500
Viernes	1800 - 2600
Sábado	1900 - 2800
Domingo	2300 - 2900

Tabla 4.9: Comportamiento preliminar de logs

Los 5 grupos generados son los siguientes:

- Comportamiento preliminar: como se puede observar en la tabla 4.9, este grupo es generado totalmente al azar con entradas que varían en los rangos mostrados. Los días con mayor cantidad de visitas son los viernes, sábado y domingo. Mientras que el día de menos visitas es el lunes

- Comportamiento ascendente: a partir del comportamiento preliminar se generó un grupo cuyas visitas fueran aumentando en el tiempo. Por cada día que transcurre se multiplica una variable llamada $valor\_actual$ que aumenta entre $valor\_actual$ y $valor\_actual + 0{,}5$ esta variable comienza con un valor de $valor\_actual = 1$

- Comportamiento anómalo: creado a partir del grupo preliminar, tiene un comportamiento el cual se multiplica un día $attackDay$ el cual puede iniciar entre $12 <= attackDay <= 17$ y este puede tener una duración $2 <= attackEnd <= 4$. En el momento que sea $attackDay$ el comportamiento es multiplicado por una variable que fluctúa entre $10 <= valor\_actual <= 14$. Una vez terminados los días de comportamiento anómalo se tendrán $2 <= downTime <= 4$ días los cuales la cantidad de entradas serán reducidas un 95 %

- Comportamiento ascendente y mantenido: grupo el cual tiene un comportamiento similar al ascendente pero a partir del día $10 <= upDay <= 13$ deja de aumentar y es multiplicado por la constante $valor_actual$ sin cambiar su valor

- Comportamiento descendente: Es un comportamiento al contrario del ascendente, se multiplican los valores de entrada por $10 <= aux <= 15$ y estos se van disminuyendo en un rango de $aux - 0{,}5 <= aux$ hasta llegar a $aux = 1$

Dentro de cada uno de los grupos generados, se realizaron 50 corridas con entradas distintas para poder tener datos suficientes para realizar un análisis sobre estos. Cada una de las corridas son simulaciones del mes de mayo del 2015.

4.3.2. Procesos ETLC

El proceso ETLC fue realizado utilizando Logstash, como se puede observar en la figura 4.16 las bitácoras son cargadas con Logstash utilizando el tipo de entrada el cual permite leer archivos.

Inicialmente se había pensado utilizar Logstash, pero este no provee soporte para escritura en HDFS al momento de instalar. Al realizar una investigación más profunda se encontró Fluentd que Fluentd era la herramienta ideal para el proceso ETLC gracias a que soportaba de forma nativa la escritura en HDFS, pero al momento de realizar la escritura sobre HDFS, Fluentd aplicaba un tratamiento a las marcas de tiempo que no entregaba los resultados que se estaban esperando. Esta herramienta cambia las marcas de tiempo y las trata todas con el horario GMT, aquellas que no estén en este tipo de horario son llevadas a esta por lo que entrega resultados fuera de los tiempos esperados. Lo que llevó a realizar una investigación sobre como escribir con en HDFS con Logstash y al descubrir el plugin [36] de escritura en HDFS mediante WebHDFS se pudo proseguir con la creación del sistema para el caso de prueba.

4.3.2.1. Extracción y transformación de los datos

Las bitácoras son extraídas desde una ruta en específico donde son escritas por el servidor web, para ser extraídas se utiliza el plugin de entrada de Logstash el cual permite cargar archivos desde una posición dentro del archivo la cual se especifica al momento de crear el archivo de configuración de

Logstash. Al ser cargadas, estas son transformadas en formato JSON, el formato de cada línea de una bitácora transformada es:

```
{
  "message":"Linea del log leida",
  "@timestamp":"Fecha extraida del log (utilizada como indice)",
  "@version":"1",
  "host":"Nombre del host",
  "clientip":"Direccion IP del cliente",
  "ident":"Identidad del usuario",
  "auth":"Nombre de usuario determinado por autenticacion HTTP",
  "timestamp":"Fecha la cual fue procesada la peticion",
  "verb":"Verbo HTTP",
  "request":"Peticion",
  "httpversion":"Version HTTP utilizada",
  "response":"Respuesta del protocolo",
  "bytes":"Peso de la peticion",
  "referrer":"Pagina desde la cual fue accedida",
  "agent":"User Agent/Navegador utilizado"
}
```

Al ser transformadas las bitácoras en formato JSON, son mucho más sencillas de procesar por otras herramientas.

4.3.2.2. Almacenamiento de los datos

Los datos ya procesados por Logstash fueron almacenados en HDFS utilizando el plugin [36] que permite almacenar en este sistema de archivos utilizando WebHDFS que permite almacenar datos en HDFS utilizando el protocolo HTTP. Una vez almacenados los datos en HDFS se puede continuar con el procesamiento de estos.

4.3.3. Generación de data de entrenamiento

Para poder generar datos que sean útiles para crear los distintos grupos, se creó el siguiente algoritmo en R:

```
for(folder in folder_list){

   #This reads everything in a folder
   log <- do.call("rbind",lapply(file_list, FUN=function(files){
      read.table(paste(folder,"/",files,sep=""))
   }))

   colnames(log) = c('host', 'ident', 'authuser', 'date',
     'time', 'request', 'status', 'bytes', 'refeerer', 'useragent')
```

```
    log$time = as.POSIXct(log$date, format="[%d/%b/%Y:%H")
    reqs = as.data.frame(table(log$time))

    write.table(reqs$Freq, "hour_count", eol = " ",
      col.names = FALSE, row.names = FALSE, append = TRUE)

    write("","hour_count", append = TRUE)

}
```

Este algoritmo permite generar un archivo el cual en cada línea tiene la cantidad de visitas por hora del servidor, es decir, una serie de tiempo donde cada hora es separada por un espacio. Este archivo es creado a partir de las bitácoras generadas en el punto 4.3.1. Como fue explicado anteriormente en el punto 4.3.1, las bitácoras fueron generadas para el mes de mayo del 2015, al utilizar el generador de series de tiempo al final se tendrán 24 valores por día y 744 valores en un mes. Para realizar las pruebas de este algoritmo fue ejecutado computador con las características mostradas en la tabla 4.10.

Componente	Modelo
Tarjeta Madre	AsRock Z77 Extreme 4
Procesador	Intel i5 3570k @ 4.2GHz
Memorias	Corsair Vengeance 4 x 4GB DDR3 @ 1600MHz
Disco principal	SSD Crucial M4 128GB
Disco secundario	2 x 1TB Samsung SpinPoint
Tarjeta Gráfica	AMD Radeon - Gigabyte 7870 GHz Edition
Sistema Operativo	Windows 8.1 Pro

Tabla 4.10: Computador donde fueron realizadas las pruebas de algoritmos clásicos

Tipo	Tiempo Inicial	Tiempo Final	Tiempo Transcurrido	Tamaño de los datos
Preliminar	2015-07-13 00:06:50	2015-07-13 00:08:35	1.756188 mins	825.571 MB
Ascendente	2015-07-12 23:02:18	2015-07-12 23:11:09	8.847815 mins	3.998 GB
Anómalo	2015-07-12 23:13:31	2015-07-12 23:19:18	5.776329 mins	2.577 GB
Ascendente y mantenido	2015-07-12 23:27:48	2015-07-12 23:33:33	5.743006 mins	2.571 GB
Descendente	2015-07-12 23:45:58	2015-07-13 00:03:06	17.1264 mins	7.190 GB

Tabla 4.12: Generación de visitas por hora en un único nodo

Como se puede observar en la tabla 4.12 los tiempos para generar los archivos de visitas por hora pueden ser un poco altos, por lo que se pensó un mecanismo distinto para crear estos archivos los cuales

serán utilizados con los algoritmos de minería de datos. Este nuevo mecanismo (según el punto 1, 2 y 3 de la figura 4.16) fue generado utilizando Fluentd, R y MapReduce. El algoritmo en R creado para generar los archivos que tienen la cuenta de las visitas por día en MapReduce fue el siguiente:

```
Sys.setenv("HADOOP_CMD"="/opt/hadoop/bin/hadoop")
Sys.setenv("HADOOP_STREAMING"=
    "/opt/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.4.0.jar")

map <- function(k,lines) {
  words.list <- strsplit(lines, '\n')
  words.list <- unlist(words.list)
  words.time <- lapply(words.list, FUN=function(x){
   as.POSIXct(unlist(strsplit(x, " "))[4], format="[%d/%b/%Y:H")
  })
  words <- unlist(words.time)
  return(keyval(words, 1))
}

reduce <- function(word, counts) {
  keyval(word, sum(counts))
}

getTimes <- function(input, output=NULL){
  mapreduce(input=input, output=output, input.format="text",
          map=map, reduce=reduce, verbose=T)
}
hdfs.root <- '/user/hduser'
hdfs.data <- file.path(hdfs.root, 'data')
hdfs.out <- file.path(hdfs.root, 'out')
out <- getTimes(hdfs.data, hdfs.out)

results <- from.dfs(out)
results.df <- as.data.frame(results, stringsAsFactors=F)
colnames(results.df) <- c('word', 'count')
head(results.df)
```

Como se puede observar en la tabla 4.14 el tiempo de ejecución utilizando el algoritmo mostrado anteriormente es menor al tiempo de ejecución en un único nodo sin utilizar MapReduce. El único punto a destacar es con el tipo de dato *preliminar* que es procesado en un tiempo mayor a la generación inicial utilizando algoritmos secuenciales, esto debido al overhead producido por MapReduce.

Tipo	Tiempo Inicial	Tiempo Final	Tiempo Transcurrido	Tamaño de los datos
Preliminar	2015-07-21 13:23:45	2015-07-21 13:27:31	3.76667 mins	825.571 MB
Ascendente	2015-07-21 13:29:16	2015-07-21 13:35:01	5.74801 mins	3.998 GB
Anómalo	2015-07-21 13:38:49	2015-07-21 13:43:44	4.916531 mins	2.577 GB
Ascendente y mantenido	2015-07-21 14:04:02	2015-07-21 14:08:48	4.75913 mins	2.571 GB
Descendente	2015-07-21 14:11:26	2015-07-21 14:23:27	12.01638 mins	7.190 GB

Tabla 4.14: Generación de visitas por hora utilizando MapReduce

4.3.4. Implementación de algoritmos de Minería de Datos en MapReduce

En primera instancia se realizaron una serie de pruebas utilizando algoritmos clásicos de minería de datos los cuales fueron ejecutados en el computador mostrado en la tabla 4.10, a continuación se muestran una serie de tablas las cuales muestran la ejecución de 2 algoritmos de clusterización, clusterización jerárquica y K-Medias.

	Tamaño de muestra	Tiempo Inicial	Tiempo Final	Tiempo Transcurrido
1	50	2015-07-03 21:55:53	2015-07-03 21:58:19	2.444176 mins
2	50	2015-07-03 22:02:08	2015-07-03 22:04:41	2.55741 mins
3	250	2015-07-02 00:07:16	2015-07-02 01:24:23	1.285175h
4	250	2015-07-02 12:02:24	2015-07-02 13:26:22	1.399659h

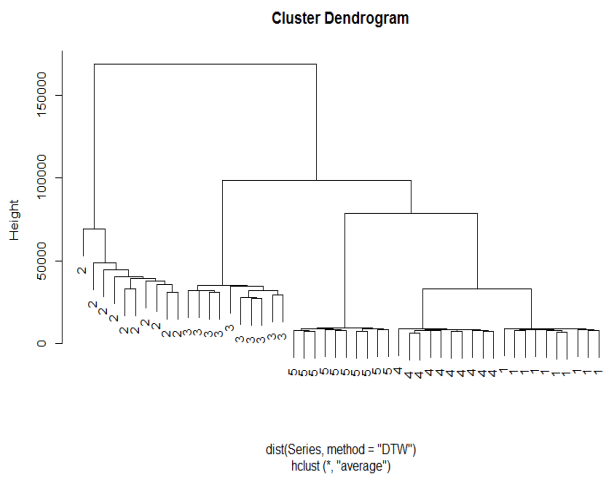
Tabla 4.15: Pruebas de clusterización jerárquica utilizando distancia DTW

En la tabla 4.15 se pueden observar los tiempos del algoritmo de clusterización jerárquica el cual fue ejecutado con una muestra de 50 y 250 bitácoras (10 y 50 por cada tipo de grupo respectivamente). El método de distancias entre elementos utilizado fue DTW. En las figuras 4.17(a) y 4.17(b) se puede observar el resultado de la clusterización jerárquica.

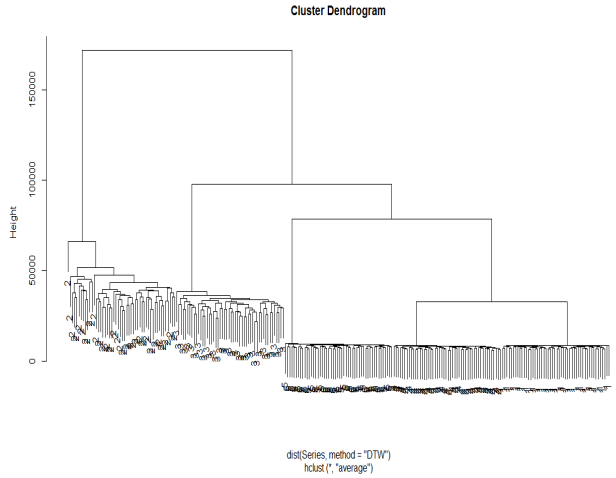
	Tiempo Inicial	Tiempo Final	Tiempo Transcurrido
1	2015-07-03 21:34:02	2015-07-03 21:34:02	0.02301717 seg
2	2015-07-03 21:36:32	2015-07-03 21:36:32	0.02001715 seg
3	2015-07-03 21:40:27	2015-07-03 21:40:27	0.01800895 seg
4	2015-07-03 21:41:09	2015-07-03 21:41:09	0.01901698 seg

Tabla 4.16: Algoritmo de K-Medias ejecutado secuencialmente

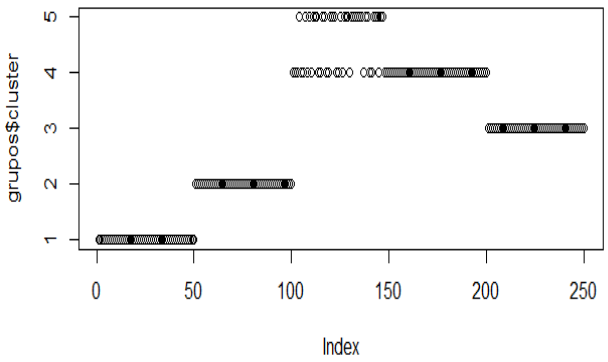
Al igual que con el método de clusterización jerárquica, se realizaron pruebas con el algoritmo de K-Medias, como se puede observar en la tabla 4.16 se realizó una prueba con las 250 bitácoras y dio como resultado un tiempo promedio de 0.02 segundos. Como se puede observar en la figura 4.17(c)



(a) Clusterización jerárquica con una muestra de 50



(b) Clusterización jerárquica con una muestra de 250



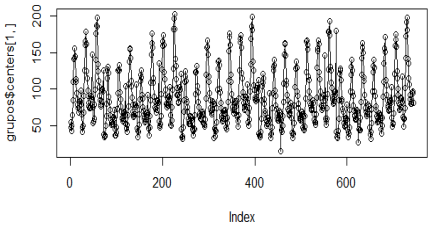
(c) Clusterización KMedias con una muestra de 250

Figura 4.17: Resultados de clusterización

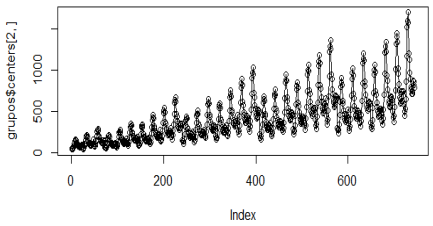
el resultado entregado por KMedias no es tan preciso como el de clusterización jerárquica, esto debido a que en los datos no se hizo una limpieza de los valores atípicos. Como se puede observar en los grupos 4.18(d) y 4.18(e), los resultados anómalos de KMedias son resultado del valor que se encuentra cercano a $Index = 400$ y $grupos\$centers[4,] > 700$ en el grupo 4. Sin embargo, el resultado de KMedias fue bastante bueno comparando la velocidad con cual se generó la clasificación y el resultado final.

Todo esto llevó a tomar una decisión con respecto al algoritmo de minería de datos a utilizar para clusterizar los distintos grupos. Los factores decisivos en el algoritmo de clusterización fueron: tiempo de ejecución y porcentaje de aciertos en los grupos. También hay que tomar en cuenta que si se piensa trabajar con grandes datos, es necesario ofrecer resultados veloces y certeros al momento de clusterizar. Gracias a los tiempos tan bajos de K-Medias en un solo nodo se prosiguió a realizar a una implementación del algoritmo de K-Medias en MapReduce (usando como guía [38]). A continuación el algoritmo de K-Medias en R:

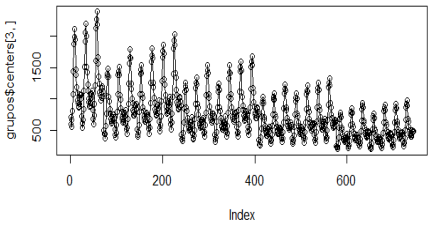
```
#Ejecucion del algoritmo
```



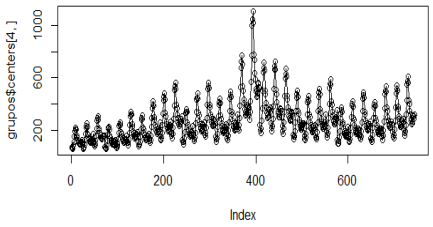
(a) Grupo 1 de centros de KMedias



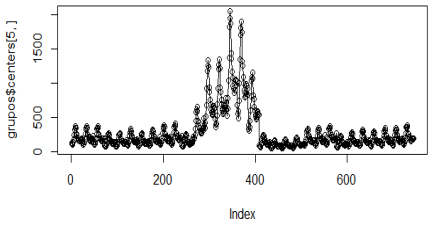
(b) Grupo 2 de centros de KMedias



(c) Grupo 3 de centros de KMedias



(d) Grupo 4 de centros de KMedias



(e) Grupo 5 de centros de KMedias

Figura 4.18: Centros resultantes de KMedias

```
#kmediasMR(to.dfs(P), k = <numero_clusters>,
#    numIter = <Numero_iteraciones>,
#    combine = FALSE, in.memory.combine = FALSE)

kmediasMR = function(P, k, numIter,
    combine, in.memory.combine){

    dist.fun = function(result, P) {
        apply(result,1,function(x){colSums((t(P) - x) ^ 2})
    }

    kmediasMR.map
    kmediasMR.reduce

    result = values(from.dfs(mapreduce(P,
        map = kmediasMR.map, reduce = kmediasMR.reduce)))
```

```
   if(combine || in.memory.combine)
      result = result[, -1]/result[, 1]

   if(nrow(result) < k) {
      result = rbind(result,
         matrix(rnorm((k - nrow(result))* nrow(result)),
            ncol = nrow(result)) %*% result)
   }
   result
}
```

```
#Funcion Mapper

kmediasMR.map = function(., P) {
   nearest = {
      if(is.null(result)){
         sample(1:k, nrow(P), replace = TRUE)
      }else{
         D = dist.fun(result, P)
         nearest = max.col(-D)
      }
   }

   if(!(combine || in.memory.combine)){
      keyval(nearest, P)
   }else{
      keyval(nearest, cbind(1, P))
   }
}

#Funcion Reducer

kmediasMR.reduce = {
   if (!(combine || in.memory.combine)){
      function(., P)
         t(as.matrix(apply(P, 2, mean)))
   }else{
      function(k, P)
         keyval(k,t(as.matrix(apply(P, 2, sum))))
   }
}
```

	Tiempo Inicial	Tiempo Final	Tiempo Transcurrido
1	2015-07-25 17:07:16	2015-07-25 17:08:14	57.18584 seg
2	2015-07-25 17:11:42	2015-07-25 17:12:39	57.22896 seg
3	2015-07-25 17:13:52	2015-07-25 17:14:47	55.48794 seg
4	2015-07-25 17:15:16	2015-07-25 17:16:13	57.06087 seg

Tabla 4.17: K-medias Map Reduce

Como se puede observar en la tabla 4.17 los tiempos al utilizar MapReduce son más altos que la implementación por defecto de R. Tomando en cuenta el trabajo realizado en [38] kmedias puede expandirse mucho más con una mayor cantidad de datos que la que se está utilizando actualmente, toda esta diferencia de tiempo viene dada gracias al overhead causado por utilizar MapReduce, pero al momento de utilizar una mayor cantidad de datos, este tiempo puede llegar a ser poco comparado con la implementación por defecto en R.

4.3.5. Clasificación de nuevas bitácoras

Para poder clasificar las nuevas bitácoras introducidas al servidor fueron probados 2 métodos para medir distancias entre 2 puntos. Una de ellas la distancia DTW (explicada en el punto 2.5.3) y la bien conocida distancia euclidiana. Para probar los algoritmos mencionados anteriormente se hicieron pruebas con un nuevo conjunto de bitácoras previamente clasificadas, se modificaron los valores desde 744 hasta 1 (tomando en cuenta lo mencionado en el punto 4.3.3), cambiando cada posición por un 0 el cual es un valor nulo en las series de tiempo trabajadas, todo esto para saber hasta que momento es un buen clasificador este método.

Existen distintos tipos de anomalías las cuales pueden atacar a un servidor, para este caso de prueba se están tratando los comportamientos que sean anómalos con respecto a un comportamiento normal que fue previamente establecido, se definieron 5 grupos de los cuales 4 son considerados como comportamientos normales y un grupo el cual fue considerado como anómalo o peligroso.

Método	Tiempo de ejecución	Último índice al 100 %	Último índice >= 80 %	Último índice >= 75 %
Distancia euclidiana	7.749629seg	731	683	575
Dynamic Time Warp (DTW)	8.014683h	NA	679	677

Tabla 4.19: Pruebas de tiempo de clasificación

Luego de realizar las pruebas, se puede observar en la tabla 4.19 claramente como el método de distancia euclidiana da resultados mucho más veloces y correctos que DTW, lo que ayudó a tomar la decisión de utilizar el método de distancia euclidiana. Uno de los puntos curiosos de las pruebas realizadas es que DTW es un método que se utiliza para poder calcular eficazmente la distancia entre 2 series de tiempo y es el que entrega peores resultados tanto en tiempo tiempo como en precisión.

Al decidir el método de clasificación se aplicó el mismo mecanismo como fueron generadas las primeras series de tiempo, es decir, sobre un conjunto de datos de bitácoras cargados en HDFS con

Logstash se le aplicó el algoritmo explicado en el punto 4.3.3 una vez obtenidas las series de tiempo, estas son clasificadas tomando la distancia de los centros calculados con K-Medias y una vez obtenido el centro de menor distancia, ese nuevo grupo de datos va a pertenecer a dicho grupo. Esta clasificación viene dada por un número del 1 al 5, uno de los 5 grupos es tomado como grupo anómalo y cada nueva bitácora que pertenezca al grupo anómalo va a ser tratada de forma distinta.

Cada vez que una nueva bitácora sea clasificada como anómala, se mostrará un aviso al usuario mediante una interfaz realizada con el paquete de R llamado Shiny (explicada en el siguiente punto) y además, será enviado un correo al administrador del servidor indicando que existe un comportamiento anómalo y que es necesaria su supervisión.

4.3.6. Interfaz de usuario

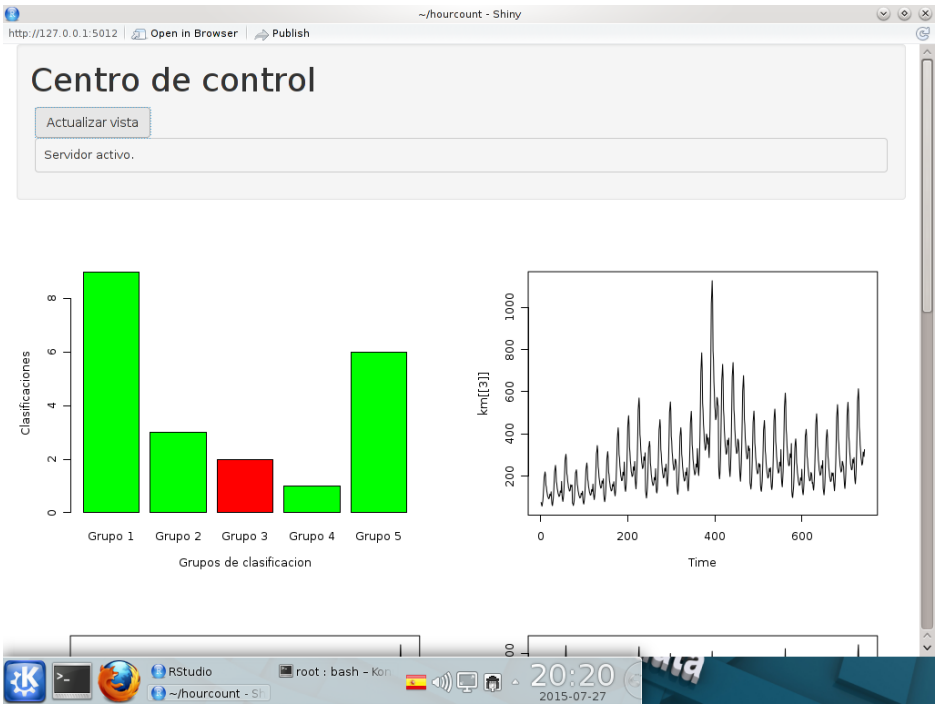


Figura 4.19: Interfaz del centro de control

Para facilitar el trabajo al administrador del servidor en cuestión, fue realizada una interfaz utilizando el paquete Shiny el cual permite la creación de interfaces web (desplegadas usando HTML, CSS y Javascript) utilizando únicamente comandos de R. La interfaz creada permite observar rápidamente el estado actual de las clasificaciones realizadas por los algoritmos explicados previamente. Fue incluido un botón que permite actualizar el gráfico de los grupos de clasificación. Además se tienen 5 gráficas que permiten al usuario observar el comportamiento de cada uno de los grupos creados con el algoritmo de K-Medias.

4.3.6.1. Configuración de servidor de correos y utilización

Para poder realizar alertas al administrador del servidor se realizó la instalación de un servidor de correos (Postfix) en una máquina virtual la cual utiliza CentOS 6.6, a continuación los pasos a seguir para su instalación y configuración:

```
1- Instalacion: Ejecutar el comando yum install postfix
   ca-certificates
2- Configuracion:
   2.1- Editar el archivo /etc/postfix/main.cf y concatenar las
      siguientes lineas:
      relayhost = [smtp.gmail.com]:587
      smtp_use_tls = yes
      smtp_sasl_auth_enable = yes
      smtp_sasl_password_maps = hash:/etc/postfix/sasl_passwd
      smtp_tls_CAfile = /etc/ssl/certs/ca-bundle.crt
      smtp_sasl_security_options =

   2.2- Colocar en el archivo \textit{/etc/postfix/sasl_passwd} la
      siguiente linea:
      [smtp.gmail.com]:587 <nombre_usuario>:<clave>

   2.3- Ejecutar postmap /etc/postfix/sasl_passwd}
   2.4- Ejecutar chown postfix /etc/postfix/sasl_passwd*
   2.5- Finalmente ejecutar /etc/init.d/postfix reload
```

Una vez instalado y configurado el servidor de correos, este puede ser utilizado con el comando *echo "<Mensaje_a_enviar>"| mail -s "<Asunto>" <destinatario>*.

4.3.6.2. Alertas

Como fue explicado en el punto 4.3.5, una vez que una nueva bitácora es clasificada como anómala, un correo es enviado al administrador del servidor. Para enviar un correo desde R se necesita ejecutar un comando del sistema el cual indique al servidor de correos que una alerta debe ser enviada. Al introducir en R las siguientes líneas de código:

```
ip <- "192.168.1.119"
message <- "Por favor revisar el servidor, posible amenaza en proceso."
subject <- "Alerta"
email <- "<destinatario>"
userpasswd <- "123456"
mail <- sprintf('ssh root@%s "echo "%s" | mail -s "%s" %s"', ip, message,
   subject, mail)
system(mail, input=<clave_de_usuario>)
```

Capítulo 5

Conclusiones

Se logró desarrollar todo lo planteado en el alcance del trabajo y como extra se agregaron las herramientas, Mahout, Fluentd, HBase, ZooKeeper, Spark y fueron añadidas modificaciones a las rutinas de arranque de Apache Hadoop que no fueron planteadas desde un inicio.

Logstash y Fluentd sirven para hacer extracción, carga y limpieza de grandes volúmenes de datos. Elasticsearch permite indexar datos en una base de datos orientada a documentos y lo hace de forma distribuida. Kibana provee una visualización de los datos almacenados en Elasticsearch usando una interfaz web, permite hacer consultas y gráficas. Mahout incluye con una amplia biblioteca de algoritmos de minería de datos que pueden usarse desde distintos lenguajes de programación. Hbase permite indexar datos desde el sistema de archivos distribuido de Hadoop en una base de datos orientada a columnas usando distintos procesos que corren de manera distribuida. Zookeeper tiene toda una interfaz que provee funciones de coordinación y administración para procesos distribuidos. Finalmente Spark permite hacer procesamiento de grandes volúmenes de datos, tiene una amplia biblioteca de algoritmos de minería de datos y puede escribir resultados en el sistema de archivos distribuidos o en una base de datos. Las modificaciones de las rutinas de arranque de Apache Hadoop ayudaron en gran parte a facilitar la configuración del cluster y de las herramientas mencionadas anteriormente.

Una vez añadidas todas las herramientas, se desarrolló un sistema el cual permite la detección de anomalías en servidores utilizando series de tiempo extraídas de bitácoras, lo que llevó a crear distintos algoritmos en R utilizando MapReduce para poder cumplir el objetivo del caso de prueba. Facilitando además el trabajo al administrador del servidor donde sea ejecutado el sistema ya que fue creada una interfaz utilizando la librería Shiny, esta interfaz permite visualizar rápidamente el estado de las detecciones que ha realizado el algoritmo.

Para poder realizar pruebas más certeras de las herramientas y del caso de prueba se hizo una instalación de *XenServer* en un pequeño cluster el cual fue prestado por la Facultad de Ciencias de la UCV, esta instalación quedará en la Facultad y podrá ser usada para realizar cualquier tipo de investigaciones.

5.1. Contribución

Este trabajo de investigación contribuye principalmente con la Escuela de Computación de la Facultad de Ciencias de la Universidad Central de Venezuela, ya que esta distribución fue creada inicialmente para ser utilizada por los docentes y alumnos de la escuela. Más allá de la UCV, esta distribución puede ser útil para todo aquel que desee crear un cluster Hadoop sin tener que realizar configuraciones, solo consta de ejecutar 2 scripts y ya se tiene un cluster funcional donde se pueden realizar cálculos.

Otro de los aportes realizados a nivel nacional es el desarrollo en el área de Ciencia de los Datos o Big Data, lo cual en otros países ya se desarrolla de forma natural y muchas empresas basan sus estrategias de mercado en estudios realizados aplicando Ciencia de los Datos como también existen gobiernos los cuales toman decisiones a partir del procesamiento de grandes volúmenes de datos. Anteriormente en Venezuela no había sido realizada una distribución de Linux la cual facilitara el procesamiento de grandes volúmenes de datos.

Finalmente con el sistema desarrollado en el caso de prueba se deja abierto un proyecto el cual puede ser expandido mucho más allá del trabajo realizado, ya que existen distintos tipos de anomalías que se pueden detectar en un servidor y estas pueden ser tratadas de distintas maneras.

5.2. Recomendaciones

Para ejecutar el sistema creado en el caso de prueba es recomendable utilizar R en su versión 3.1.2. Tener el paquete rmr2 el cual permite realizar cálculos sobre Hadoop desde R y Shiny en su versión 0.12.

Es muy recomendable disponer de un cluster para el entorno de pruebas, ya que las pruebas con algoritmos de MapReduce entregan tiempos mucho menores que en computadores caseros.

Para realizar la expansión del sistema operativo y la instalación del cluster es recomendable tener conocimientos de Linux para poder solventar cualquier problema que pueda ocurrir al momento de realizar configuraciones o instalaciones adicionales.

5.3. Trabajos Futuros

Sobre el trabajo realizado existen distintas modificaciones que pueden ser hechas para mejorar aún más la distribución Live Hadoop. Para facilitar mucho más las capacidades de personalizar el cluster, es una buena opción modificar el archivo “prepare-hadoop-multinode.sh” para que permita una configuración mucho más detallada del cluster. Esto incluye también la modificación del archivo JSON creado para la configuración de las IPs, la estructura de este archivo puede ser modificada para añadir una configuración detallada y fácil de leer lo cual es mucho más agradable para los usuarios que deseen configuraciones distintas a la ofrecida hasta el momento. Dentro del ámbito de la configuración también es recomendable realizar una interfaz gráfica la cual permita la configuración de las IPs de una manera mucho más amigable, todo esto antes de ejecutar el archivo “prepare-hadoop-multinode.sh”. Dentro de esta última

recomendación se puede habilitar el uso del comando *nmap* el cual permite escanear toda la red donde se encuentra el nodo donde sea ejecutado el comando y así poder obtener fácilmente todas las IPs que pertenecen a la subred.

Una de las fallas actuales es que las rutinas de arranque de Hadoop configuran los puntos clave de HDFS (Namenode y Secondary Namenode) y YARN (Resource Manager y Node Manager) únicamente en el nodo maestro, por lo que se recomienda crear una configuración la cual permita al usuario decidir en que nodos específicos van a ser colocados estos administradores.

Otra herramienta que facilitaría el uso de la distribución es una interfaz gráfica la cual cree una lista de los módulos disponibles y permita la activación de estos de una forma mucho más amigable para el usuario.

Al momento de crear el archivo .iso para ser utilizado, este contiene todos los módulos añadidos en este trabajo de investigación lo cual ocupa espacio. Puede que algún usuario necesite de este espacio para realizar cálculos por lo que puede prescindir de algunos módulos. Por lo tanto se plantea la creación de un pequeño sistema que permita a los usuarios del sistema operativo crear un archivo .iso adaptado a las necesidades del usuario mediante una interfaz amigable.

Todas las herramientas están siendo ejecutadas por el usuario *root* lo cual puede traer fallos de seguridad y no permite la modularización completa de las herramientas, por lo que se recomienda como trabajo futuro la adaptación de los módulos a usuarios personalizados.

Otros de los trabajos que pueden ser realizados sobre la distribución es la inclusión de nuevas herramientas las cuales faciliten aún más el trabajo de sus usuarios y permita expandir aún más la variedad de problemas que puedan ser resueltos, entre algunas de las herramientas recomendadas se tienen:

- Falcon: Extracción y carga de datos
- Flume: Extracción de archivos de bitácoras y carga de datos
- Hive: Infraestructura que trabaja sobre Hadoop para la consulta y análisis de datos con un lenguaje al estilo SQL
- Knox: Permite administrar el acceso a las distintas herramientas del ecosistema
- Oozie: Planificador de tareas y manejador de tareas en flujos
- Pig: Plataforma para el análisis de grandes volúmenes de datos
- Sqoop: Permite la transferencia de archivos en lotes desde Hadoop hacia almacenes estructurados
- Storm: Motor de procesamiento de datos en tiempo real

Anexos

.1. Guía de instalación Hortonworks

How to install HortonWorks

Using CentOS 6.6 and Ambari 2.0

Without using Internet at all

1. Download all:

   `http://public-repo-1.hortonworks.com/ambari/centos6/2.x/updates/2.0.1/ambari-2.0.1-centos6.tar.gz`
   `http://public-repo-1.hortonworks.com/HDP/centos6/HDP-2.2.4.2-centos6-rpm.tar.gz`
   `http://public-repo-1.hortonworks.com/HDP-UTILS-1.1.0.20/repos/centos6/HDP-UTILS-1.1.0.20-centos6.tar.gz`
   `Oracle JDK 1.7_67 64-bit`
2. Turn off the firewall (all nodes):

   `/etc/init.d/iptables stop`
3. Add nodes to hosts file:

   `/etc/hosts`
4. Create ssh keys on all the nodes:

   1. Create ssh-keygen:

      ```
      ssh-keygen
      ```

   2. add public key to authorized_keys file:

      ```
      cat /root/.ssh/id_rsa.pub >> /root/.ssh/authorized_keys
      ```

```
cat /root/.ssh/id_rsa.pub | ssh root@cluster1.home "cat >>
/root/.ssh/authorized_keys"
```

3. Change permission:

```
chmod 600 ~/.ssh/*
chmod 700 ~/.ssh
restorecon -R -v /root/.ssh
```

4. Deactivate SE-Linux:

```
setenforce 0
```

5. Put main node as CentOS repo:

1. Install Apache server:

```
yum install httpd
```

2. Mount DVD and create symlink to mount directory:

```
mkdir /mnt/CentOS

mount -t iso9660 /dev/dvd /mnt/CentOS

cd /var/www/html/

ln -s /mnt/CentOS/ ./CentOS-Repo
```

3. Modify baseurl from CentOs-Base or create a new one on all nodes:

```
baseurl=http://cluster1.home/CentOS-Repo/
```

6. Copy and Extract Ambari on main node:

```
cd /var/www/html/
```

```
tar -xzf ambari-2.0.1-centos6.tar.gz
```

1. Copy ambari.repo to /etc/yum.repos.d/
2. Change baseurl from [Updates-ambari-2.0.0] to:

```
baseurl=http://<first_node_hostname>/ambari/centos6/2.x/updates/2.0.0
```

3. Disable [ambari-2.x]:

```
enabled=0
```

7. Copy and Extract Hortonworks on main node:

```
cd /var/www/html/
mkdir hdp
cd hdp/

tar -xzf HDP-2.2.4.2-centos6-rpm.tar.gz
tar -xzf HDP-UTILS-1.1.0.20-centos6.tar.gz
```

1. Change baseurl from hdp.repo on both folders to:

```
baseurl=http://<first_node_hostname>/<route_to_folder_on_main_node>
```

8. Install Java (all nodes):

```
mkdir /usr/lib/java_1.7
cd /usr/lib/java_1.7

tar -xzf java_1.7_67.tar.gz
```

9. Install, setup and run Ambari:

1. On setup, you have to setup the java directory

```
yum install ambari-server
```

```
ambari-server setup

ambari-server start
```

10. Connect to ambari with IP:8080 -> credentials= admin:admin

Written by:
Pedro Valdivieso (pedro.valdivieso@gmail.com)
Sebastian Ziegler (sebastian.ziegler.m@gmail.com)

.2. Guía de instalación y uso de Elasticsearch

ElasticSearch

Basic ElasticSearch Concepts

Near Realtime (NRT)

Elasticsearch is a near real time search platform. What this means is there is a slight latency (normally one second) from the time you index a document until the time it becomes searchable.

Cluster

A cluster is a collection of one or more nodes (servers) that together holds your entire data and provides federated indexing and search capabilities across all nodes. A cluster is identified by a unique name which by default is “elasticsearch”. This name is important because a node can only be part of a cluster if the node is set up to join the cluster by its name. It is good practice to explicitly set the cluster name in production, but it is fine to use the default for testing/development purposes.

Note that it is valid and perfectly fine to have a cluster with only a single node in it. Furthermore, you may also have multiple independent clusters each with its own unique cluster name.

Node

A node is a single server that is part of your cluster, stores your data, and participates in the cluster’s indexing and search capabilities. Just like a cluster, a node is identified by a name which by default is a random Marvel character name that is assigned to the node at startup. You can define any node name you want if you do not want the default. This name is important for administration purposes where you want to identify which servers in your network correspond to which nodes in your Elasticsearch cluster.

A node can be configured to join a specific cluster by the cluster name. By default, each node is set up to join a cluster named elasticsearch which means that if you start up a number of nodes on your network and—assuming they can discover each other—they will all automatically form and join a single cluster named elasticsearch.

In a single cluster, you can have as many nodes as you want. Furthermore, if there are no other Elasticsearch nodes currently running on your network, starting a single node will by default form a new single-node cluster named elasticsearch.

Index

An index is a collection of documents that have somewhat similar characteristics. For example, you can have an index for customer data, another index for a product catalog, and yet another index for order data. An index is identified by a name (that must be all lowercase) and this name is used to refer to the index when performing indexing, search, update, and delete operations against the documents in it.

In a single cluster, you can define as many indexes as you want.

Type

Within an index, you can define one or more types. A type is a logical category/partition of your index whose semantics is completely up to you. In general, a type is defined for documents that have a set of common fields. For example, let's assume you run a blogging platform and store all your data in a single index. In this index, you may define a type for user data, another type for blog data, and yet another type for comments data.

Document

A document is a basic unit of information that can be indexed. For example, you can have a document for a single customer, another document for a single product, and yet another for a single order. This document is expressed in JSON (JavaScript Object Notation) which is an ubiquitous internet data interchange format.

Within an index/type, you can store as many documents as you want. Note that although a document physically resides in an index, a document actually must be indexed/assigned to a type inside an index.

Shards & Replicas

An index can potentially store a large amount of data that can exceed the hardware limits of a single node. For example, a single index of a billion documents taking up 1TB of disk space may not fit on the disk of a single node or may be too slow to serve search requests from a single node alone.

To solve this problem, Elasticsearch provides the ability to subdivide your index into multiple pieces called shards. When you create an index, you can simply define the number of shards that you want. Each shard is in itself a fully-functional and independent "index" that can be hosted on any node in the cluster.

Sharding is important for two primary reasons:

It allows you to horizontally split/scale your content volume
It allows you distribute and parallelize operations across shards (potentially on multiple nodes) thus increasing performance/throughput
The mechanics of how a shard is distributed and also how its documents are aggregated back into search requests are completely managed by Elasticsearch and is transparent to you as the user.

In a network/cloud environment where failures can be expected anytime, it is very useful and highly recommended to have a failover mechanism in case a shard/node somehow goes offline or disappears for whatever reason. To this end, Elasticsearch allows you to make one or more copies of your index's shards into what are called replica shards, or replicas for short.

Replication is important for two primary reasons:

It provides high availability in case a shard/node fails. For this reason, it is important to note that a replica shard is never allocated on the same node as the original/primary shard that it was copied from.
It allows you to scale out your search volume/throughput since searches can be executed on all replicas in parallel.
To summarize, each index can be split into multiple shards. An index can also be replicated zero (meaning no replicas) or more times. Once replicated, each index will have primary shards (the original shards that were replicated from) and replica shards (the copies of the primary shards). The number of shards and replicas can be defined per index at the time the index is created. After the index is created, you may change the number of replicas dynamically anytime but you cannot change the number shards after-the-fact.

By default, each index in Elasticsearch is allocated 5 primary shards and 1 replica which means that if you have at least two nodes in your cluster, your index will have 5 primary shards and another 5 replica shards (1 complete replica) for a total of 10 shards per index.

Install ElasticSearch 1.5.2 in a UNIX-like OS

1. You need to download Java JDK (version 7 or later) and have $JAVA_HOME set
2. Next download ElasticSearch from:
   https://www.elastic.co/downloads/elasticsearch
3. Extract it
4. Now you can run /bin/elasticsearch to get started (the default port number is 9200)

ElasticSearch API tutorial

This API tutorial assumes that ElasticSearch is up and running on a host named “testserver.com”.

Basic Information

Lets see some info about the cluster we are running ElasticSeacrh on.

To get the cluster health we can do:

```
curl 'testserver.com:9200/_cat/health?v'
```

The output should be something like this:

```
epoch      timestamp cluster       status node.total node.data shards pri
relo init unassign pending_tasks
1430706023 21:50:23  elasticsearch yellow          1         1     10  10
0    0       10              0
```

We can also get a list of nodes in our cluster as follows:

```
curl 'testserver.com:9200/_cat/nodes?v'
```

And the response:

```
host       ip        heap.percent ram.percent load node.role master name
testserver 127.0.1.1            4          92 0.00 d         *
Tailhook
```

Indexes and Documents

Lets create an index using HTTP PUT method:

```
curl -XPUT 'testserver.com:9200/customer?pretty'
```

You can print all indexes with:

```
curl 'testserver.com:9200/_cat/indices?v'
```

The output should be something like this:

```
health status index    pri rep docs.count docs.deleted store.size
pri.store.size
yellow open   customer   5   1          3            0      7.8kb
7.8kb
```

You can also create an index and a document inside with something like:

```
curl -XPUT 'testserver.com:9200/customer/external/1?pretty' -d '
{
  "name": "John Doe"
}'
```

The document with ID 1 can be query with:

```
curl 'testserver.com:9200/customer/external/1?pretty'
```

You can also create a document and query it using the HTTP POST method:

```
curl -XPOST 'testserver.com:9200/customer/external?pretty' -d '
{
  "name": "Pedro Valdivieso"
}'
```

```
curl 'testserver.com:9200/customer/external/AU0cJ8yBsa3HR3sC8u42?pretty'
```

You can delete an index using:

```
curl -XDELETE 'testserver.com:9200/customer?pretty'
```

When you query all indexes you will not see the one named customer.

The ElasticSearch API follows all of the HTTP standards, meaning that if we do another PUT on the same document it will be overwritten.

But you can also modify or overwrite a document with a POST request:

```
curl -XPOST 'testserver.com:9200/customer/external/1/_update?pretty' -d '
{
  "doc": { "name": "Jane Doe", "age": 20 }
}'
```

You can also make a POST request an pass a script in groovy language to it, as an example this script updates the document:

```
curl -XPOST 'testserver.com:9200/customer/external/1/_update?pretty' -d '
{
  "script" : "ctx._source.age += 5"
}'
```

In the above example, ctx._source refers to the current source document that is about to be updated.

> **Note:** Note that as of this writing, updates can only be performed on a single document at a time. In the future, Elasticsearch might provide the ability to update multiple documents given a query condition (like an SQL UPDATE-WHERE statement).

To delete a document you can:

```
curl -XDELETE 'testserver.com:9200/customer/external/2?pretty'
```

To delete it with a query:

```
curl -XDELETE 'testserver.com:9200/customer/external/_query?pretty' -d '
{
  "query": { "match": { "name": "Pedro" } }
}'
```

Batch processing

You can create an index, type and add documents to it in a single request, like this:

```
curl -XPOST 'testserver.com:9200/customer/external/_bulk?pretty' -d '
{"index":{"_id":"3"}}
{"name": "Pedro" }
{"index":{"_id":"4"}}
{"name": "Sebastian" }
'
```

You can also update and delete:

```
curl -XPOST 'testserver.com:9200/customer/external/_bulk?pretty' -d '
{"update":{"_id":"1"}}
{"doc": { "name": "John Doe becomes Jane Doe" } }
{"delete":{"_id":"2"}}
'
```

Query, filter, count and group by

You can populate ElasticSearch with some data from a file, download a JSON file with some data in it (https://github.com/bly2k/files/blob/master/accounts.zip?raw=true) and load it:

```
curl -XPOST 'testserver.com:9200/bank/account/_bulk?pretty' --data-binary
@accounts.json
```

And you can search using the query string:

```
curl 'http://testserver.com:9200/bank/_search?q=*'
```

Also, you can search using the body of a request with a POST:

```
curl -XPOST 'testserver.com:9200/bank/_search?pretty' -d '
{
  "query": { "match_all": {} },
  "from": 10,
  "size": 10
}'
```

In this case we use the query parameter and tell it to “match_all”, with “from” we tell it to get documents after the ten document and with “size” we tell it to get ten documents in total (so, we end up getting

documents from 11 to 20).

Lets make the same “match_all” query but this time we pass it a “sort” and we tell it to sort by balance in descendent order:

```
curl -XPOST 'testserver.com:9200/bank/_search?pretty' -d '
{
  "query": { "match_all": {} },
  "sort": { "balance": { "order": "desc" } }
}'
```

Search with request body and return only two fields:

```
curl -XPOST 'testserver.com:9200/bank/_search?pretty' -d '
{
  "query": { "match_all": {} },
  "_source": ["account_number", "balance"]
}'
```

Search with ‘match’ instead of ‘match_all’:

```
curl -XPOST 'testserver.com:9200/bank/_search?pretty' -d '
{
  "query": { "match": { "account_number": 20 } }
}'
curl -XPOST 'testserver.com:9200/bank/_search?pretty' -d '
{
  "query": { "match": { "address": "mill" } }
}'
```

Search with “match_phrase”:

```
curl -XPOST 'testserver.com:9200/bank/_search?pretty' -d '
{
  "query": { "match_phrase": { "address": "mill lane" } }
}'
```

Search with ‘bool’ and ‘match’. With the “bool” parameter we can add conditioning to our queries and tell it that the query “must” match, “should” match, “must_not” match, etc. For example:

```
curl -XPOST 'testserver.com:9200/bank/_search?pretty' -d '
{
  "query": {
    "bool": {
      "must": [
        { "match": { "address": "mill" } },
        { "match": { "address": "lane" } }
      ]
    }
  }
}'

curl -XPOST 'testserver.com:9200/bank/_search?pretty' -d '
{
  "query": {
    "bool": {
      "should": [
        { "match": { "address": "mill" } },
        { "match": { "address": "lane" } }
      ]
    }
  }
}'

curl -XPOST 'testserver.com:9200/bank/_search?pretty' -d '
{
  "query": {
    "bool": {
      "must_not": [
        { "match": { "address": "mill" } },
        { "match": { "address": "lane" } }
      ]
    }
  }
}'

curl -XPOST 'testserver.com:9200/bank/_search?pretty' -d '
{
  "query": {
    "bool": {
      "must": [
        { "match": { "age": "40" } }
      ],
      "must_not": [
        { "match": { "state": "ID" } }
```

```
        ]
      }
    }
}'
```

Using filters:

```
curl -XPOST 'testserver.com:9200/bank/_search?pretty' -d '
{
  "query": {
    "filtered": {
      "query": { "match_all": {} },
      "filter": {
        "range": {
          "balance": {
            "gte": 20000,
            "lte": 30000
          }
        }
      }
    }
  }
}'
```

This will match all balance where its value is greater than 20000 and less than 30000. The advantage of using filters above the rest of the methods is that filters can be cached to ElasticSearch internal memory, making queries faster.

Finally, theres aggregations. As an example a count and group by:

```
curl -XPOST 'testserver.com:9200/bank/_search?pretty' -d '
{
  "size": 0,
  "aggs": {
    "group_by_state": {
      "terms": {
        "field": "state"
      }
    }
  }
}'
```

You can also count, group by and get the average balance:

```
curl -XPOST 'testserver.com:9200/bank/_search?pretty' -d '
{
  "size": 0,
  "aggs": {
    "group_by_state": {
      "terms": {
        "field": "state"
      },
      "aggs": {
        "average_balance": {
          "avg": {
            "field": "balance"
          }
        }
      }
    }
  }
}'
```

More information at:
http://www.elastic.co/guide/en/elasticsearch/reference/current/index.html

Written by:
Pedro Valdivieso (pedro.valdivieso@gmail.com)
Sebastian Ziegler (sebastian.ziegler.m@gmail.com)

.3. Guía de instalación y uso de LogStash

How to install LogStash 1.4.2 in CentOS 6.6

---

1. Install Java 1.7.0. It could be Oracle Java or OpenJDK.
2. Installing Logstash
    1. Download Logstash

        ```
        curl -O https://download.elastic.co/logstash/logstash/logstash-1.4.2.tar.gz
        ```

    2. Extracting Logstash

        ```
        tar -zxf logstash-1.4.2.tar.gz
        cd logstash-1.4.2
        ```

3. Running Logstash

    ```
    ./bin/logstash
    ```

Configuring Log formatting

Testing with console input

First we open Logstash and use -e to told the program which formatting is going to being used. This formatting is taken from the console:

```
./bin/logstash -e 'input { stdin { } } output { stdout {} }'
```

Now if you run write anything you'll see the following:

```
For science! You monster...

2015-04-19T17:21:32.245+0000 0.0.0.0 For science! You monster...
```

The interesting thing with Logstash is it capacity to support different input and output formats. Now we'll see some of this formats.

Testing configuration files

Now we're going to create a configuration file to play a little with input/output.

First you've to create a file with the name **test.conf** with the following:

```
input { stdin { } }
output {
    stdout { codec => json }
}
```

Now to tell Logstash to use this config file you should run the following command:

```
./bin/logstash -f test.conf
```

Now if you run the previous command and input something in the console it will print this:

```
something
{"message":"something","@version":"1","@timestamp":"2015-04-
19T17:37:04.332Z","host":"0.0.0.0"}
```

A little bit of theory before continue

Extracted from *http://www.logstash.net*:

- Inputs: are the mechanism for passing log data to Logstash.
- Filters: are used as intermediary processing devices in the Logstash Chain. They're often combined with conditionals in order to perform a certain action on an event, if it matches particular criteria.
- Outputs: are the final phase of the Logstash pipeline. An event may pass through multiple outputs during processing, but once all outputs are complete, the event has finished its execution.
- Codecs: are basically stream filters which can operate as part of an input, or an output. Codecs allow you to easily separate the transport of your messages from the serialization process.

Formatting tests

Now, for example, if we want to get the format from an Apache Log. We only need to create a new config file, we'll name it **apachelog.conf** and we're going to write the following:

```
input { stdin { } }

filter {
  grok {
    match => { "message" => "%{COMBINEDAPACHELOG}"}
  }

  date {
    match => ["timestamp", "dd/MMM/yyyy:HH:mm:ss Z" ]
  }
}

output {
  stdout { codec => json }
}
```

Now we proceed to pass the configuration file to Logstash and once in the input we enter the following line:

```
127.0.0.1 - - [11/Dec/2013:00:01:45 -0800] "GET /xampp/status.php HTTP/1.1"
200 3891 "http://cadenza/xampp/navi.php" "Mozilla/5.0 (Macintosh; Intel Mac
OS X 10.9; rv:25.0) Gecko/20100101 Firefox/25.0"
```

And with this entry we're going to obtain this:

```
{"message":"127.0.0.1 - - [11/Dec/2013:00:01:45 -0800] "GET
```

```
/xampp/status.php HTTP/1.1" 200 3891 "http://cadenza/xampp/navi.php"
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:25.0) Gecko/20100101
Firefox/25.0"","@version":"1","@timestamp":"2011-05-
18T08:48:10.000Z","host":"0.0.0.0","path":"/tmp/access_log.log","type":"apa
che_accesso","clientip":"71.141.244.242","ident":"-
","auth":"kurt","timestamp":"18/May/2011:01:48:10
-0700","verb":"GET","request":"/admin","httpversion":"1.1","response":"301"
,"bytes":"566","referrer":"\"-\"","agent":"\"Mozilla/5.0 (Windows; U;
Windows NT 5.1; en-US; rv:1.9.2.3) Gecko/20100401 Firefox/3.6.3\""}
```

Now with this you know that Logstash can treat Apache logs. But, Logs are not received from stdin but they're created in a file. Finally we're going to open a file.

To do this first we need to create a configuration file, it'll be named **apachefileconfig.conf**:

```
input {
  file {
    path => "/tmp/access_log.log"
    start_position => beginning
  }
}

filter {
  if [path] =~ "access" {
    mutate { replace => { "type" => "apache_access" } }
    grok {
      match => { "message" => "%{COMBINEDAPACHELOG}" }
    }
  }
  date {
    match => ["timestamp", "dd/MMM/yyyy:HH:mm:ss Z"]
  }
}

output {
  stdout { codec => json }
}
```

With the previous code we're going to search the path that has "access" in his name, then we replace the **type** field with **apache_access** and then using **grok** we obtain the relevant fields from an apache log entry.

With this code we're going to access a file named **/tmp/access_log.log**. That file has the following:

```
71.141.244.242 - kurt [18/May/2011:01:48:10 -0700] "GET /admin HTTP/1.1"
301 566 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2.3)
Gecko/20100401 Firefox/3.6.3"
134.39.72.245 - - [18/May/2011:12:40:18 -0700] "GET /favicon.ico HTTP/1.1"
200 1189 "-" "Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1;
Trident/4.0; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR
3.5.30729; InfoPath.2; .NET4.0C; .NET4.0E)"
98.83.179.51 - - [18/May/2011:19:35:08 -0700] "GET /css/main.css HTTP/1.1"
200 1837 "http://www.safesand.com/information.htm" "Mozilla/5.0 (Windows NT
6.0; WOW64; rv:2.0.1) Gecko/20100101 Firefox/4.0.1"
```

Now if we run the new config file it will return:

```
{"message":"71.141.244.242 - kurt [18/May/2011:01:48:10 -0700] \"GET /admin
HTTP/1.1\" 301 566 \"-\" \"Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
rv:1.9.2.3) Gecko/20100401
Firefox/3.6.3\"","@version":"1","@timestamp":"2011-05-
18T08:48:10.000Z","host":"0.0.0.0","path":"/tmp/access_log","type":"apache_
accesso","clientip":"71.141.244.242","ident":"-
","auth":"kurt","timestamp":"18/May/2011:01:48:10
-0700","verb":"GET","request":"/admin","httpversion":"1.1","response":"301"
,"bytes":"566","referrer":"\"-\"","agent":"\"Mozilla/5.0 (Windows; U;
Windows NT 5.1; en-US; rv:1.9.2.3) Gecko/20100401 Firefox/3.6.3\""}

{"message":"134.39.72.245 - - [18/May/2011:12:40:18 -0700] \"GET
/favicon.ico HTTP/1.1\" 200 1189 \"-\" \"Mozilla/4.0 (compatible; MSIE 8.0;
Windows NT 5.1; Trident/4.0; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152;
.NET CLR 3.5.30729; InfoPath.2; .NET4.0C;
.NET4.0E)\"","@version":"1","@timestamp":"2011-05-
18T19:40:18.000Z","host":"0.0.0.0","path":"/tmp/access_log","type":"apache_
accesso","clientip":"134.39.72.245","ident":"-","auth":"-
","timestamp":"18/May/2011:12:40:18
-0700","verb":"GET","request":"/favicon.ico","httpversion":"1.1","response"
:"200","bytes":"1189","referrer":"\"-\"","agent":"\"Mozilla/4.0
(compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; .NET CLR 2.0.50727;
.NET CLR 3.0.4506.2152; .NET CLR 3.5.30729; InfoPath.2; .NET4.0C;
.NET4.0E)\""}

{"message":"98.83.179.51 - - [18/May/2011:19:35:08 -0700] \"GET
/css/main.css HTTP/1.1\" 200 1837
\"http://www.safesand.com/information.htm\" \"Mozilla/5.0 (Windows NT 6.0;
WOW64; rv:2.0.1) Gecko/20100101
Firefox/4.0.1\"","@version":"1","@timestamp":"2011-05-
```

```
19T02:35:08.000Z","host":"0.0.0.0","path":"/tmp/access_log","type":"apache_
accesso","clientip":"98.83.179.51","ident":"-","auth":"-
","timestamp":"18/May/2011:19:35:08
-0700","verb":"GET","request":"/css/main.css","httpversion":"1.1","response
":"200","bytes":"1837","referrer":"\"http://www.safesand.com/information.ht
m\"","agent":"\"Mozilla/5.0 (Windows NT 6.0; WOW64; rv:2.0.1)
Gecko/20100101 Firefox/4.0.1\""}
```


More information at: http://www.logstash.net/docs/1.4.2/

Written by:
Pedro Valdivieso (pedro.valdivieso@gmail.com)
Sebastian Ziegler (sebastian.ziegler.m@gmail.com)

.4. Guía de instalación y uso de R-Studio Server

Installing R Studio Server x64 on a cluster

This tutorial should work on Redhat 5.4+

1. Installing R
    1. First you need to install EPEL repo:

        ```
        yum install epel-release
        ```

    2. Install R

        ```
        yum install R
        ```

2. Installing R Studio Server:
    1. Installing openssl 0.98 (only in RedHat/CentOS 6 and 7):

        ```
        yum install openssl098e
        ```

    2. Download and install R Studio Server:

        ```
        wget http://download2.rstudio.org/rstudio-server-0.98.1103-
        x86_64.rpm
        yum install --nogpgcheck rstudio-server-0.98.1103-x86_64.rpm
        ```

3. Run R Studio Server if it isn't running yet:

    ```
    rstudio-server start
    ```

4. To access to R Studio Server you simply enter to:

```
http://<your_server_ip>:8787
```

To config server edit the following route:

```
/etc/rstudio/rstudio.conf
```

If you want to add more security to your server, you can use apache to add a Reverse Proxy:

```
<VirtualHost *:80>

  <Proxy *>
    Allow from localhost
  </Proxy>

  ProxyPass        / http://localhost:8787/
  ProxyPassReverse / http://localhost:8787/

</VirtualHost>
```

Written by:
Pedro Valdivieso (pedro.valdivieso@gmail.com)
Sebastian Ziegler (sebastian.ziegler.m@gmail.com)

.5. Algoritmo de WordCount en MapReduce

```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

 public static class TokenizerMapper
     extends Mapper<Object, Text, Text, IntWritable>{

  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(Object key, Text value, Context context
              ) throws IOException, InterruptedException {
    StringTokenizer itr = new StringTokenizer(value.toString());
    while (itr.hasMoreTokens()) {
     word.set(itr.nextToken());
     context.write(word, one);
    }
  }
 }

 public static class IntSumReducer
     extends Reducer<Text,IntWritable,Text,IntWritable> {
  private IntWritable result = new IntWritable();

  public void reduce(Text key, Iterable<IntWritable> values,
                Context context
                ) throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable val : values) {
     sum += val.get();
    }
    result.set(sum);
    context.write(key, result);
  }
 }

 public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
```

```
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

.6. Algoritmo de WordCount en Spark

```
package org.apache.spark.examples;

import scala.Tuple2;
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.api.java.function.FlatMapFunction;
import org.apache.spark.api.java.function.Function2;
import org.apache.spark.api.java.function.PairFunction;

import java.util.Arrays;
import java.util.List;
import java.util.regex.Pattern;

public final class JavaWordCount {
  private static final Pattern SPACE = Pattern.compile(" ");

  public static void main(String[] args) throws Exception {

    if (args.length < 1) {
      System.err.println("Usage: JavaWordCount <file>");
      System.exit(1);
    }

    SparkConf sparkConf = new SparkConf().setAppName("JavaWordCount");
    JavaSparkContext ctx = new JavaSparkContext(sparkConf);
    JavaRDD<String> lines = ctx.textFile(args[0], 1);

    JavaRDD<String> words = lines.flatMap(new FlatMapFunction<String,
        String>() {
      @Override
      public Iterable<String> call(String s) {
        return Arrays.asList(SPACE.split(s));
      }
    });

    JavaPairRDD<String, Integer> ones = words.mapToPair(new
        PairFunction<String, String, Integer>() {
      @Override
      public Tuple2<String, Integer> call(String s) {
        return new Tuple2<String, Integer>(s, 1);
      }
    });

    JavaPairRDD<String, Integer> counts = ones.reduceByKey(new
        Function2<Integer, Integer, Integer>() {
      @Override
      public Integer call(Integer i1, Integer i2) {
```

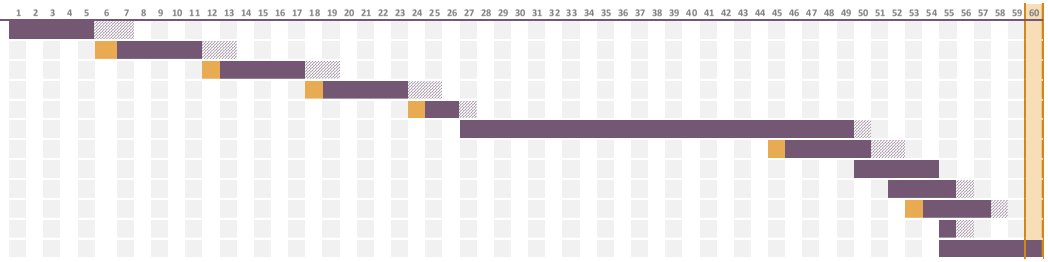
```
      return i1 + i2;
    }
  });

  List<Tuple2<String, Integer>> output = counts.collect();
  for (Tuple2<?,?> tuple : output) {
    System.out.println(tuple._1() + ": " + tuple._2());
  }
  ctx.stop();
 }
}
```

.7. Planificación Ad-Hoc

Expansión de Live Hadoop

Period Highlight 60 | Plan | Actual | % Complete | Actual (beyond plan) | % Complete (beyond plan)

ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	PERCENT COMPLETE
Instalación de prueba Hortonworks	1	7	1	5	100%
Investigación de Elasticsearch	7	7	6	6	100%
Investigación de Logstash	13	7	12	6	100%
Investigación de Kibana	19	7	18	6	100%
Investigación de R y R Studio Server	25	3	24	3	100%
Instalación de cluster XenServer	27	24	27	23	100%
Modificación de rutinas de arranque de Apache Hadoo	46	7	45	6	100%
Integración de Elasticsearch	50	5	50	5	100%
Integración de Logstash	52	5	52	4	100%
Integración de Kibana	54	5	53	5	100%
Instalación y actualización de paquetes de R	55	2	55	1	100%
Creación de sistema de detección de anomalías	55	14	55	14	100%



Glosario

Palabra	Significado
ACID	Atomicidad, Consistencia, Aislamiento (Isolation) y Durabilidad. Propiedades que se deben garantizar en bases de datos transaccionales
BIOS	Basic Input/Output System
Bootloader	Programa que carga un sistema operativo u otro sistema en memoria luego de la carga del BIOS
Cassandra	Base de datos escalable de tipo NoSQL que provee un excelente rendimiento
Cluster	Conjunto de computadoras unidas entre sí por una red y que se comportan como si fuesen una única computadora
Cluster Beowulf	Es un cluster con la característica especial de ser una infraestructura de pruebas de forma AdHoc o temporal, sobre un conjunto de equipos en red, que no necesariamente estén pre-configurados para ello y con características no similares o heterogéneas
CPU	Central Processing Unit/Unidad central de procesamiento. Pieza de hardware que realiza el procesamiento de datos en un dispositivo
CSS	Cascading Style Sheets, lenguaje de marcado para definición de estilos en HTML
DHCP	Dynamic Host Configuration Protocol. Protocolo para asignación automática de IP
ESMTP	Extended SMTP, ver SMTP
Exim	Es un mail transport agent desarrollado en la Universidad de Cambridge para ser usado en sistemas operativos UNIX
FIFO	First In First Out, esquema de colas
FIFO Socket	Es un archivo especial, que abren dos o mas procesos para comunicarse entre ellos
Framework	Base utilizada para desarrollar software, funciona como una capa intermedia que simplifica la utilización de capas inferiores
GMT	Greenwich Mean Time, horario calculado en el Royal Observatory de Londres
HTML	HyperText Markup Language. Lenguaje de marcado para la definición de estructuras y contenido de documentos
HTTP	HyperText Transfer Protocol. Protocolo utilizado en para el intercambio de información en Internet
IDE	Entorno de desarrollo integrado
IMAP	Internet Message Acess Protocol. Protocolo utilizado para la transferencia de correos entre el servidor y el cliente
IP	Internet Protocol. Protocolo utilizado principalmente en la Internet
Javascript	Lenguaje de programación utilizado comúnmente en navegadores web

JDBC	Java DataBase Connector, conector utilizado desde Java para conectarse con bases de datos
JSON	JavaScript Object Notation
Middleware	Capa de abstracción utilizada para simplificar o unificar el uso de componentes debajo de dicha capa
NFS	Network File System, sistema de archivos por red
POP	Post Office Protocol. Protocolo utilizado para la transferencia de correos entre el servidor y el cliente
Qt	Framework multiplataforma para desarrollo de interfaces
RAM	Random Access Memory, memoria de tipo volátil
S3	Amazon Simple Storage Service. Servicios para almacenamiento de Amazon
Script	Un pequeño programa no compilado escrito para un lenguaje de programación o intérprete de comandos
Sendmail	Uno de los primeros mail transport agent desarrollado para UNIX
SMS	Short Message Service, servicios de mensajería de texto
SMTP	Es un estandar en Internet para la transmisión de correos electrónicos
Socket	Es un mecanismo de comunicación entre procesos
Spam	Mensajes no solicitados, correo basura
Standalone	Funciona sin depender de otros componentes (software, hardware)
USB	Universal Serial Bus
Virus informático	Programa que altera el funcionamiento normal de un sistema

Bibliografía

[1] C.-F. J. Wu, "Statistics = data science?"

[2] A. Hidoussi, "Pelicanhpc," http://pelicanhpc.awict.net/, 2014, [Online; Accedido el 25-Julio-2015].

[3] R. M. Abreu y et al., "Journal of cheminformatics," http://www.jcheminf.com/content/2/1/10, 2010, [Online; Accedido el 25-Julio-2015].

[4] E. C. Foster y S. V. Godbole, *Database Systems. A pragmatic approach.*, 1ra. ed. APress, 2014.

[5] A. Silberschatz y et al., *Operating System Concepts*, 9th ed. Wiley, 2013.

[6] A. S. Tanenbaum, *Modern Operating Systems*, 3ra. ed. Pearson, Prentice Hall, 2007.

[7] W. A. Wulf y et al., "Hydra: the kernel of a multiprocessor operating system." en *Communications of the ACM*, 1974, pp. 337–345.

[8] E. Nemeth y et al., *UNIX and Linux System Administration Handbook*, 4th ed. Prentice Hall, 2011.

[9] Y. Inc, "Module 4: Mapreduce," https://developer.yahoo.com/hadoop/tutorial/module4.html, [Online; Accedido el 18-Enero-2015].

[10] T. Matejicek, "Slax," https://www.slax.org/, 2015, [Online; Accedido el 15-Julio-2015].

[11] P. Lougher, "Squashfs," http://squashfs.sourceforge.net/, 2011, [Online; Accedido el 15-Julio-2015].

[12] M. K. McKusick, "Union mounts in 4.4bsd-lite." [En línea]. Disponible en: https://www.usenix.org/legacy/publications/library/proceedings/neworl/full_papers/mckusick.a

[13] A. S. Foundation, "Hadoop," http://hadoop.apache.org/, 2014, [Online; Accedido el 11-Julio-2015].

[14] ——, "Hdfs architecture," http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html, 2015, [Online; Accedido el 14-Julio-2015].

[15] D. Sullivan, "Hadoop architecture," [imagen] Disponible en: http://media.bestofmicro.com/X/8/430172/original/yarn.png, 2014, [Online; Accedido el 27-Diciembre-2014].

[16] M. Zaharia, "Powered by yarn," http://wiki.apache.org/hadoop/PoweredByYarn, 2011, [Online; Accedido el 27-Diciembre-2014].

[17] K. Kambatla y et al., "Arquitectura de yarn," [imagen] Disponible en: http://blog.cloudera.com/wp-content/uploads/2013/11/mr21.png, [Online; Accedido el 27-Diciembre-2014].

[18] ——, "How apache hadoop yarn ha works," http://blog.cloudera.com/blog/2014/05/how-apache-hadoop-yarn-ha-works/, 2014, [Online; Accedido el 27-Diciembre-2014].

[19] J. Dean y S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," pp. 1–13, 2004.

[20] Y. Inc., "Mapping list," [imagen] Disponible en: https://farm3.static.flickr.com/2151/3529146569_f80eb39a44_o.png, [Online; Accedido el 18-Enero-2015].

[21] ——, "Reducing list," [imagen] Disponible en: https://farm4.static.flickr.com/3213/3529959720_273aca53fe_o.png, [Online; Accedido el 18-Enero-2015].

[22] T. Rabl y et al., "Solving big data challenges for enterprise application performance management," University of Toronto, Canada, Rep. Tec., 2012.

[23] E. BV, "Kibana interface," [imagen] Disponible en: https://www.elastic.co/guide/en/kibana/current/images/TFL-CompletedTrips.jpg, [Online; Accedido el 11-Julio-2015].

[24] D. Inc, "Statistics - textbook," http://documents.software.dell.com/Statistics/Textbook, 2015, [Online; Accedido el 15-Julio-2015].

[25] B. Palace, "Data mining," http://www.anderson.ucla.edu/faculty/jason.frand/teacher/technologies/palace/index.htm, 1996, [Online; Accedido el 15-Julio-2015].

[26] Oracle, "Data mining concepts," http://docs.oracle.com/cd/B28359_01/datamine.111/b28129/toc.htm, [Online; Accedido el 15-Julio-2015].

[27] R. O. Duda, "The fuzzy-k-means procedure."

[28] M. Müller, *Information Retrieval for Music and Motion*, 1ra. ed. Springer-Verlag Berlin Heidelberg, 2007.

[29] I. Citrix Systems, "Xenserver," http://xenserver.org/, 2015, [Online; Accedido el 13-Abril-2015].

[30] E. BV, "Elastic," https://www.elastic.co, 2015, [Online; Accedido el 14-Julio-2015].

[31] A. S. Foundation, "Apache zookeeper," http://zookeeper.apache.org/, 2010, [Online; Accedido el 25-Julio-2015].

[32] ——, "Apache hbase," http://hbase.apache.org/, 2015, [Online; Accedido el 14-Julio-2015].

[33] ——, "Apache spark," https://spark.apache.org/, 2015, [Online; Accedido el 14-Julio-2015].

[34] ——, "Javawordcount.java," https://github.com/apache/spark/blob/master/examples/src/main/java/org/apache/spark/examples/JavaWordCount.java, 2014, [Online; Accedido el 25-Julio-2015].

[35] S. Chamberlain, "elastic: General purpose interface to 'elasticsearch'," https://cran.r-project.org/web/packages/elastic/index.html, 2015, [Online; Accedido el 22-Julio-2015].

[36] Björn, "logstash-output-webhdfs," https://github.com/dstore-dbap/logstash-output-webhdfs, 2015, [Online; Accedido el 23-Julio-2015].

[37] D. Jaffe, "Bigdatademos," https://github.com/DaveJaffe/BigDataDemos, 2013, [Online; Accedido el 13-Junio-2015].

[38] W. Zhao, H. Ma, y Q. He, "Parallel k-means clustering based on mapreduce," pp. 674–679, 2009. [En línea]. Disponible en: http://www.cs.ucsb.edu/~veronika/MAE/parallelkmeansmapreduce_zhao.pdf

[39] G. Coulouris y et al., *Distributed Systems: Concepts and Design*, 5th ed. Pearson, 2012.

[40] T. White, *Hadoop: The Definitive Guide*, 3ra. ed. O'Reilly, 2012.

[41] H. Karau y et al., *Learning Spark: Lightning-fast data analysis*, 1ra. ed. O'Reilly, 2015.

[42] Y. Zhao, *R and Data Mining: Examples and Case Studies*, 1ra. ed. Elsevier, 2014.

[43] K. Beck, *Extreme Programming Explained*, 2da. ed. Addison-Wesley, 1999.

[44] D. J. Berndt y J. Clifford, "Using dynamic time warping to find patterns in time series," New York University, Rep. Tec. 94-03, 1994.

[45] I. Ukhov, "Distributed systems (tddd25)."

[46] TechTinker.com, "Arquitectura de unix/linux," [imagen] Disponible en: http://www.techtinker.com/linux/images/UnixArchitecture.gif, 2005, [Online; Accedido el 15-Julio-2015].

[47] Yearofthedragon, "Topología comunes de red," [imagen] Disponible en: https://commons.wikimedia.org/wiki/File:TopologÃ■a_de_red.png, 2004, [Online; Accedido el 13-Febrero-2015].

[48] F. Project, "Fluentd," http://www.fluentd.org, 2015, [Online; Accedido el 14-Julio-2015].

[49] I. Hortonworks, "Hortonworks," http://hortonworks.com, 2015, [Online; Accedido el 14-Julio-2015].

[50] I. Cloudera, "Cloudera," http://www.cloudera.com/, 2015, [Online; Accedido el 14-Julio-2015].

[51] I. MapR Technologies, "Mapr," https://www.mapr.com/, 2015, [Online; Accedido el 14-Julio-2015].

[52] I. Hortonwoks, "Apache mahout," http://hortonworks.com/hadoop/mahout/, 2015, [Online; Accedido el 14-Julio-2015].

[53] A. S. Foundation, "Apache mahout," http://mahout.apache.org/, 2015, [Online; Accedido el 14-Julio-2015].

[54] K. V. Shvachko, "Scalability of the hadoop distributed file system," https://developer.yahoo.com/blogs/hadoop/scalability-hadoop-distributed-file-system-452.html, 2010, [Online; Accedido el 14-Julio-2015].

[55] arunc, "The next generation of apache hadoop mapreduce," https://developer.yahoo.com/blogs/hadoop/next-generation-apache-hadoop-mapreduce-3061.html, 2011, [Online; Accedido el 27-Diciembre-2014].

[56] A. I. Pavlov y M. Cecchetti, "Squashfs," http://www.tldp.org/HOWTO/SquashFS-HOWTO/index.html, 2008, [Online; Accedido el 16-Julio-2015].

[57] J. R. Okajima, "Aufs," http://aufs.sourceforge.net/, 2015, [Online; Accedido el 16-Julio-2015].

[58] K. Beck y et al., "Manifesto for agile software development," http://agilemanifesto.org/, 2001, [Online; Accedido el 19-Julio-2015].

[59] A. Alliance, "Guide to agile practices," http://guide.agilealliance.org/, 2013, [Online; Accedido el 19-Julio-2015].

[60] R. Analytics, "Mapreduce in r," https://github.com/RevolutionAnalytics/rmr2/blob/master/docs/tutorial.md, 2014, [Online; Accedido el 12-Mayo-2015].