ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

INGENIERO TÉCNICO EN INFORMÁTICA DE GESTIÓN

Título del proyecto:

Benchmarking Distributed System in the Cloud: Yahoo! YCSB

Alumna: Sana Nawazish Ali

Tutor: José Enrique Armendáriz Iñigo

Pamplona, Fecha de defensa

Agradecimientos

Quiero aprovechar estas primeras líneas para agradecer a mi tutor: José *Enrique Armendáriz Iñigo por su dedicación, su esfuerzo, paciencia y todo el conocimiento que me ha aportado.*

Además, dado que con este documento cerraré un ciclo, quiero acordarme también de mis compañeros de la carrera, especialmente de Ander y César que han sido un gran apoyo durante todos estos años.

Por supuesto, no puedo dejar de lado a mi familia, que han sido una fuente de apoyo todos estos años, muchas gracias por vuestra paciencia.

Por último mi mayor fuente de inspiración quiero agradecer a Daniel Ventura por todo su apoyo y ayuda.

Índice

Índice de figuras

Capítulo 1. INTRODUCCIÓN

1.1. Objetivo del proyecto

El objetivo principal del proyecto es el estudio, aprendizaje y utilización del framework YCSB. Para ello también se realizará un estudio sobre las herramientas Hadoop, HBase y Cassandra y posteriormente su instalación.

YCSB es un servicio diseñado para realizar pruebas de rendimiento sobre bases de datos. Permite crear cargas de trabajo para poder analizar el comportamiento de los gestores ante diferentes entornos de cargas de trabajo y poder realizar comparativas de base de datos. Existen los siguientes escenarios de cargas:

- Workload A: Update heavily.
- Workload B: Read mostly.
- Workload C: Read only.
- Workload D: Read latest.
- Workload E: Scan short ranges.
- Workload F: Read-modify-write.

- Workload G: Write heavily.

Las cargas de trabajo se crearán sobre HBase y Cassandra, pudiendo así analizar y comparar su comportamiento en cada uno de los diferentes entornos. HBase y Cassandra son bases de datos con un tipo de almacenamiento orientado a columna, ambos escritos en Java y de libre distribución. El primero se basa en BigTable (Google) y está construido en la cima del ecosistema de Hadoop y el segundo se basa en BigTable y DynamoDB (Amazonas).

BigTable es un hash-map de datos persistente, multidimensional, ordenado, poco denso y distribuido. El hash-map está indexado por una fila, una columna y un timestamp, y cada valor en él es una matriz de bytes no interpretados. Por otro lado, DynamoDB es un servicio de bases de datos NoSQL rápido y totalmente gestionado que permite almacenar y recuperar de manera fácil y económica cualquier cantidad de datos, así como atender cualquier nivel de tráfico de solicitudes para obtener una mayor disponibilidad y durabilidad. Una de las diferencias más importantes entre HBase y Cassandra es que el primero está optimizado para la lectura y el segundo para la escritura.

Para usar HBase en modo distribuido necesitamos Hadoop. Éste es un framework que permite el procesamiento distribuido de grandes conjuntos de datos a través de grupos de ordenadores que utilizan modelos de programación simples.

En la actualidad HBase está siendo usado por grandes empresas como Facebook y Twitter, y Cassandra por empresas como Netfix y eBay. En el estudio realizado en este proyecto se puede observar que cada base de datos tiene sus ventajas e inconvenientes, y que en función del escenario una es más apropiada que otra.

1.2. Descripción del proyecto

El presente proyecto lleva por título “Benchmarking Distributed System in the Cloud: Yahoo! YCSB”. Este documento se divide en 5 bloques principales.

El primer bloque consiste en una breve introducción de la tecnología estrella: YCSB. En este bloque se realiza un análisis de **YCSB** como herramienta para realizar la motorización y análisis de carga de trabajo sobre HBase y Casandra. Este software facilita la obtención de los datos necesarios para la comparativa entre los diferentes sistemas de almacenamiento distribuidos. En el segundo bloque se detallará una descripción de lo que son las bases de datos NoSQL y los diferentes modelos que existen así como las herramientas Open Source disponibles. En este bloque también se lleva a cabo un estudio de las ventajas e inconvenientes que aporta este nuevo paradigma tecnológico, así como los servicios que puede prestar. A continuación, en el tercer bloque se estudia en profundidad la herramienta **Hadoop** como un sistema que permite procesar y analizar grandes volúmenes de datos. En los siguientes dos bloques se realiza el estudio de **HBase y Casandra** como sistemas gestores de almacenamiento de datos en la nube y se muestra el

proceso de instalación y configuración de todos los sistemas antes mencionados (Hadoop, HBase, Casandra e YCSB). Para finalizar y, como bloque principal y objetivo del proyecto, se presentarán los resultados de diversas pruebas de rendimiento y comparativa entre HBase y Casandra mediante el uso del framework YCSB.

1.3. Justificación de la tecnología

La tecnología elegida para el objetivo del proyecto ha sido la anteriormente descrita debido a que entre las diversas herramientas y motores de bases de datos son los más populares y extendidos (Hadoop, HBase y Cassandra). Su uso se extiende a grandes empresas como Yahoo, Google, Microsoft, Amazon, etc... Estas herramientas son óptimas para la utilización del framework YCSB.

Capítulo 2. YCSB

2.1. Introducción

Hoy en día es difícil decidir cuál es la base de datos más apta para una aplicación, en parte debido a las diferentes características de los distintos sistemas, y en parte porque no hay una manera fácil de comparar el rendimiento de un sistema con otro. En los últimos años ha habido una explosión de nuevos sistemas de almacenamiento de datos y de gestión en la nube. Algunos sistemas sólo se ofrecen como servicios en la nube, ya sea directamente en el caso de los servicios SimpleDB de Amazon y Azure de Microsoft, o como parte de un entorno de programación como Google App Engine de Google o Yahoo!'s YQL. Sin embargo, otros sistemas sólo se utilizan dentro de una empresa en particular, como PNUTS de Yahoo! Dynamo, BigTable de Google, y de Amazon. Muchos de estos sistemas, también se les conoce como almacenes de claves y valores o sistemas NoSQL, pero sin tener en cuenta el apodo, estos comparten los objetivos de aumentar y disminuir el número de nodos en función de la carga (elasticidad), la capacidad de prestar servicio a un gran número de clientes sin aumentar el retardo ni la calidad (escalabilidad) y la simplificación y despliegue de infraestructura.

El objetivo del proyecto YCSB es desarrollar un framework creando puntos de referencia estándar y un marco de referencia para ayudar en la evaluación de diferentes sistemas en la nube. Se centra en los sistemas que proporcionan lectura/escritura de los datos en línea. El framework consiste en una carga de trabajo que genera el cliente y un paquete de cargas de trabajo estándar que cubren partes interesantes del espacio de actuación (cargas de trabajo de lectura pesada, cargas de trabajo escritura pesada, cargas de trabajo de exploración, etc.) Un aspecto importante del marco YCSB es su extensibilidad: es fácil definir nuevos tipos de carga de trabajo e implementar una capa de interfaz de base

de datos. YCSB y las cargas de trabajo están disponibles en código abierto para que los desarrolladores puedan utilizarlo para evaluar los sistemas, y aportar nuevos paquetes de carga de trabajo.

YCSB por sí mismo, no es particularmente útil, y sólo es útil cuando se agrega el código para interactuar con un sistema de dato. El primer paso para poder utilizar YCSB es descargar, instalar y configurar un sistema de datos. Actualmente contiene código para interactuar con los siguientes sistemas:

- Cassandra (http://cassandra.apache.org/)
- DynamoDB http:/aws.amazon.com/es/dynamodb/)
- VMware vFabric GemFire (https://www.vmware.com/support/pubs/vfabric-gemfire.html)
- GigaSpaces XAP (http://www.gigaspaces.com/)
- HBase(http://hbase.apache.org/
- Infinispan (http://infinispan.org/)
- JDBC
- MapKeeper
- MongoDB (http://www.mongodb.org/)
- Oracle NoSQL Database
- Redis (http://redis.io/)
- Voldemort (http://www.project-voldemort.com/voldemort/)

2.2. Descargar YCSB

Para utilizar YCSB basta con descargar la última versión ya compilada. Para ello en una consola del sistema ejecutar los siguientes comandos:

```
wget
https://github.com/downloads/brianfrankcooper/YCSB/ycsb-
0.1.4.tar.gz
tar xfvz ycsb-0.1.4.tar.gz
cd ycsb-0.1.4
```

En el caso de que se desee implementar alguna funcionalidad nueva, es posible descargar el código del repositorio oficial; esto se puede hacer de la siguiente manera:

```
git clone git://github.com/brianfrankcooper/YCSB.git
cd YCSB
mvn clean package
```

2.3. Ejecución de una carga de trabajo

- **Establecer el sistema de base de datos a probar**: Esto se puede hacer en un sólo equipo o en un clúster, dependiendo de la configuración que se desea probar. Antes de ejecutar el cliente YCSB, se deben crear las tablas. Para algunos sistemas, hay un paso (humana-operado) manual para crear las tablas, y para otros sistemas, las tablas deben ser creadas antes de que se inicie el clúster de base de datos. Las tablas que se deben crear depende de la carga de trabajo deseada. Para CoreWorkload, el Cliente YCSB asumirá que existe una tabla llamada "usertable" con un esquema flexible de columnas.

- **Seleccionar una capa apropiada como interfaz de la Base de Datos**: La capa de interfaz DB es una clase Java que ejecuta las operaciones de lectura, inserción, actualización, borrado y escaneo. Estas llamadas son generadas por el Cliente YCSB y actúan contra la API de la base de datos. Al ejecutar el cliente de YCSB mediante la línea de comandos, se especifica el nombre de la clase de la capa a utilizar y así el cliente carga de forma dinámica la interfaz de la clase. Consultar Uso de las librerias de bases de datos para obtener más información sobre cómo utilizar los clientes mientras se ejecuta YCSB.

- **Elegir la carga de trabajo adecuada**: La carga de trabajo define los datos que se pueden cargar en la base de datos durante la fase de carga y las operaciones que se ejecutarán contra el conjunto de datos durante la fase de transacción. Si los workload existentes no satisfacen las necesidades del usuario, es posible definir workloads propios creando una clase que extienda de com.yahoo.ycsb.Workload. Para profundizar en mayor medida ver la implementación de nuevas cargas de trabajo.

- **Elegir los parámetros de ejecución apropiados:**

    - **-threads:** el número de subprocesos de cliente. De forma predeterminada, el cliente de YCSB utiliza un único subproceso de trabajo. Esta opción se utiliza a menudo para aumentar la cantidad de carga ofrecida contra la base de datos.

    - **-target:** el número de operaciones por segundo. De forma predeterminada, el cliente de YCSB tratará de realizar tantas operaciones como pueda. Por ejemplo, si cada operación tarda un promedio de 100 milisegundos, el cliente

realizará cerca de 10 operaciones por segundo por subproceso de trabajo. Por ejemplo, para generar una latencia frente a la curva de rendimiento, se pueden probar diferentes caudales de destino, y medir la latencia resultante para cada uno de ellos.

  - **-s:** el estado. Para una carga de trabajo de larga duración puede ser útil contar con el estado del informe de cliente, sólo para asegurar que no se ha interrumpido o para mostrar el progreso. Al especificar "-s" en la línea de comandos, el cliente informará del estado cada 10 segundos por la salida stderr.

- **Cargar los datos**: Las cargas de trabajo tienen dos fases: la fase de carga (que define los datos que se van a insertar) y la fase de las transacciones (que define las operaciones que se ejecutan en el conjunto de datos).
  - El parámetro "**load**" indica al cliente que sección de workload ejecutar.
  - El parámetro "**basic**" indica al cliente que utilice la capa BasicDB layer. También se puede especificar como una propiedad en el archivo de parámetros mediante la propiedad "db" (por ejemplo, "db = com.yahoo.ycsb.BasicDB").
  - El parámetro "**-P**" se utiliza para cargar los archivos de propiedades.
  - El parametro "**-p**" se utiliza para establecer los ajustes de los parámetros.
- **Ejecutar el workload**: Una vez cargados los datos, se puede ejecutar la carga de trabajo. Para ejecutar la carga de trabajo, puede utilizar el siguiente comando:

```
./bin/ycsb run basic -P workloads/workloada -P large.dat -
s > transactions.dat
```

Al final del proceso, el Cliente informará al usuario con las estadísticas de rendimiento por la salida estándar. Los códigos de retorno se definen por su capa de interfaz de base de datos y permiten ver si ha habido algún error durante la carga de trabajo.

Capítulo 3. Bases de Datos No-Relacionales

3.1. Introducción

Las bases de datos no-relacionales son comúnmente llamadas bases de datos NoSQL debido a que no utilizan el lenguaje SQL para realizar las consultas. Esta es una definición controvertida, ya que define estas bases de datos justamente por lo que no son, en lugar de por lo que son.

El término fue acuñado por Calor Strozzi en 1998 y resucitado más tarde por Eric Evans (un empleado de Rackspace,) en 2009. Éste último, además, sugirió que se denominaran a estas bases de datos como Big Data. La contribución más grande al desarrollo de los productos NOSQL fueron la serie de trabajos publicados por Google en 2003, 2004 y 2006 sobre cómo construir una infraestructura escalable para el procesamiento paralelo de grandes (enormes) cantidades de datos. Más tarde, en 2007 Amazon público su historia sobre Dynamo el almacenamiento clave/valor de alta disponibilidad.

Las bases de datos NoSQL son sistemas de almacenamiento de información que no cumplen con el esquema entidad-relación. Mientras que las tradicionales bases de datos relacionales basan su funcionamiento en tablas, joins y transacciones ACID (atomicidad, coherencia, aislamiento y durabilidad), las bases de datos NoSQL no imponen una estructura de datos en forma de tablas y relaciones (no imponen un esquema pre-fijado de tablas). Esto hace que a estas bases de datos también se les denomine “schema-less” o “schema-free”. Las bases de datos NoSQL permiten almacenar información en otros formatos como clave-valor (similar a tablas Hash), mapeo de columnas, documentos o grafos. Además de la carencia de un esquema “predeterminado”, la principal característica de las bases de datos NoSQL es que están pensadas para manipular enormes cantidades de información de manera muy rápida. Para ello suelen almacenar toda la información que

pueden en memoria (utilizando el disco como una mera herramienta de persistencia), y están preparadas para escalar horizontalmente sin perder rendimiento. Suelen funcionar bastante bien en hardware de bajo coste.

3.2. Tipos de bases de datos NoSQL

Aunque sea muy difícil de clasificar los tipos de bases de datos NoSQL, hay algunas clasificaciones generales que se pueden hacer. Es posible que productos específicos tengan más de una clasificación ya que toman distintos conceptos e ideas de varias fuentes, pero en general estos son los distintos tipos de bases de datos NoSQL que existen:

- **Bases de datos documentales:** permiten indexación de texto completo y realizar búsquedas más potentes. Los documentos que se manejan son un conjunto de datos identificados por etiquetas, los cuales internamente pueden ser JSON o otro tipo que permitan recuperación mediante claves primarias. Un uso común es el almacenamiento de datos recibidos mediante formularios web. Un ejemplo de este sistema es MongoDB.

- **Bases de datos Clave-Valor**: Son las más sencillas en cuanto a funcionalidad. En ellas simplemente se recupera un objeto binario (BLOB) a partir de una clave. Este tipo de bases de datos se suelen usar para almacenar y recuperar objetos dónde la estructura interna no es visible a la aplicación cliente. Cassandra es un ejemplo de este tipo de sistemas.

- **Bases de datos basadas en grafos**: En este tipo de base de datos, la información se representa como nodos de un grafo y sus relaciones como las aristas del mismo. El recorrido de este tipo de base de datos es similar a un algoritmo de búsqueda en profundidad o de búsqueda en anchura; algoritmos estudiados en la clase de Algoritmia. Como características interesantes de estos sistemas se encuentran que las consultas son más amplias y no demarcadas por tablas y que no es necesario definir un número determinado de atributos. HyperGraphDB es un interesante ejemplo de ello.

- **Bases de datos orientadas a objetos**: En éste tipo de base de datos la información se representa mediante objetos tal y como los presenta el modelo POO. Están diseñadas para trabajar con lenguajes de programación orientados a objetos como Java, C#, Visual Basic.NET y C++.

- **Bases de datos tabular**: Es un modelo relativamente sencillo de construir. Las tareas administrativas son similares a la administración de una base de datos multidimensional. Las mismas herramientas y enfoques son aplicables.

- **Cachés de memoria**: Las cachés en memoria son almacenamientos simples que suelen colocarse entre la base de datos y la aplicación. Se encargan de guardar en caché

objetos muy utilizados u objetos costosos de generar. Estos objetos se almacenan ya generados y se pueden recuperar de forma muy rápida.

3.3. Características

- **Consistencia eventual:** No se implementan mecanismos rígidos de consistencia como los presentes en las bases de datos relacionales, donde la confirmación de un cambio implica una comunicación del mismo a todos los nodos que lo repliquen. Esta flexibilidad hace que la consistencia se aplique cuando no se hayan modificado los datos durante un periodo de tiempo. Esto se conoce también como BASE (Basically Available Soft-state Eventual Consistency o coherencia eventual flexible básicamente disponible).

- **Estructura distribuida:** Generalmente se distribuyen los datos mediante mecanismos de tablas hash distribuidas.

- **Escalabilidad horizontal:** Consiste en la posibilidad de aumentar el rendimiento del sistema simplemente añadiendo más nodos, sin necesidad en muchos casos de realizar ninguna otra operación más que indicar al sistema cuáles son los nodos disponibles. Muchos sistemas NoSQL permiten utilizar consultas del tipo Map-Reduce, las cuales pueden ejecutarse en todos los nodos a la vez (cada uno operando sobre una porción de los datos) y reunir luego los resultados antes de devolverlos.

- **Tolerancia a fallos y redundancia.**

- **No generan cuellos de botella: el problema de fondo de los sistemas** SQL, es que deben de transcribir cada sentencia para poder ser ejecutada y, cada sentencia compleja requiere además, de un nivel de ejecución más concreto para poderse llevar a cabo, por lo que constituye un punto de entrada común, único y conflictivo en base al rendimiento.

- **Solo lo estrictamente necesario:** son sistemas simples que no tienen un sistema de consulta complejo ni con capacidad declarativa de forma que puedan realizar en una sola línea una cantidad interna de operaciones desorbitada.

- **Estructura dinámica:** La primera característica significativa es que los datos no tienen una definición de atributos fija, es decir, cada registro puede contener una información con diferente forma cada vez, pudiendo así almacenar sólo los atributos que interesen en cada uno de ellos, facilitando el polimorfismo de datos bajo una misma colección de información. También se pueden almacenar estructuras de datos complejas en un sólo documento como, por ejemplo, almacenar la información sobre una publicación de un blog (título, cuerpo de texto, autor, etc.) junto a los comentarios y etiquetas sobre el mismo, todo en un único registro. Hacerlo así aumenta la claridad (al tener todos los datos relacionados en un mismo bloque de información) y el rendimiento (no hay que

hacer un JOIN para obtener los datos relacionados, pues éstos se encuentran directamente en el mismo documento).

3.4. Modelos de implementación

- **Imagen de máquina virtual**: Las plataformas en la nube permiten a los usuarios comprar instancias de máquinas virtuales por un tiempo limitado. Es posible ejecutar una base de datos en estas máquinas virtuales. Los usuarios pueden subir su imagen propia con una base de datos instalada en ella, o utilizar imágenes prefabricadas de máquinas que ya incluyen una instalación optimizada de una base de datos.
- **Base de datos como servicio**: Algunas plataformas en la nube ofrecen opciones para el uso de bases de datos como servicio, sin lanzar físicamente una instancia de máquina virtual para la base de datos. En esta configuración, los propietarios de aplicaciones no tienen que instalar y mantener la base de datos por su cuenta. En cambio, el proveedor de servicios de base de datos se encarga de la instalación y el mantenimiento de la base de datos, y los propietarios de aplicaciones pagan de acuerdo a su uso.
- **Base de datos en la nube:** Una tercera opción es el alojamiento de la base de datos en la nube, donde la base de datos no se ofrece como un servicio, pero el proveedor de la nube aloja la base de datos y administra en nombre del propietario de la aplicación.

3.5. Punto débil: Consistencia

La consistencia de los datos es otro tema y para nada menor. De hecho, uno de los puntos que más debates concita en torno a los sistemas NoSQL, es que mientras ofrecen la gestión de un mayor volumen y variedad de datos con mayor agilidad, no necesariamente brindan garantías de la fiabilidad de los datos. Puntualmente, su mayor desventaja se presenta en que al priorizar la disponibilidad y la inmediatez de los datos para el usuario, trabajan con datos distribuidos y a veces replicados. Si los datos son muy volátiles, puede generar inconsistencia o distintas versiones.

Como la mayoría de los sistemas NoSQL no están diseñados para eso, en general, es difícil que alcancen la fiabilidad (consistencia) de las bases de datos relacionales en las que se pueden forzar modelos de consistencia fuertes mediante la presencia de restricciones de integridad, concurrencia, múltiples actualizaciones simultáneas, etc.

Al encontrase los datos distribuidos, se puede hablar de tres niveles: fiabilidad en el sentido de detección de fallas a nivel de nodos, confiabilidad de datos en términos de consistencia entre lecturas y fiabilidad por corrupción de datos por concurrencia. Existen

modelos más o menos ‘débiles’ en cada uno de esos aspectos y de acuerdo a las necesidades del negocio, se puede aplicar una u otra solución.

Bajar el nivel fiabilidad en pro de una mejor disponibilidad de los datos ante una consulta, es razonable ya que para un sistema no transaccional, no es tan vital contar con la última actualización del dato como sí lo es, disponer de un sistema con alta disponibilidad, sin caídas.

3.6. SQL vs NoSQL

Tipos	SQL	NoSQL
Modelo de almacenamiento de datos	Los registros individuales (por ejemplo, "empleados") se almacenan como filas en tablas, con cada columna que guarda un pedazo específico de datos acerca de ese registro (por ejemplo, "gerente", "fecha contratada", etc), muy similar a una hoja de cálculo. Tipos de datos separados se almacenan en tablas separadas, y luego se unen cuando se ejecutan consultas más complejas.	Varía en función del tipo de base de datos. Por ejemplo, las tiendas de clave-valor funcionan de manera similar a las bases de datos SQL, pero sólo tienen dos columnas ("clave" y "valor"), con información más compleja a veces almacenadas en las columnas de "valor". Las bases de datos de documentos pretenden el almacenamiento de todos los datos pertinentes juntos en un solo "documento" en JSON, XML u otro formato, que puede anidar valores jerárquicamente.
Esquema	Estructura y tipos de datos se fijan por adelantado. Para almacenar información acerca de un nuevo elemento de datos, toda la base de datos debe ser alterado, tiempo durante el cual la base de datos debe estar fuera de línea.	Típicamente dinámico. Se puede añadir nueva información de los registros sobre la marcha.
Escala	Verticalmente, es decir, un único servidor debe hacerse cada vez más poderoso con el fin de hacer frente a la creciente demanda. Es posible difundir bases de datos SQL entre miles de servidores, pero la ingeniería adicional significativa se	Horizontalmente, lo que significa que para agregar capacidad, un administrador de base de datos puede simplemente añadir más servidores básicos o instancias de nube. La base de datos propaga automáticamente los datos a través

	requiere generalmente.	de servidores cuando es necesario.
Modelo de desarrollo	Mezcla de código abierto (Postgres, MySQL) y de código cerrado (Oracle Database)	Código abierto.
Soporte de transacciones	Las actualizaciones se pueden configurar para completar del todo o nada en absoluto	En determinadas circunstancias y en ciertos niveles (por ejemplo, nivel de documento frente a nivel de base de datos)
Manipulación de datos	Instrucciones del lenguaje específico usando Select, Insert y Update	A través de las API orientada a objetos
Consistencia	Puede ser configurado para una fuerte coherencia	Depende del producto. Algunos proporcionan consistencia fuerte (MongoDB), mientras que otros ofrecen consistencia eventual (Cassandra).

Capítulo 4. Hadoop

4.1. Introducción

Apache Hadoop se trata de una tecnología de libre distribución que pertenece a Apache Software Foundation, está desarrollado en Java y se ejecuta dentro de la JVM. Este framework permite el tratamiento distribuido de grandes cantidades de datos (del orden de peta bytes) a través de clusters y trabajar con miles de máquinas de forma distribuida.

Actualmente las empresas recogen y generan más datos que nunca. Las bases de datos Relacionales y Datawarehouse destacan en las cargas de trabajo OLTP y OLAP de datos estructurados. Sin embargo, Hadoop fue diseñado para resolver un problema diferente: el análisis rápido y fiable tanto de los datos estructurados como de datos complejos. Como resultado, muchas empresas implementan Hadoop junto con su legado de los sistemas de IT, lo que les permite combinar conjuntos de datos antiguos y nuevos de forma potente.

La infraestructura de Hadoop está diseñada para pasar de servidores individuales a miles de máquinas, cada una con una oferta propia de computación y almacenamiento. En lugar de confiar en un hardware para ofrecer disponibilidad, la propia biblioteca Apache está diseñada para detectar y controlar los errores de programación. Es un sistema distribuido que utiliza una arquitectura Master-Slave, usando para almacenar su Hadoop Distributed File System (HDFS) y algoritmos de MapReduce para hacer cálculos. Se inspiró en los documentos sobre MapReduce y Google File System publicados por Google.

4.2. Características

- **Acceso universal a los datos:** las organizaciones utilizan Hadoop para almacenar y procesar diversas fuentes de datos y a menudo afrontarán desafíos relacionados con la combinación y el procesamiento de todos los datos pertinentes. Una plataforma de integración de datos ayuda a las organizaciones a obtener facilidad y fiabilidad en el procesamiento previo y posterior de datos dentro y fuera de Hadoop.

- **Análisis e intercambio de datos:** Hadoop sobresale en el almacenamiento de una gran diversidad de datos, pero la capacidad para extraer significados y darles sentido en todos los tipos de datos pertinentes plantea un enorme desafío. Una plataforma de integración de datos ayuda a mejorar la productividad para extraer más valor de las fuentes de datos no estructurados (imágenes, textos, binarios, estándar del sector, etc.).

- **Gestión de metadatos.** Hadoop carece de gestión de metadatos, sin los cuales, los resultados de los proyectos son sospechosos y pueden ser incoherentes y tener escasa visibilidad. Una plataforma de integración de datos proporciona exhaustivas capacidades de gestión de metadatos, con linaje y audibilidad de datos, y promueve la estandarización.

- **Calidad y gobierno de datos** Aunque algunos datos en Hadoop se conservan para tareas de almacenamiento o de experimentación que no requieren un alto nivel de calidad de datos, muchas organizaciones utilizarán Hadoop para el análisis y la elaboración de informes del usuario final. Una plataforma de integración de datos proporciona capacidades para perfilar, limpiar y gestionar datos con el fin de comprender mejor su significado, aumentar la confianza y gestionar el crecimiento de los datos de forma eficaz y segura.

- **Gestión de cargas de trabajo mixtas.** Hadoop no puede gestionar cargas de trabajo mixtas de conformidad con los acuerdos de nivel de servicio. Una plataforma de integración de datos permite la integración de conjuntos de datos de Hadoop y otras fuentes de transacciones para llevar a cabo business intelligence y análisis en tiempo real según las necesidades.

- **Optimización y reutilización de recursos.** Las organizaciones deberán encontrar y conseguir recursos de Hadoop y crear una estructura para reutilizar y estandarizar las tareas de integración de datos. Una plataforma de integración de datos promueve la reutilización de recursos de IT en diversos proyectos y aumenta el retorno de la inversión en términos de contratación y formación de personal, al tiempo que garantiza la disponibilidad de recursos compatibles con el ecosistema.

- **Interoperabilidad con el resto de la arquitectura.** La racionalización e incorporación de Hadoop como parte del entorno ampliado supone un desafío. Las capacidades de acceso universal a los datos y la transformación de una plataforma de integración de datos respalda la incorporación de Hadoop como parte de un ciclo de procesamiento de

datos y análisis integral, que ayuda a salvar las distancias entre Hadoop y su inversión en IT existente.

4.3. Distribuciones y Versiones de Hadoop

Una de las primeras tareas a asumir en la planificación de una implementación de Hadoop es la selección de la distribución y versión de Hadoop la más apropiada, dadas las características y estabilidad requerida. Este proceso requiere la participación de los que finalmente usaran el clúster: desarrolladores, analistas, y posiblemente otros sistemas tales como aplicaciones de inteligencia de negocios.

Cloudera[15] es una compañía que proporciona herramientas de soporte, consultoría y gestión para Hadoop, también tiene una distribución de software llamado Cloudera's Distribution Including Apache Hadoop, o simplemente CDH. Al igual que con la versión ASF (Apache Software Foundation), es un software de código abierto 100% disponible bajo la Licencia Apache Software y es gratuito para uso personal y comercial. Al igual que muchas compañías de software de código abierto, Cloudera ofrece una versión estable de Apache Hadoop para un número amplio de sistemas operativos y tiene una garantía de calidad de grado comercial y proceso de pruebas. Cloudera tiene empleados a muchos de los committers Apache Hadoop (las personas que tienen privilegios para subir código a los repositorios de Apache) que trabajan en Hadoop de tiempo completo. Dado que muchos usuarios despliegan muchos de los proyectos relacionados con Apache Hadoop, Cloudera incluye estos proyectos en la CDH, garantizando así la compatibilidad entre componentes. CDH actualmente incluye Apache Hadoop, Apache HBase, Apache Hive, Apache Pig, Apache Sqoop, Apache Flume, Apache ZooKeeper, Apache Oozie, Apache Mahout y Hue. En la web oficial de Cloudera se puede encontrar una lista completa de los componentes incluidos en el CDH.

La última gran versión del CDH es CDH4, que está basado en Apache Hadoop 2.0.0. Incluye las principales mejoras HDFS tales como alta disponibilidad en los NameNode, así como también un puerto delantero de los demonios MRv1 probados. CDH está disponible como archivos binarios encapsulados en formato tar, RedHat, Enterprise Linux 5 y 6 RPMs, SuSE, Enterprise Linux RPMs y paquetes deb de Debian. Además los gestores de paquetes Yum, Zypper y APT lo incluyen en sus repositorios para sus respectivos sistemas con tal de facilitar la instalación.

El interés por Hadoop ha crecido en los últimos años. Esto ha llevado a corregir e implementar un número proporcional de características y correcciones. Algunas de estas características fueron tan importante o tuvieron un impacto tan radical que se desarrollaron en otras ramas. Como es de esperar, esto a su vez condujo a una matriz, de un tanto mareante, de las salidas del software y de las líneas paralelas de desarrollo.

A continuación se expone una breve descripción de las distintas líneas de desarrollo y su estado. Esta información también se representa visualmente en la Figura 4-1.

0.20.0-0.20.2: La rama de 0.20 Hadoop es extremadamente estable y ha visto un poco de la producción de burn-in. Esta rama ha sido una de las ramas de más larga vida de la historia de Hadoop ya que es la primera versión, la cual apareció en abril de 2009. CDH2 y CDH3 están basados en esta rama, aunque con muchas características y correcciones de errores de 0.21, 0.22 y 1.0.

0.20-append: Una de las características que faltan de 0.20 fue el apoyo a archivo anexo en HDFS. Apache HBase se basa en la capacidad de sincronizar su escritura delante de registro, (por ejemplo, el contenido del archivo vigor en el disco), utiliza la misma funcionalidad básica que el archivo de datos anexados. Append se considera una característica potencialmente desestabilizadora y muchos no estuvieron de acuerdo sobre la aplicación, por lo que fue relegada en una rama. Esta rama fue llamada 0.20-anexados.

0.20.203-0.20.205: Hubo un fuerte deseo dentro de la comunidad para producir una versión oficial de Hadoop que incluyó el trabajo 0.20-seguridad. Las versiones 0.20.20X no solo contenían las características de seguridad de 0.20-seguridad, sino también correcciones de errores y mejoras en la línea de desarrollo 0.20. En general, ya no tiene sentido desplegar estas versiones a medida que se sustituya por la versión 1.0.0.

0.21.0: Esta se consideró una vista previa para desarrolladores o liberación alfa, calidad para destacar algunos de las características que se encuentra actualmente en desarrollo en el momento. Esta versión no incluye la seguridad, pero sí que tiene anexados.

0.23.0: La versión 0.23 incluye la seguridad, añada, hilo y HDFS federación. Este comunicado ha sido apodado como la vista previa para desarrolladores o liberación calidad alfa. Esta línea de desarrollo es sustituida por 2.0.0.

1.0.0: En un tema permanente de confusión. La versión 1.0.0 de Hadoop fue liberada de la línea de desarrollo 0.20.205. Esto significa que 1.0.0 no contiene todas las características y correcciones que se encuentran en el 0.21, 0.22, y 0.23 comunicados. Dicho esto, sí incluye la seguridad.

2.0.0: En mayo de 2012, la versión 2.0.0 fue liberada de la rama 0.23.0 y 0.23.0 al igual que, se considera calidad alfa. Principalmente, esto se debe a que incluye HILO y elimina los tradicionales JobTracker y TaskTracker demonios MRv1. Mientras HILO es API compatible con MRv1, la implementación subyacente es lo suficientemente diferente como para que pida más pruebas significativas antes de ser considerado listo para la producción.

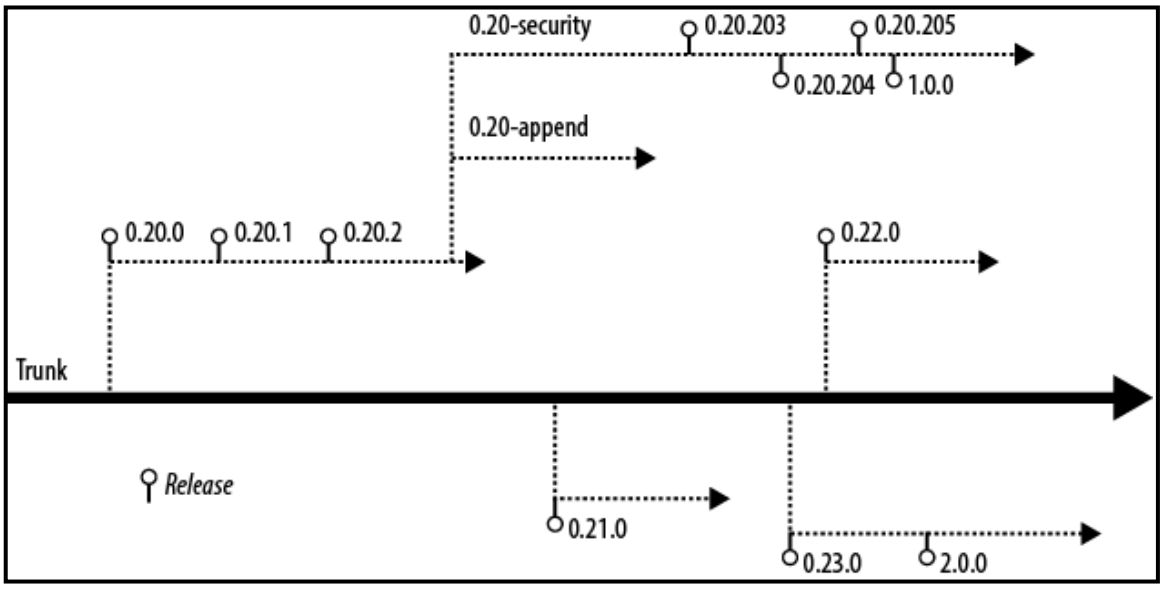


Figura 1. Ramas de Hadoop y versiones

4.4. Arquitectura

El diseño de Hadoop más tradicional se divide en dos partes principales. Por un lado, la implementación de MapReduce que se encarga del procesamiento de la información de forma distribuida. Este es un framework desarrollado por Google que soporta procesamiento distribuido sobre grandes cantidades de datos en un clúster. Es la combinación de dos procesos llamados Map y Reduce. El algoritmo Map and Reduce se basa en el clásico paradigma computacional llamado divide y vencerás. Desde un punto de vista abstracto este algoritmo divide el problema en problemas más sencillos, los resuelve y al final une todos los resultados intermedios y los devuelve. Se compone de dos tipos de elementos, el JobTracker y los TaskTrackers. En un clúster de un solo nodo, existe uno de cada, pero en un multi-nodo, hay un JobTracker y n TaskTrackers (tantos como nodos).

El usuario ejecuta aplicaciones sobre los datos del sistema y en la capa de MapReduce se generan las tareas que se distribuyen entre los nodos. Esta capa está claramente separada de la capa de HDFS excepto en el momento en el que una tarea llega al nodo en el que se alojan los datos con los que tiene que trabajar. En este momento, ambas capas (MapReduce y HDFS) interaccionan para producir los datos resultantes de la aplicación de dicha tarea y la información almacenada necesaria para la compilación. De esta interacción surgen nuevos datos que pasan a gestionarse en la capa HDFS y una nueva tarea con el resultado de la compilación.

El JobTracker es el servicio de Hadoop que se encarga de las tareas MapReduce que solicitan los clientes. Estas tareas las ejecutan los TaskTracker. Los TaskTrackers aceptan tareas Map o Reduce que le envía el JobTracker. Se configura con número determinado de tareas que puede ejecutar en paralelo, por lo tanto, si tiene slots libres será

un nodo elegible por el JobTracker para ejecutar una tarea. Un TaskTracker se comunica directamente con los DataNode (como otro cliente cualquiera) una vez que el NameNode le ha dado su localización. Normalmente el JobTracker escogerá el nodo que tenga los datos o en su defecto el que esté en el mismo rack. El JobTracker monitoriza la ejecución de las tareas en los TaskTracker para saber si se ha realizado con éxito, o si ha fallado entonces encargársela a otro TaskTracker.

La otra parte fundamental del sistema de Hadoop es el sistema de ficheros distribuido Hadoop Distributed File System (HDFS), el cual está baso en Google File System. Esta sección del entorno está dedicada plenamente a la gestión de los datos del sistema Apache Hadoop. En esta capa del sistema, se gestiona el almacenamiento de los datos, la distribución de los mismos y se generan los metadatos que dan la información necesaria sobre los datos del sistema.

El elemento principal de esta división del sistema es el NameNode. Éste divide en bloque los datos y decide en qué nodo se almacena cada bloque; el número de réplicas de los datos para mantener un cierto nivel de seguridad ante pérdidas de bloques; y genera y almacena todos los metadatos referentes a cada bloque para poder controlar la información que corre por el sistema. NameNode es el maestro o master de todos los DataNode. Es el responsable de manejar el espacio de nombres del sistema de ficheros y controlar los accesos de los clientes externos. Los metadatos contenidos en el NameNode indican en cual o cuales DataNodes está almacenado cada objeto y procesa las operaciones sobre el sistema de ficheros determinando que DataNode es el responsable de ejecutarla. Haciendo una abstracción, se puede decir que asume el rol de una btabla de ficheros en un sistema de ficheros tradicional (como FAT, MFT o la tabla de inodos en FAT32, NTFS y ext3/ext4 respectivamente) o de un DNS en Internet. Almacena la información a cerca de donde se encuentra los objetos en HDFS.

Por otro lado, el lugar donde se almacenan los bloques son los DataNodes. Generalmente, en un clúster típico, existe un único NameNode y tantos DataNodes como nodos. Este tipo de nodos actúa como un almacenamiento de bloques para HDFS. Un clúster de Hadoop puede contener cientos o miles de DataNodes. Estos DataNodes responden a las peticiones de lectura y escritura de los clientes y hacen réplicas de los bloques que le envía el NameNode. Cada DataNode envía periódicamente paquetes al NameNode con información de sus bloques para que éste pueda validar la consistencia de la información con otros DataNodes.

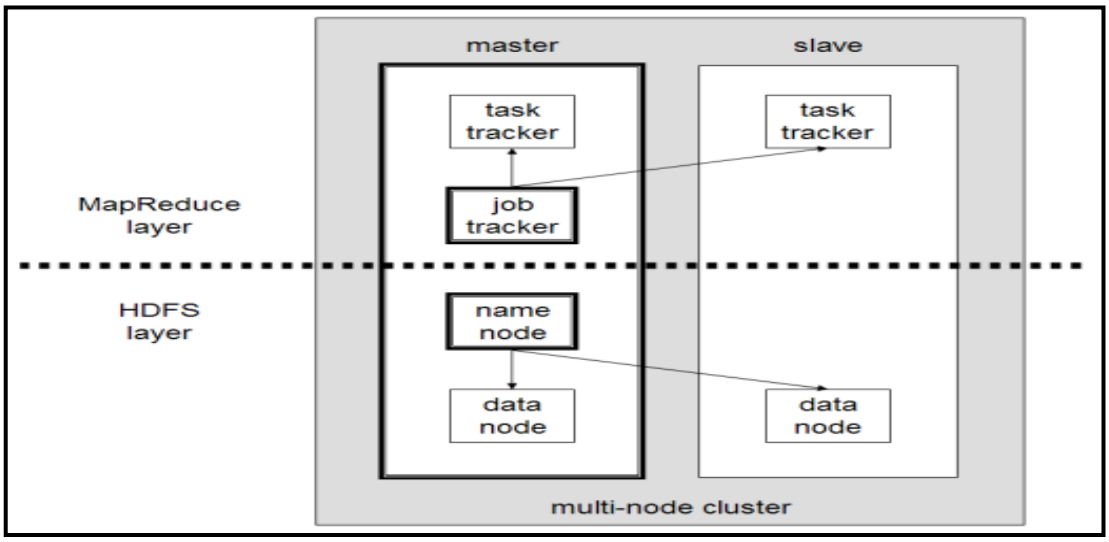


Figura 2. Arquitectura Hadoop.

Con la versión 0.23 de Hadoop se introducen cambios en MapReduce que ahora pasa a llamarse MapReduce 2.0 (alias MRv2 o YARN). Esta es la nueva infraestructura sobre la que se soporta Hadoop. Originariamente escrito para implementar MapReduce, YARN (o MR2, implementado en Hadoop **0.23/2.x y superiores**) generaliza la arquitectura original (también conocida como MR1) para dar soporte a aplicaciones no sólo MapReduce. MR1 está enfocado a la gestión eficiente de recursos para dar soporte exclusivamente a tareas MapReduce.

Cambios arquitectónicos:

Es importante notar que en MR1 teníamos el JobTracker con sus TaskTrackers, y el NameNode con sus DataNodes de la capa HDFS para dar soporte a los anteriores respectivamente. La tarea del JobTracker era doble, la gestión de recursos por un lado, y el dispatching y monitorización de los trabajos por otro. En YARN, el JobTracker se transforma en el ResourceManager. El ResourceManager a su vez se separa en dos componentes especializados enfocados a las dos tareas: Scheduler y ApplicationManager.

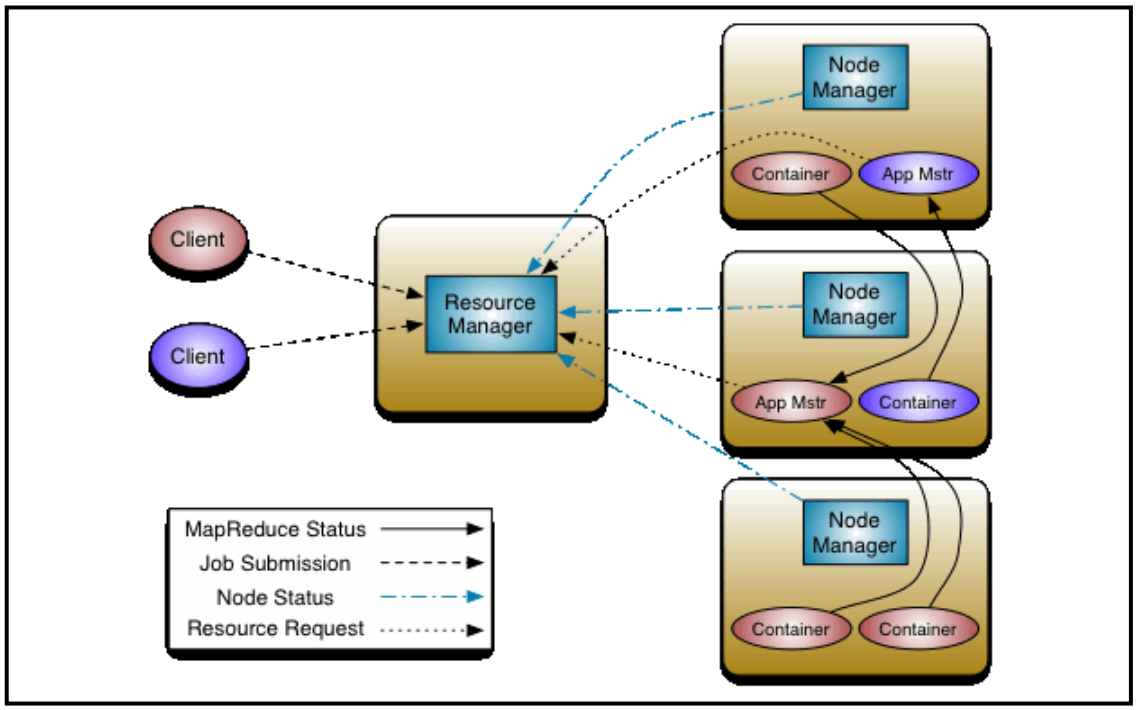


Figura 3.Scheduler y ApplicationsManager

El scheduler se encarga de disponibilizar los recursos para las distintas aplicaciones agrupándolas según sus restricciones de capacidad, colas, etc... Pero no se ocupa de monitorizar estas aplicaciones. Es el encargado de particionar los recursos del clúster entre las distintas aplicaciones.

Los NodeManagers son los delegados del scheduler en cada nodo, responsables de monitorizar el consumo de recursos de cada nodo, y de enviar esta información al scheduler. Los ApplicationMaster son los responsables de solicitar al scheduler los recursos necesarios para las aplicaciones, y éstos sí que se encargan de monitorizar el estado de las mismas y su progreso. Son gestionados por el ApplicationManager.

Los recursos se gestionan en forma de “containers” (Resource Containers, que encapsulan CPU, memoria, disco, red, etc...). Los trabajos MapReduce de versiones anteriores siguen siendo compatibles en esta nueva arquitectura.

En MR1 un cliente enviaba un job al JobTracker, y este lo desmenuzaba en tasks que remitía a los TaskTrackers. En MR2, el cliente envía una aplicación al Resource Manager, que remite esta petición al ApplicationManager, que a su vez lanza los ApplicationMaster en cada nodo. Los ApplicationMaster son los responsables de obtener un container (pack de recursos) del Scheduler para ejecutar su aplicación, de lanzarla y de monitorizarla y relanzarla si es necesario. El ApplicationManager se encarga de mantener levantados los ApplicationMasters.

Debajo de estos nodos sigue estando la capa del HDFS, que implementa un sistema de ficheros distribuido y que abstrae el hardware subyacente, sin sufrir modificaciones. Los mismos nodos NameNode y DataNode (y extensiones, SecondNameNode, backup)

4.5. Modos de ejecución de Hadoop

- **Modo single node:** Es el modo más sencillo ya que se compone de un solo nodo y en él se configuran las dos capas de las que se compone el clúster. Además de diferenciar ambas secciones, se instalan en cada una de ellas los cuatro elementos. Es decir, este clúster con nodo único, se compone de una máquina que ejerce todas las tareas tanto para gestionar los datos como las aplicaciones. En ella se instala el controlador de tareas (JobTracker), el ejecutor de las mismas (TaskTracker), el NameNode que gestiona los datos y los metadatos y el DataNode que almacena la información. Al estar todos los elementos del clúster alojados en una única máquina, el rendimiento del sistema es peor. Sin embargo, la funcionalidad del clúster es análoga a la de otro clúster con miles de nodos, la única diferencia es el rendimiento. Los datos que se gestionan en el NameNode, son distribuidos dentro del mismo disco, eso sí se generan los mismos metadatos que para un clúster superior y los mismos bloques para el almacenamiento de la información. Por otro lado, al JobTracker le llegan aplicaciones ejecutadas por el usuario y éste genera las tareas Map y Reduce que, en lugar de distribuir entre los nodos del clúster, manda al TaskTracker que está alojado dentro de la misma máquina. Como puede apreciar, el funcionamiento es igual que el de un clúster, solo que se verá afectado el rendimiento del sistema.

- **Modo pseudo-distribuido:** Esta forma de ejecutar es igual que la anterior a diferencia de que cada servicio de Hadoop se ejecuta en diferentes procesos Java pero en la misma máquina.

- **Modo distribuido**: Un clúster Apache Hadoop distribuido se compone de un Nodo Máster y tantos Nodos Esclavos como se quiera desplegar según los límites que tenga el entorno. Esos límites, tienen que ver con los recursos hardware de los que dispone el entorno. Según los recursos de los que disponga respecto a procesamiento, memoria, etc., es posible desplegar clústeres con mayor número de nodos.

- En ocasiones, formar un clúster y configurar todos los nodos del mismo para que trabajen de manera conjunta se convierte en algo realmente complejo. Sin embargo, en el caso que nos ocupa, formar un entorno de trabajo bajo esta tecnología es algo más sencillo de lo que pueda parecer y a la vez realmente flexible y adaptable, ya que permite incluir cualquier tipo de equipo (máquinas virtuales incluidas) y tantas cuantas deseemos, incluso una vez concluida la configuración inicial. Esta configuración se describe más delante en la instalación de las herramientas.

4.6. Inconvenientes

- Un posible intercambio de mensajes excesivo entre los nodos de Hadoop (réplicas, sincronizaciones, actualizaciones de metadatos, localización) puede ralentizar la red.

Una solución posible es utilizar una red separada para Hadoop y otra para el tráfico de los datos.

- No todos los problemas se pueden resolver o traducir a problemas MapReduce. Aun así, se puede utilizar Hadoop como sistema de ficheros distribuido por medio HDFS.

4.7. El ecosistema de Hadoop

- En Hadoop tenemos un ecosistema muy diverso, que crece día tras día, por lo que es difícil saber de todos los proyectos que interactúan con Hadoop de alguna forma. A continuación sólo se muestran los más comunes.

- **Chukwa**: es un sistema de captura de datos y framework de análisis que trabaja con Hadoop para procesar y analizar grandes volúmenes de logs. Incluye herramientas para mostrar y monitorizar los datos capturados.

- **Apache Flume**: es un sistema distribuido para capturar de forma eficiente, agregar y mover grandes cuantidades de datos log de diferentes orígenes (diferentes servidores) a un repositorio central, simplificando el proceso de recolectar estos datos para almacenarlos en Hadoop y poder analizarlos. ***Flume*** y ***Chukwa*** son proyectos parecidos, la principal diferencia es que ***Chukwa*** está pensado para ser usado en ***Batch***.

- **Hive**: es un sistema de Data Warehouse para Hadoop que facilita el uso de la agregación de los datos, ad-hoc queries, y el análisis de grandes datasets almacenados en Hadoop. ***Hive*** proporciona métodos de consulta de los datos usando un lenguaje parecido al SQL, llamado ***HiveQL***. Además permite de usar los tradicionales Map/Reduce cuando el rendimiento no es el correcto. Tiene interfaces***JDBC/ODBC***, por lo que empieza a funcionar su integración con herramientas de BI.

- **Apache HBase**: se trata de la base de datos de Hadoop. ***HBase*** es el componente de Hadoop a usar, cuando se requiere escrituras/lecturas en tiempo real y acceso aleatorio para grandes conjuntos de datos. Es una base de datos orientada a la columna, eso quiere decir que no sigue el esquema relacional. No admite SQL.

- **Apache Mahout**: es un proyecto para crear aprendizaje automático y data mining usando Hadoop. Es decir, Mahout nos puede ayudar a descubrir patrones en grandes datasets. Tiene algoritmos de ***recomendación***, ***clustering*** y ***clasificación***.

- **Apache Sqoop**: (“Sql-to-Hadoop”), es una herramienta diseñada para transferir de forma eficiente bulk data entre Hadoop y sistemas de almacenamiento con datos estructurados, como bases de datos relacionales. Algunas de sus características son:

    - Permite importar tablas individuales o bases de datos enteras a HDFS.

    - Genera clases Java que permiten interactuar con los datos importados.

- Permite importar de las bases de datos SQL a Hive.

- **Apache ZooKeeper**: es un proyecto de Apache que proporciona una infraestructura centralizada y de servicios que permiten la sincronización del clúster. ZooKeeper mantiene objetos comunes que se necesiten en grandes entornos de clúster. Algunos ejemplos de estos objetos son información de la configuración, jerarquía de nombres…

- **Apache Lucene**: se trata de una librería escrita en Java, para buscar textos. ***Lucene*** permite indexar cualquier texto que deseemos, permitiéndonos después encontrarlos basados en cualquier criterio de búsqueda. Aunque Lucene sólo funciona en texto plano, hay plugins que permite la indexación y búsqueda de contenido en documentos Word, Pdf, XML o páginas HTML.

- **Apache Pig**: ***inicialmente*** desarrollado por Yahoo, permite a los usuarios de Hadoop centrarse más en el análisis de los datos y menos en la creación de programas MapReduce. Para simplificar el análisis proporciona un lenguaje procedural de alto nivel. Su nombre viene de la siguiente analogía, al igual que los cerdos comen de todo, el lenguaje de programación Pig está pensado para poder trabajar en cualquier tipo de datos. Pig consta de dos componentes: el lenguaje en sí, llamado ***PigLatin y e***l ***entorno de ejecución***, donde los programas PigLatin se ejecutan.

- **Jaql**: es un lenguaje de consulta funcional y declarativo que facilita la explotación de información organizada en formato ***JSON*** (JavaScript Object Notation), e incluso en archivos semi-estructurados de texto plano. Diseñado inicialmente por IBM. Jaql permite hacer select, join, group y filtrar datos almacenados en HDFS. El objetivo de Jaql es que el desarrollador de aplicaciones de Hadoop pueda concentrarse en qué quiere obtener, y no en cómo lo tenga que obtener. Jaql analiza la lógica y la distribuye en Map y Reduce según sea necesario.

- **Apache Avro**: es un sistema de serialización de datos. En los proyectos en Hadoop, suele haber grandes cuantidades de datos, la serialización se usa para procesarlos y almacenar estos datos, de forma que el rendimiento en tiempo sea efectivo. Esta serialización puede ser en texto en plano, JSON, en formato binario. Con Avro podemos almacenar y leer los datos fácilmente desde diferentes lenguajes de programación. Está optimizado para minimizar el espacio en disco necesario para nuestros datos.

- **Apache UIMA**: Unstructured Information Management Applications): es un framework para analizar grandes volúmenes de datos no estructurados, como texto, video, datos de audio, etc… y obtener conocimiento que sea relevante para el usuario final. Por ejemplo a partir de un fichero plano, poder descubrir que entidades son personas, lugares, organizaciones, etc…

4.8. Ficheros de configuración

Nombre del fichero	Formato	Descripción
Hadoop-env.sh	Bash script	Las variables de entorno que se utilizan en las secuencias de comandos para ejecutar Hadoop.
core-site.xml	XML	Opciones de configuración de Hadoop Core, como la configuración de E/S que son comunes a HDFS y MapReduce.
hdfs-site.xml	XML	Opciones de configuración de los demonios: HDFS NameNode, el secundario NameNode y el DataNode.
mapred-site.xml	XML	Opciones de configuración de los demonios MapReduce: el JobTracker y los TaskTrackers.
masters	Texto plano	Una lista de equipos (uno por línea) que ejecutan un NameNode secundaria.
slaves	Texto plano	Una lista de equipos (uno por línea). Cada uno ejecuta un DataNode y TaskTracker.
Hadoop-metrics.properties	Propiedades Java	Propiedades para el control de cómo las métricas se publican en Hadoop.
log4j.properties	Propiedades Java	Propiedades en los archivos de registro del sistema, el registro de auditoría NameNode, y el registro de tareas para el proceso hijo TaskTracker.

4.9. Hadoop en la actualidad

Hadoop se puede utilizar en teoría para casi cualquier tipo de trabajo batch, mejor para trabajos en tiempo real, ya que son más fáciles de dividir y ejecutar en paralelo. Entre los campos actuales a aplicación se encuentran:

- Análisis de logs.
- Análisis de mercado.
- Machine learning y data mining.

- Procesamiento de imágenes.
- Procesamiento de mensajes XML.
- Web crawling.
- Indexación de textos.

Actualmente Hadoop es un framework muy extendido en el ámbito empresarial, sobre todo en compañías que manejan grandes volúmenes de datos. Entre las que podemos descartar las siguientes empresas:

- Yahoo: La aplicación Yahoo! Search Webmap está implementado con Hadoop sobre un clúster de más de 10.000 nodos Linux y la información que produce es la utilizada por el buscador de Yahoo.
- Facebook: Tiene a día de hoy el mayor clúster Hadoop del mundo que almacena hasta 30 peta bytes de información.
- Amazon A9: Se utiliza para la generación de índices de búsqueda de los productos ofertados en el portal. Disponen de varios clústeres de entre 1 y 100 nodos
- The New York Times: Utiliza Hadoop y EC2 (Amazon Elastic Compute Cloud) para convertir 4 Tera bytes de imágenes TIFF en imágenes PNG de 800 KB para ser mostradas en la Web en 36 horas.
- Además existen compañías cuyo negocio es principal es Hadoop, como Cloudera, que comercializa CDH (Cloudera's Distribution including Apache Hadoop), dando soporte en la configuración y despliegue de clústeres Hadoop. Además proporciona servicios de consultoría y formación en esta tecnología. Todo el software que distribuyen es Open Source.

Capítulo 5. HBase

5.1. Introducción

HBase es el nombre de la base de datos que utiliza el proyecto Hadoop, la cual se desarrolla en el marco de Apache Software Foundation (Hadoop Apache Project). En 2006, el equipo de Google Labs publicó un artículo titulado BigTable: Un sistema de almacenamiento distribuido de datos estructurados**.** En él se describe un índice distribuido diseñado para gestionar grandes bases de datos (petabytes de datos) en un clúster de servidores de datos. BigTable soporta claves de búsqueda: búsqueda por rango y análisis de archivos de alto rendimiento, y también proporciona un almacenamiento flexible para los datos estructurados. BigTable es la tecnología que permite que Google escale toda su infraestructura a centenares de miles de servidores de bajo costo de una manera homogénea, sin restricciones de una d base de datos relacional (como MySQL, Oracle o MS SQL Server). HBase es un clon en código abierto de BigTable e imita su diseño.

Más específicamente, HBase es una base de datos distribuida que permite escalar casi linealmente con simplemente agregar más servidores al sistema. Para esto HBase puede utilizar opcionalmente HDFS. HBase es apropiado para casos en los que se necesita tiempos de acceso de lectura/escritura muy bajos en una infraestructura BigData. Este gestor de datos es capaz de almacenar tablas muy grandes (miles de millones de filas multiplicado por millones de columnas) en clúster compuestos de hardware básico. A diferencia de las bases de datos racionales, HBase no admite un lenguaje de consultas tipo SQL. Con esta arquitectura es posible lograr no sólo altos niveles de rendimiento, sino también altos niveles de redundancia en datos, ya que a cierto nivel los datos están distribuidos de manera redundante similar a como los datos son distribuidos en un sistema de discos duros tecnología RAID 5 por ejemplo.

Algo más importante a reseñar es que si algo hemos aprendido de Internet, y en particular de Aplicaciones Web, es que en muchos casos la demanda de usuarios es impredecible. Una pequeña empresa con una gran idea puede de la noche a la mañana recibir millones de visitas, y por ello es mejor estar preparado a no poder ofrecer el servicio a todos los usuarios. Es aquí en donde radica la importancia de HBase.

HBase no es solo una base de datos altamente escalable, sino que además es un nuevo paradigma que por fin quizás deje atrás las bases de datos relacionales en muchos casos típicos. Por décadas hemos dependido de bases de datos relacionales, y estas han sobrevivido a las bases de datos de objetos, XML, y muchos otros, pero gracias a las lecciones que Google le ha dado al mercado de la infraestructura, se ha hecho evidente que esta nueva filosofía de diseñar bases de datos es quizás la más adaptable para las necesidades de hoy en Internet. Sin embargo, hay que entender que aunque HBase representa una amenaza a largo plazo tanto para las bases de datos comerciales como Oracle, DB2 o MS SQL, así como a las de código libre como PostgreSQL o hasta la misma MySQL, aun así hay situaciones en donde es mejor contar con una base de datos relacional.

HBase está implementado en Java, por lo que funciona en cualquier plataforma que tenga una máquina virtual Java, lo que hace de HBase una tecnología fácil de adoptar en entornos de Unix, Linux, Windows y OS X. También funciona en Windows pero para ello requiere una instalación adicional con Cygwin.

Quizás una de las mejores noticias sobre esta tecnología es que es Open Source, lo que significa que puede ser utilizado por cualquiera, así como poder aportar a su desarrollo o aprender de su código fuente.

5.2. Características

- Escalabilidad lineal y modular.
- Consistencia estricta en lectura y escritura.
- Almacenamiento y configuración de registros en tablas automático.
- Soporte de fallos automático entre RegionServers.
- Clases para realizar copias de trabajos Hadoop MapReduce con tablas HBase Apache.
- API en Java fácil de utilizar que permite el acceso de clientes.
- Bloqueo de caché y Bloom de filtros para consultas en tiempo real.
- Thrift gateway y Web REST-full, servicios que soportan XML, protobuf y opciones de codificación de datos binarios.

- Extensible, basado en jruby (JIRB) shell.
- Soporte para exportar las métricas de Hadoop a diferentes formatos.

5.3. Arquitectura

En HBase hay dos tipos de máquinas: maestros (HBase Master o HMaster) y esclavos (HRegionServers). Cada uno de ellos necesita diferentes especificaciones de hardware. El maestro no necesita tanto espacio como el esclavo, por lo que no tiene sentido tener un maestro con mucha memoria.

El primer paso para comprender la arquitectura de HBase es entender la imagen de la Figura 6-1 que muestra una visión general de cómo HBase y el sistema de archivos de Hadoop se combinan para almacenar datos. Uno de los aspectos más ocultos de HBase es como se almacenan realmente los datos. Aunque la mayoría de los usuarios nunca tengan que preocuparse por esto, es interesante tener conocimientos de la arquitectura para saber cuáles son las distintas opciones de configuración existentes y poder optimizar al máximo su uso. HBase maneja dos tipos de archivos. El primero se utiliza para el registro de escritura (write-ahead log) y el otro para el almacenamiento de datos reales. Los archivos son manejados principalmente por los HRegionServer pero en algunos casos HMaster también puede realizar operaciones con archivos de bajo nivel. También se puede observar que los archivos se dividen en bloques más pequeños cuando se almacenan en el sistema de archivos distribuido Hadoop (HDFS). Esta es una de las configuraciones con las que se podría jugar para manejar mejor los datos más grandes o más pequeños.

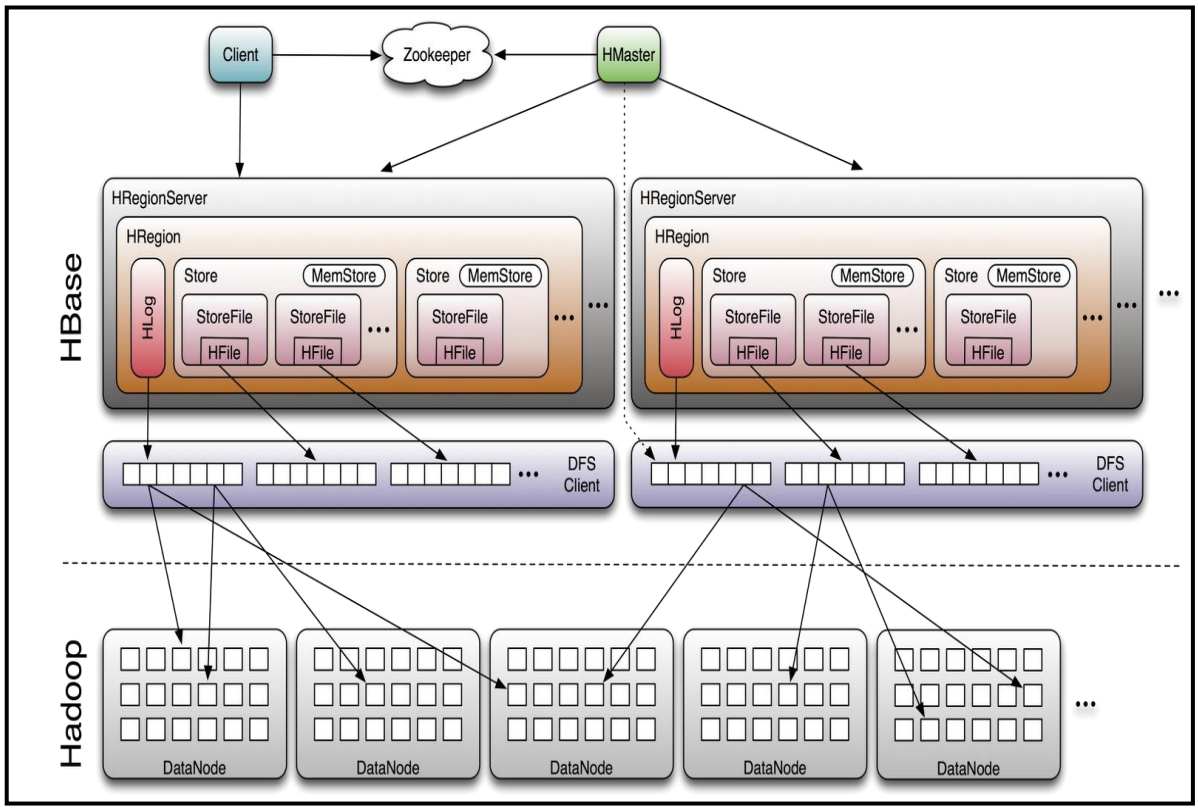


Figura 4. Arquitectura HBase

HBase proporciona baja latencia en lectura y escritura en la parte superior de HDFS y es capaz de gestionar petabytes de datos. Las unidades que proporcionan la escalabilidad horizontal son las llamadas Regiones. Las regiones son un subconjunto de datos de la tabla que se almacenan ordenados en una gama contigua de filas.

En HBase los esclavos son llamados RegionServer. Cada servidor de la región es responsable de servir un conjunto de regiones, y una región (es decir, un intervalo de filas) puede ser servida por una sola RegionServer. Una RegionServer realiza el hosting y gestiona las Regiones, divide las regiones en caso de ser necesario, manipula las solicitudes de escritura y lectura y establece la comunicación con el cliente. Las regiones se asignan a las RegionServers.

En HBase el Master es el responsable de la coordinación de las Regiones en el clúster y ejecuta las operaciones administrativas. Un RegionServer puede servir a una o más regiones. Cada región está asignada a un servidor de la región y el maestro puede decidir mover una región de una RegionServer a otra como resultado de una operación de equilibrio de carga. El Maestro también se ocupa de los fallos de las RegionServer, asignando en caso de fallo, la región a otra RegionServer. El mapeo de las regiones a RegionServer se mantiene en la tabla META. Al leer META se puede fácilmente identificar qué región contiene lo que se busca. Esto significa que para operaciones de lectura y escritura, el maestro no está involucrado en absoluto, y los clientes pueden ir directamente a la RegionServer para obtener los datos.

Cada RegionServer contiene un registro de lectura llamado HLog y varias regiones. Cada región a su vez se compone de MemStore y múltiples StoreFiles (HFile). Los datos se localizan en estos StoreFiles en forma de familias de columnas. El mapeo de las regiones a RegionServer se mantiene en la tabla META. Al intentar leer o escribir datos desde HBase, los clientes leen la información requerida de la tabla META localizada en la región. Y se comunican directamente con el servidor de la región correspondiente.

La asignación y distribución de las regiones a RegionServers es automática y en gran parte no requiere intervención por parte del operador. Inicialmente solo hay una región por tabla. Los datos almacenados en la BigTable se encuentran en su RowKey. Ésta es similar a una clave principal de una base de datos relacional. El diseño de un esquema HBase realmente se reduce a optimizar el uso de ese RowKey. Una de las funciones interesantes en HBase es el Auto-Sharding Figura 6-2, la cual muestra como las tablas se distribuyen dinámicamente por el sistema cuando se vuelven demasiado grandes. Cuando las regiones se vuelven demasiado grandes después de añadir más filas, la región se divide en dos creando así dos mitades aproximadamente iguales.

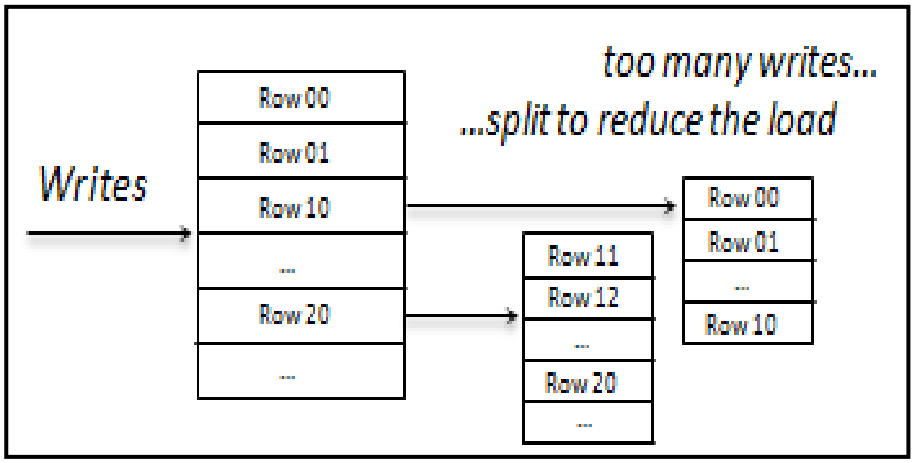


Figura 5. Auto-Sharding

5.4. Flujo de datos

El flujo general comienza cuando un nuevo cliente contacta con el quórum Zookeeper (un grupo independiente de nodos Zookeeper) para encontrar la clave de una fila en particular. Lo hace mediante la recuperación del nombre del servidor (es decir, el nombre de host) que aloja la región ROOT de Zookeeper. Con esa información se puede consultar el servidor para obtener el servidor que aloja la tabla META. Por último se consulta la tabla META del servidor para recuperar el servidor que contiene la fila que el cliente está buscando. Una vez que se sabe qué región contiene la fila buscada se almacena en cache la información de dicho servidor y la del correspondiente HRegionServer. Así para las futuras consultas el cliente tiene almacenada en la cache la información y no tendrá la necesidad de consultar la tabla META.

El flujo de escritura comienza cuando el cliente envía una solicitud put (HTable.put) al HRegionServer. El primer paso es escribir el dato en el write-ahead-log (WAL) usando la clase HLog.

El WAL (write-ahead log) es un fichero estándar de Hadoop SequenceFile que almacena instancias de HLogKey. WAL es un salvavidas en caso de desastres. Al igual que un registro BIN de MySQL, registra todos los cambios de datos. El WAL es importante en caso de que algo pase con el almacenamiento primario. Así que si el servidor se bloquea o sufre algún daño, se pueden recuperar todos los datos que tenía el servidor hasta el momento del desastre. También significa que si la escritura de la operación en el log falla la operación no se ejecutará sobre el servidor.

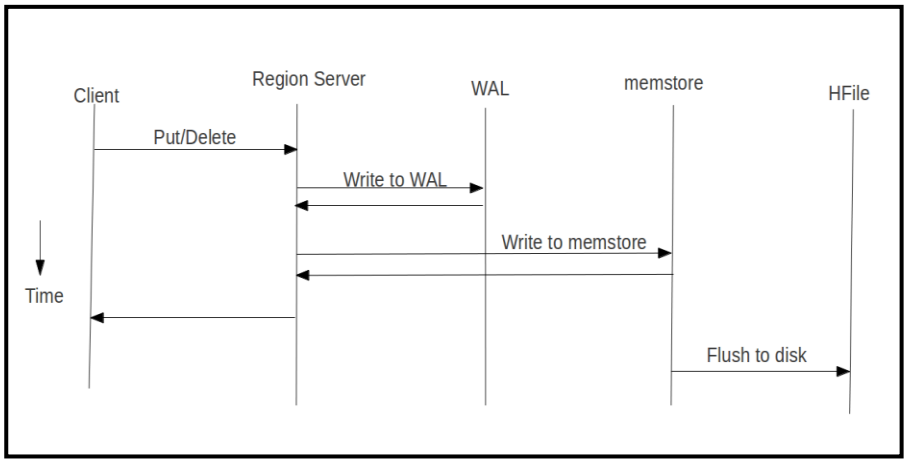


Figura 6. Esquema escritura en HBase

Esta información se utiliza en el caso de que haya un fallo del servidor para saber que registros se deberían haber escrito y no están. Una vez que los datos están escritos en el WAL se proceden a escribirlos en el MemStore. En primer lugar se comprueba si la MemStore está llena, en el caso de que lo estuviera se utiliza la memoria del disco. Se realiza una petición al HRegionServer con un nuevo hilo que se encarga de escribir los datos en un nuevo HFile situado en el HDFS. También guarda el último número de secuencia + para que el sistema sepa los datos existentes hasta el momento. Finalmente cuando el MemStore llega a cierto tamaño o después de un tiempo específico, de forma asíncrona los datos se conservan en el sistema de archivos. Durante todo el proceso los datos se guardan en la memoria volátil.

5.5. Réplicas

Las réplicas de Apache HBase proporcionan una forma de copiar los datos entre las implementaciones HBase. Puede servir como una solución de recuperación ante desastres y puede contribuir a proporcionar una mayor disponibilidad en la capa HBase. También

puede servir de manera más práctica, por ejemplo, como una manera de copiar fácilmente ediciones de un grupo orientado al web a un clúster de MapReduce que procesará los datos nuevos y viejos devolviendo los resultados automáticamente. Otra ventaja es el aumento del rendimiento mediante la distribución de la carga entre las diferentes réplicas.

El patrón de arquitectura utilizada para las réplicas de Apache HBase es (HBase clúster) master-push. Con él es mucho más fácil hacer un seguimiento de lo que está siendo reproducido actualmente ya que cada servidor de la región tiene su propia escritura write-ahead-log (WAL o HLog), al igual que otras soluciones conocidas, como la replicación MySQL maestro/esclavo donde sólo hay un registro de bin. Un grupo principal puede replicar a cualquier número de grupos de esclavos, y cada servidor de la región realizará su propio flujo de ediciones y réplicas.

Las réplicas se realizan de forma asíncrona, lo que significa que los grupos pueden estar geográficamente distantes, los vínculos entre ellos pueden estar desconectados por algún tiempo, y las filas insertadas en el clúster principal no estarán disponibles al mismo tiempo en los grupos de esclavos (consistencia eventual).

Los grupos que participan en la replicación pueden ser de tamaños asimétricos y el clúster master hará su mejor esfuerzo para equilibrar el flujo de replicación en los esclavos, apoyándose en la aleatoriedad. Existen tres formas de configurar las réplicas de HBase:

- **Master-Slave:** En este modo la replicación se realiza en una sola dirección, es decir, las transacciones de un clúster son empujados al otro clúster. Hay que tener en cuenta que el clúster esclavo es igual que cualquier otro clúster y pueden tener sus propios tablas, tráfico, etc.

- **Master-master:** En este modo, la replicación se envía en ambas direcciones para las mismas o diferentes tablas, es decir, las agrupaciones actúan como maestro y esclavo. En el caso de que estén replicando la misma tabla, uno puede pensar que puede conducir a un bucle sin fin, pero esto es evitado estableciendo la ClusterID de un cambio (Put / Eliminar) para la ClusterID del clúster originario.

- **Cíclico:** En este modo, hay más de dos clústers HBase que participan en la configuración de las réplicas, y uno puede tener varias posibles combinaciones de maestro-esclavo y maestro de maestros establecen entre los dos grupos.

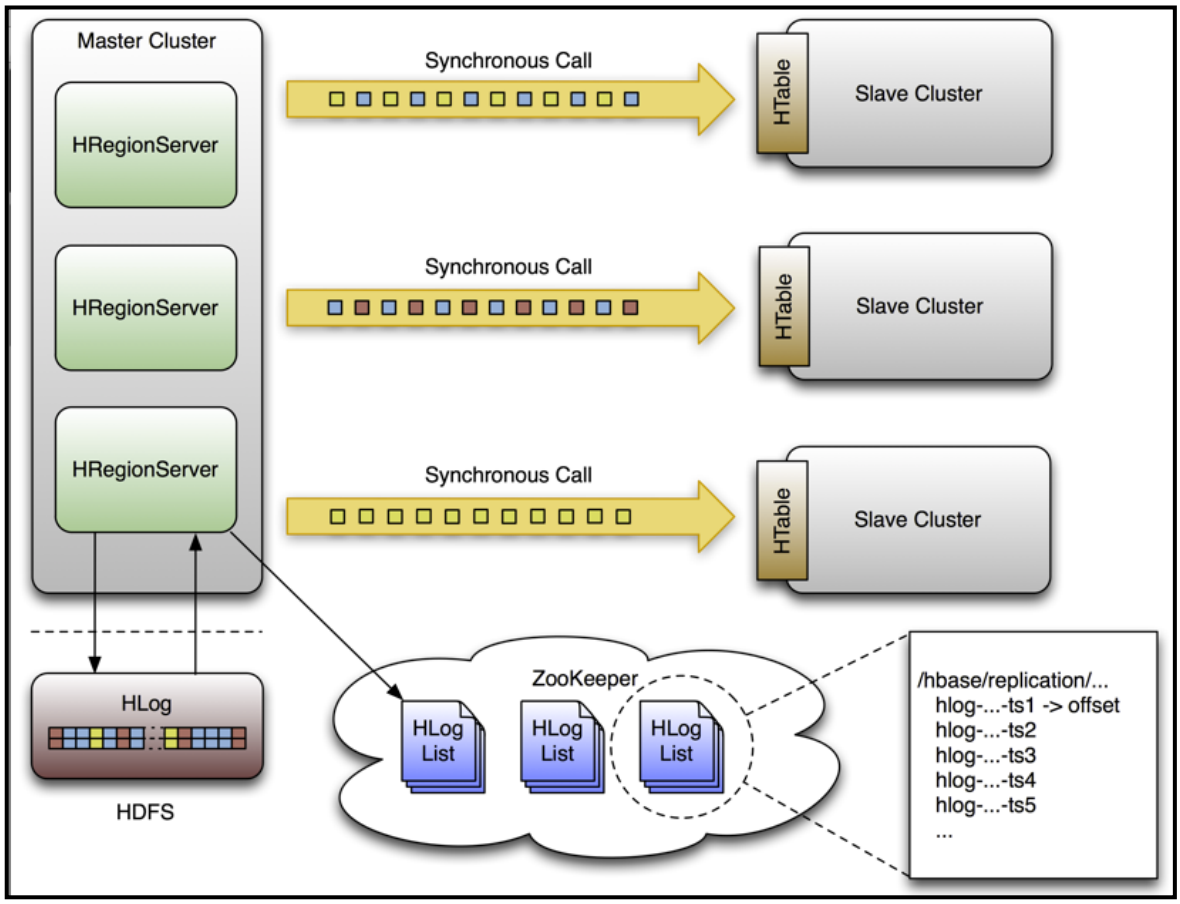


Figura 7. Arquitectura HBase

Cuando un Maestro de RegionServer inicia una réplica en un clúster de esclavo, esta primero se conecta al ZooKeeper del esclavo utilizando la clave de clúster proporcionada (esa clave se compone del valor de hbase.zookeeper.quorum, zookeeper.znode.parent y HBase. zookeeper.property.clientPort). A continuación, analiza los directorios "rs" para descubrir todos los sumideros disponibles (servidores región que están aceptando los flujos de entrada de ediciones de réplicas) y elegirá al azar un subconjunto de ellos utilizando una relación configurada (que tiene un valor por defecto de 10 %). Por ejemplo, si un grupo de esclavos cuenta con 150 máquinas, 15 serán elegidas como beneficiarias potenciales de ediciones que el clúster RegionServer va a enviar. Puesto que esto se hace mediante el clúster, la probabilidad de que se utilicen todos los RegionServers es muy alta, y este método funciona para grupos de cualquier tamaño. Por ejemplo, un clúster maestro de 10 máquinas de replicación a un clúster esclavo de 5 máquinas con una relación de 10% significa que el clúster RegionServer maestro elije una máquina al azar y por tanto la posibilidad de solapamiento y el uso completo de la agrupación del esclavo superior.

Un observador ZooKeeper se coloca en el nodo ${ zookeeper.znode.parent }/RS de la agrupación esclavo por cada uno de los servidores de la región de clúster principal . Este reloj se utiliza para monitorear los cambios en la composición de la agrupación esclavo. Cuando los nodos se eliminan del grupo de esclavos, los servidores de la región del clúster

principal responderán al seleccionar un nuevo grupo de servidores región esclavo para replicar.

Siempre y cuando los servidores región no fallen, el seguimiento de los registros en ZK no añade ningún valor. Desafortunadamente estos fallan y como ZooKeeper es altamente disponible se puede contar con él y su semántica para ayudar a gestionar las transferencias de las colas.

Todas las fuentes de clústers maestros mantienen un vigilante en cada uno de ellos para ser notificado cuando uno muere (al igual que el maestro lo hace). Cuando esto sucede, se crea un znode llamado "lock" dentro de la RegionServer que contiene las colas. El que lo crea con éxito procederá a la transferencia de todas las colas para su propio znode (uno por uno desde ZK no soporta la operación de cambio de nombre), y eliminará todos los viejos cuando esté hecho. Las colas recuperadas serán nombradas por Znodes con el id del grupo esclavo adjunto con el nombre del servidor muerto.

Una vez hecho esto, el maestro del clúster RS creará un nuevo hilo fuente para la cola copiada, y cada uno de ellos seguirán el patrón de lectura/filter/buque. La principal diferencia es que esas colas nunca tendrán nuevos datos, ya que no pertenecen a su nuevo RegionServer, lo que significa que cuando el lector llegue al final del último registro, el znode de la cola se eliminará y los maestros de clúster RS cerrarán esa fuente de duplicación.

5.6. HBase filesystems

A continuación se listan toda la jerarquía de directorios que conforman el sistema de ficheros de HBase,

- **bin:** directorio binario contiene los scripts proporcionados por HBase para iniciar y detener HBase, ejecutar daemons separados, o iniciar nodos maestros adicionales.
- **conf**: El directorio de configuración contiene los archivos que definen cómo HBase se ejecuta.
- **docs:** Este directorio contiene una copia de la página web del proyecto HBase, incluyendo la documentación de todas las herramientas, la API, y el proyecto en sí. Abrir el navegador web y abrir el archivo docs/index.html, ya sea arrastrándolo en el navegador, hacer doble clic en ese archivo, o utilizando el menú Archivo → Abrir (o nombre similar).
- **HBase – webapps:** HBase tiene interfaces de usuario basadas en la Web que se ejecutan como aplicaciones Web en Java, utilizando los archivos que se encuentran en este directorio.

- **lib:** Aplicaciones basadas en Java son por lo general un conjunto de muchas librerías auxiliares más el archivo JAR que contiene el programa en sí. Todas estas bibliotecas se encuentran en el directorio lib.

- **logs:** Dado que los procesos HBase se inician como demonios (es decir, que se están ejecutando en segundo plano del sistema operativo cumpliendo con su deber) usan los archivos de registro para informar de su estado, el progreso y opcionalmente, los errores que se producen durante su ciclo de vida.

- **src:** directorio de las fuentes que contiene todo lo necesario para crear una propia versión.

5.7. Modos de ejecución

- **Modo single node:** En este modo HBase no utiliza HDFS. Utiliza el sistema de archivos local y ejecuta todos los demonios de HBase y un ZooKeeper local en el mismo proceso de JVM. ZooKeeper se une a un conocido puerto para que los clientes pueden interactuar con HBase.

- **Modo pseudo-distribuido:** es un modo distribuido que se ejecuta en un solo host.

- **Modo distribuido**: los modos distribuidos requieren una instancia del sistema de archivos distribuido Hadoop (HDFS).

5.8. Ficheros de configuración

- **hbase-site.xml** y **hbase-default.xml**: Al igual que en Hadoop, es donde se agregan las configuraciones de HDFS. Si se realizan cambios de configuración relacionados con el HDFS en el Hadoop clúster, HBase no tendrá conocimiento de estas propiedades a menos que haga una de las siguientes:

    - Añadir un puntero a su HADOOP_CONF_DIR a la variable de entorno HBASE_CLASSPATH en HBase-env.sh.

    - Añada una copia de hdfs-site.xml (o Hadoop-site.xml) o, mejor, enlaces simbólicos, menos de $ {HBASE_HOME}/conf.

    - Agregar a HBase-site.xml directamente.

- **hbase-env.sh**: Establecer las variables de entorno HBase en este archivo. Los ejemplos incluyen las opciones para pasar a la JVM cuando un demonio HBase comienza,

tamaño de la pila y las configuraciones de colector de basura. También se puede configurar en este fichero las opciones para la configuración HBase, como directorios, opciones de SSH, donde localizar los archivos PID de proceso, y así sucesivamente. La ruta del archivo es conf/hbaseenv.sh. Cada opción está bastante bien documentada.

- **Regionserver**: Este archivo lista todos los nombres de servidor conocidos como región. Se trata de un archivo de texto plano que tiene un nombre de host por línea. La lista es usado por el script de mantenimiento HBase para iterar sobre todos los servidores para iniciar el proceso de servidor de la región.

- **log4j.properties:** Edite este archivo para cambiar la velocidad a la que se lanzan archivos HBase y para cambiar el nivel en el que HBase registra mensajes. Para los cambios aquí realizados será preciso reiniciar el clúster para que HBase note el cambio, aunque los niveles de registro se pueden modificar para demonios particulares a través de la interfaz de usuario HBase.

Capítulo 6. Cassandra

6.1. Introducción

Cassandra es una base de datos NoSQL, hoy en día es un proyecto de Apache. Fue desarrollado por Facebook en 2008 y liberado como un proyecto Open Source en Google code. En 2009 se convirtió en un proyecto "Apache Incubator", y finalmente en 2010 se convirtió en un proyecto TOP-LEVEL. Al igual que HBase también esta implementada en Java.

Cassandra está orientado a proporcionar alto rendimiento y escalabilidad lineal. Es capaz de manejar varios terabytes de datos si lo necesita y puede manejar millones de ficheros, incluso en un clúster pequeño. Es una base de datos descentralizada, esto significa que no existe la estructura maestro esclavo que existe en otra bases de datos como HBase. En Cassandra todos los nodos son idénticos y tienen la misma funcionalidad. Esta estructura es más fácil de implementar ya que no existe una estructura autoritaria como el maestro que puede convertirse en un punto de fallo (SPOF) cuya única solución es añadir más complejidad en el sistema creando maestros de repuesto. Gracias a esta estructura el sistema es más escalable y no requiere conocimiento específico para escalar el sistema. La arquitectura distribuida de Cassandra está basada en nodos que se comunican con un protocolo P2P con lo que la redundancia es máxima.

Su infraestructura está basada en Amazon's Dynamo y el modelo de datos en Google's BigTable. La información en las bases de datos relacionales se almacenan en forma de filas, pero en Cassandra la información se almacena en columnas con pares key-value y key-map para múltiples valores, que se agrupan en column families (familia de columnas). Las column families son fijas cuando la base de datos es creada, pero las columnas se pueden agregar a las column families en cualquier momento. Por otra parte,

las columns se agregan solo a las keys específicadas, por lo tanto diferentes keys pueden tener diferentes números de columnas en cualquier columns family.

6.2. Características

- **Tolerancia a fallos:** los datos son replicados en múltiples nodos miembros del clúster. Por tanto, en caso de que un nodo caiga, los datos que almacena éste no se pierden ya que otros nodos del clúster almacenan una copia de estos datos. Además el nodo que cae es remplazado de forma muy rápida, y por tanto, no se nota su ausencia al no haber prácticamente tiempos de inactividad.
- **Descentralización:** toda su estructura está basada en un clúster de nodos que son totalmente idénticos entre ellos, es decir, no hay una jerarquía de nodos. Al no tener un punto central donde se guarda la información, no existe un punto único de fallos y, por tanto, tampoco se producen cuellos de botella.
- **Flexibilidad:** por cada nodo que se añade al clúster, los through put de lectura y escritura o volúmenes de información que fluyen a través del clúster se ven incrementados de forma lineal.
- **Alta disponibilidad:** gracias a las características anteriores, la información está siempre disponible. Por eso es un sistema adecuado para aplicaciones que no pueden permitirse perder datos.

6.3. Arquitectura

La arquitectura de Cassandra tiene una alta complejidad ya que es un sistema de almacenamiento que necesita trabajar en un entorno de producción de alta disponibilidad. Además de la persistencia de datos, el sistema necesita asegurar una serie de características:

- Debe ofrecer soluciones escalables y robustas ante el balanceo de carga.
- Adhesión de miembros y detección de fallos.
- Recuperación ante caídas del sistema.
- Sincronización de réplicas. Sobrecarga de manejo.
- Soportar y manejar sobrecargas.
- Controlar el estado de transferencia.

- Concurrencia y planificación de tareas.
- Clasificación y enrutamiento de solicitudes.
- Sistema de monitorización y alarma.
- Configuración de gestión.

Los módulos que hacen de Cassandra un gran sistema de almacenamiento distribuido son: particionado, replicación, adhesión de miembros, manejo de caídas y escalado del sistema. Todos ellos trabajan en sincronía para manejar correctamente las peticiones de lectura y escritura.

Particionado

Una de las claves del diseño de Cassandra es la capacidad de escalado incremental. Para ello, es necesaria la partición de datos de manera dinámica en el conjunto de nodos del clúster. Cuando se inicia un clúster de Cassandra, se debe elegir como dividir los datos en los nodos del sistema. Esto se hace mediante la elección de una herramienta de particionado para el clúster.

En Cassandra toda la información manejada por el clúster se representa como un espacio circular o anillo. El circulo está dividido en tantos rangos como números de nodos haya y cada nodo es responsable de uno o más rangos de los datos globales. Antes de que un nodo nuevo pueda unirse al círculo se le debe asignar un token. Este token determina la posición que ocupa en el anillo y el rango de datos del que será responsable. Cada elemento de datos identificado por una clave es asignado a un nodo haciendo un hash de la clave del elemento de datos cediendo su posición en el anillo, y moviéndose hacia la derecha para encontrar el primer nodo con una posición mayor que la suya. Este nodo se considera el responsable de esa clave.

La aplicación específica la clave y Cassandra lo utiliza para enrutar las peticiones. De esta manera, cada nodo se convierte en responsable de la región en el anillo entre ella y su nodo predecesor. La principal ventaja de este “hashing consistente” es que, la llegada o la salida de un nodo sólo afecta a sus vecinos inmediatos y no a los demás nodos.

Replicación

Cassandra utiliza la replicación de datos para conseguir una alta disponibilidad y durabilidad. Cada elemento de datos se réplica en N nodos, donde N es el factor de replicación configurado. El coordinador se encarga de la replicación de los elementos de datos que caen dentro de su rango. Además, para almacenar localmente cada clave dentro de su rango, el coordinador réplica estas claves en los N-1 nodos del anillo.

Cassandra define tres tipos diferentes de políticas de réplicas desde la aplicación:

- **Rack Unaware:** Se escogen N-1 sucesores del nodo coordinador de esos datos, para almacenar las réplicas de los datos.

- **Rack Aware y Datacenter Aware:** Son dos estrategias mucho más complejas. Cassandra elige un líder entre sus nodos utilizando un sistema llamado Zookeeper. Todos los nodos que se unen al clúster contactan con el líder que les asigna de qué rangos son responsables. Además el líder procura que ningún nodo sea responsable de más de N-1 rangos del anillo.

Adhesión de miembros y detección de fallos

Cuando se inicia un clúster de Cassandra, se debe elegir como dividir los datos en los nodos del sistema. Esto se realiza eligiendo un particionador para el clúster.

Cassandra usa el protocolo Gossip para conocer la información sobre el estado y localización de otros nodos. Este es un protocolo de comunicación P2P gracias al cual los nodos intercambian información periódicamente sobre sí mismos y sobre otros nodos. Este proceso de intercambio de información se ejecuta cada segundo y cada nodo intercambia la información con otros tres nodos en el clúster. Un mensaje de gossip tiene asociada una versión, de forma que, si durante el intercambio de información se encuentra una actualización más reciente, este será reemplazado por el nuevo.

Cuando un nuevo nodo se une al clúster por primera vez, busca en su fichero de configuración a que clúster pertenece y a que nodos (seeds) debe conectarse para obtener la información para comenzar a trabajar. Esta información esta almacenada en el fichero de configuración Cassandra.yaml.

Para prevenir cortes de comunicación en el proceso gossip, todos los nodos deben tener la misma lista de nodos en su fichero de configuración. Esto es más crítico durante el arranque, la primera vez que se ejecuta, sin embargo en las posteriores el nodo recuerda con quien ha compartido los mensajes gossip.

La detección de fallos es un mecanismo mediante el cual un nodo localmente puede determinar si otro nodo del clúster ha caído o continúa activo. En Cassandra la detección de fallos se utiliza para evitar intentos de comunicación con nodos que son inalcanzables durante varias operaciones. Para ello, Cassandra utiliza una versión modificada del Detector de Fallos Acumulativo. La idea principal de este método es que el detector de fallos en lugar de devolver un valor booleano comunicando si el nodo está activo o no, devuelve un valor que representa el nivel de sospecha de cada nodo monitorizado.

Este valor es definido como ϕ y se expresa en una escala que se ajusta dinámicamente para reflejar la red y las condiciones de carga de los nodos monitorizados.

Arranque de nodos

Cuando un nodo arranca por primera vez, escoge un token que es equivalente a su posición en el anillo. La información del token es transmitida por todo el clúster de manera que, todos los nodos saben su posición y la del resto de nodos en el anillo. Esto permite que cualquier nodo pueda enrutar una petición sobre una clave al nodo correcto.

En el momento del arranque, el nodo lee su fichero de configuración, el cual contiene una lista con una serie de puntos de acceso para unirse al clúster que se denominan semillas del clúster.

Los fallos en los nodos pueden ser causados por diferentes motivos como por ejemplo fallos de discos, mal comportamiento de la CPU, fallo en el suministro eléctrico, fallo en la red, etc. Raramente una caída de un nodo representa una salida definitiva del clúster y, por tanto, no resulta un rebalanceo en la asignación de la partición o la reparación de réplicas inalcanzables. De la misma manera, un error manual podría ocasionar la puesta en marcha involuntaria de nuevos nodos de Cassandra. A tal efecto, cada mensaje contiene el nombre del clúster de cada instancia de Cassandra.

Si un error manual en la configuración provoca que un nodo intente unirse a una instancia de Cassandra incorrecta, el intento puede ser frustrado comprobando el nombre del clúster.

Por estas razones, se consideró apropiado utilizar un mecanismo explícito para iniciar la adición y eliminación de nodos de una instancia de Cassandra. Un administrador utiliza una herramienta de línea de comandos o un navegador para conectarse a un nodo de Cassandra y emitir un cambio de adhesión a unirse o abandonar el clúster.

Escalado del clúster

Cuando un nuevo nodo se añade al sistema, se le asigna un token que le permite aliviar a un nodo muy cargado. Esto se traduce en una nueva división de rangos de los que algunos nodos eran responsables anteriormente.

El algoritmo de arranque de Cassandra se inicia desde cualquier otro nodo en el sistema por un operador utilizando la herramienta de línea de comandos. La experiencia práctica muestra que los datos pueden ser transferidos a una tasa de 40 MB/s. Actualmente, se está trabajando para mejorar este aspecto, tratando de tener varias réplicas participando en la transferencia del arranque, consiguiendo paralelizar el esfuerzo de manera similar al caso de la herramienta Bittorrent.

6.4. Modelo de datos

Cassandra no soporta un modelo de datos relacional completo. En su lugar, proporciona clientes con un modelo de datos simple que soportan un control dinámico sobre la disposición de los datos y el formato. Dentro del modelo de datos que define Cassandra se pueden encontrar cinco tipos de estructuras de datos.

Column

Se define como una tupla (triplete) que contiene un nombre, un valor y un timestamp. Esta es la estructura o contenedor de datos mínimo que existe en Cassandra. A continuación, se puede ver una representación en Java del concepto de Column:

```
public class Column {
      Byte[] name;
      Byte[] value;
      Long timestamp;
}
```

Se puede representar mediante una clase de datos, llamada Column, que contiene tres atributos:

- **El atributo name** es un array de bytes que contiene el nombre o clave mediante la cual es posible indexar el dato.
- **El atributo value** es un array de bytes que contiene el valor.
- **El atributo timestamp** es de tipo Long e indica el instante de tiempo en el que fue almacenado el dato.

SuperColumn

Es una tupla con un nombre y un valor, pero no tiene timestamp a diferencia de la Column. En este caso, el campo value no es un valor binario sino que es un mapa que contiene combinaciones de clave/columna. Lo importante en este caso, es que la clave tiene el mismo valor que el nombre de la columna a la que se refiere. Se puede decir que una SuperColumn es un contenedor de una o más Columns.

A continuación, se puede observar una representación en Java del concepto de SuperColumn:

```
public class SuperColumn {
      Byte[] name;
      // La key es igual al nombre de la Column
      Map<Byte[], Column> value = null;
}
```

Se puede representar mediante una clase de datos que se llama SuperColumn que contiene dos atributos:

- **El atributo name** es un array de bytes que contiene el nombre o clave mediante la cual se podrá indexar la SuperColumn.
- **El mapa de Columns** se encuentra formado por una pareja de clave (array de bytes) y Column.

ColumnFamily

Es una estructura que puede tener un número infinito de filas. Es un concepto similar al de las tablas de las bases de datos relacionales. Toda ColumnFamily cuenta con un nombre y un mapa con una clave/valor. El valor del mapa es otro mapa que contiene Columns. El mapa con las Columns sigue las mismas reglas que la SuperColumn, donde la clave tiene el mismo valor que el nombre de la Column que indexa.

A continuación, se puede ver una representación en Java del concepto de ColumnFamily:

```
public class ColumnFamily {
     Byte[] name;
     // La key es generada por el usuario
     Map<Byte[] ,
     // La key es igual al nombre de la Column
      Map<Byte[],
     Column>> value = null;
}
```

Se puede representar mediante una clase de datos que se llama ColumnFamily que contiene dos atributos:

- El atributo name que es un array de bytes que contiene el nombre o clave mediante la cual se podrá indexar la ColumnFamily.
- El mapa de mapas que contienen Columns, formado por una pareja de clave (array de bytes) y un mapa.

SuperColumnFamily

Es una de las estructuras de datos más grandes que existen en el modelo de datos de Cassandra. Añade una nueva dimensión al concepto de SuperColumn. Por tanto, en lugar de tener Columns dispone de un mapa de SuperColumns. De la misma manera, la clave del mapa que contiene las SuperColumns debe ser el mismo que el nombre de la SuperColumn a la que hace referencia.

A continuación, se puede observar una representación en Java del concepto de SuperColumnFamily:

```
public class SuperColumnFamily {
     Byte[] name;
     // La key es generada por el usuario
     Map<Byte[],
      // La key es igual al nombre de la SuperColumn.
     Map<Byte[],
     SuperColumn>> value = null;
}
```

Se puede representar mediante una clase de datos que se llama SuperColumnFamily y contiene dos atributos:

- **El atributo name** es un array de bytes que contiene la clave mediante la cual se podrá indexar la SuperColumnFamily.
- **El mapa de mapas** que contienen SuperColumns, formado por una pareja de clave (array de bytes) y un mapa.

Keyspace

Es la estructura más amplia del modelo de datos de Cassandra y, por tanto, almacena ColumnFamilies y SuperColumnFamilies. Es un concepto muy similar al de

Esquema de las bases de datos relacionales

Capítulo 7. Despliegue de Hadoop & HBase

7.1. JAVA

HBase y Hadoop están escritos en Java, por lo que es necesario tener instalado la máquina virtual de Java en las máquinas que se vayan a utilizar. No cualquier versión de Java SDK es compatible con las herramientas, es necesaria una versión igual o superior a la 1.6. La opción recomendada es la provista por Oracle, que se puede encontrar en su página oficial. En este proyecto la descarga de los paquetes/dependencias se realizará mediante el gestor de paquetes del sistema operativo una vez añadidos los repositorios necesarios. De esta forma se podrá automatizar la instalación mediante un script.

Para comenzar la instalación de Java primero se deben añadir los repositorios al sistema para poder descargar las dependencias. Para ello en una terminal del sistema escribimos los siguientes comandos.

```
echo "deb http://ppa.launchpad.net/webupd8team/Java/ubuntu
precise main" > /etc/apt/sources.list.d/webupd8team-
Java.list

echo "deb-src
http://ppa.launchpad.net/webupd8team/Java/ubuntu precise
main" >> /etc/apt/sources.list.d/webupd8team-Java.list
```

Se debe añadir una clave pública para estos repositorios y actualizar el gestor de paquetes.

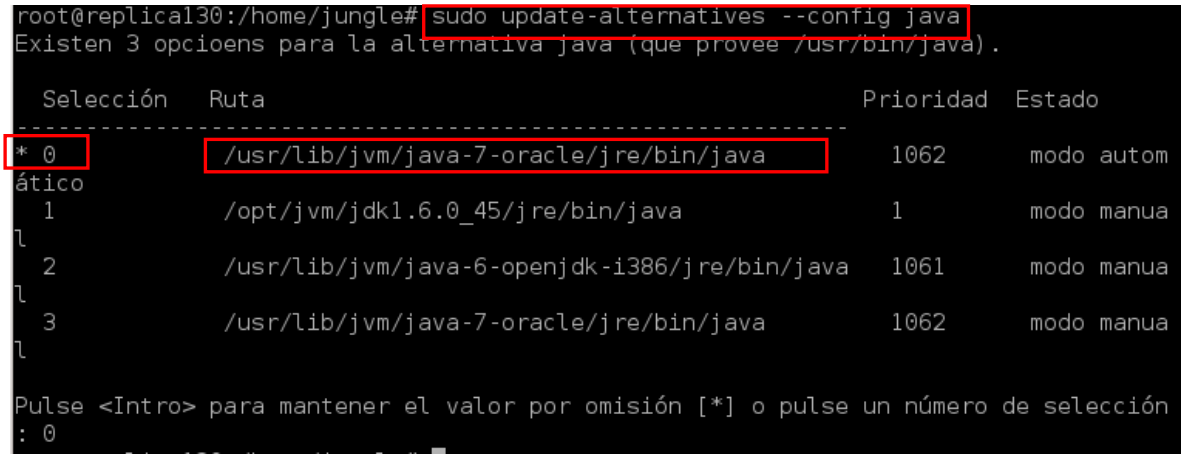
```
apt-key adv --keyserver keyserver.ubuntu.com --recv-keys
EEA14886

apt-get update
```

La versión elegida para instalar Java es la 7 (es la última versión estable existente en este momento).

```
apt-get install oracle-java7-installer
```

Para asegurar que la versión instalada se ha configurado para usarse por defecto, se puede ejecutar el siguiente comando.



Nota: Ejecutar todos los comandos con privilegios de administrador.

7.2. SSH y rsync

Otras herramientas imprescindibles son ssh y rsync que deben estar ejecutándose si se desea utilizar los scripts proporcionados para administrar Hadoop y HBase de forma remota. Un paquete de software de uso común es OpenSSH, disponible en su página web oficial. Para más información consultar los manuales del sistema operativo. Muchos sistemas operativos tienen mecanismos para instalar un paquete de la versión binaria ya compilado en lugar de tener que compilarlo manualmente.

SSH (Secure SHell, en castellano: intérprete de órdenes segura) es el nombre de un protocolo y del programa que lo implementa, y sirve para acceder a máquinas remotas a

través de una red. Permite manejar por completo la computadora mediante un intérprete de comandos, y también puede redirigir el tráfico para poder ejecutar programas gráficos si tenemos un Servidor (en sistemas Unix y Windows) corriendo.

Además de la conexión a otros dispositivos, SSH nos permite copiar datos de forma segura (tanto archivos sueltos como simular sesiones FTP cifradas), gestionar claves RSA para no escribir claves al conectar a los dispositivos y pasar los datos de cualquier otra aplicación por un canal seguro tunelizado mediante SSH. El comando de instalación es:

```
apt-get install ssh
```

Rsync es una aplicación libre para sistemas de tipo Unix y Microsoft Windows que ofrece transmisión eficiente de datos incrementales, que opera también con datos comprimidos y cifrados. Mediante una técnica de delta encoding30, permite sincronizar archivos y directorios entre dos máquinas de una red o entre dos ubicaciones en una misma máquina, minimizando el volumen de datos transferidos. Una característica importante de rsync no encontrada en la mayoría de programas o protocolos es que la copia toma lugar con sólo una transmisión en cada dirección. Rsync puede copiar o mostrar directorios contenidos y copia de archivos, opcionalmente usando compresión y recursión.

Actuando como un daemon de servidor, rsync escucha por defecto el puerto TCP 873, sirviendo archivos en el protocolo nativo rsync o vía un terminal remoto como RSH o SSH. En el último caso, el ejecutable del cliente rsync debe ser instalado en el host local y remoto. El comando para instalar el protocolo es apt-get install rsync.

Se debería poder acceder a todos los nodos por ssh sin contraseña. Para ello se deberá generar un par de claves públicas y añadirlas en cada servidor para que los scripts puedan acceder a los servidores remotos sin necesidad de intervención.

Generar el par de claves ssh en el equipo.

```
ssh-keygen
```

Copiar la clave pública al servidor.

```
scp /root/.ssh/id_rsa.pub root@localhost
```

Conectar con el servidor, añadir la clave pública a authorized_keys y eliminar la copia subida.

```
ssh root@localhost

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

rm /root/.ssh/id_rsa.pub
```

A continuación modificar la configuración del servidor ssh para asegurar que la autenticación de clave pública está habilitada. Los siguientes atributos deben estar activos en el fichero "/etc/ssh/sshd_config" (en caso contrario cambiar).

```
RSAAuthentication yes

PubKeyAutentication yes
```

Desactivar el ipv6. Para ello editar el fichero "/etc/sysctl.conf" añadiendo el siguiente texto.

```
#disable ipv6

net.ipv6.conf.all.disable_ipv6 = 1

net.ipv6.conf.default.disable_ipv6 = 1

net.ipv6.conf.lo.disable_ipv6 = 1
```

Para que el cambio funcione es necesario reniciar el sistema.

7.3. Hadoop

HBase en modo distribuido requiere de la instalación en modo distribuido de Hadoop. Para descargar Hadoop, basta con ir a la página de descarga de Apache y descargar el binario. Otra alternativa es la descarga e instalación automática añadiendo los repositorios de Cloudera al sistema.

```
echo "deb
http://archive.cloudera.com/cdh4/debian/squeeze/amd64/cdh
squeeze-cdh4 contrib

deb-src
http://archive.cloudera.com/cdh4/debian/squeeze/amd64/cdh
squeeze-cdh4 contrib " | tee
/etc/apt/sources.list.d/cloudera.list
```

Para continuar con la instalación es necesario agregar al sistema la librería cURL. Esta es una herramienta para usar en un intérprete de comandos para transferir archivos con sintaxis URL, soporta certificados HTTPS, HTTP POST, HTTP PUT, subidas FTP, Kerberos, subidas mediante formulario HTTP, proxies, cookies, autenticación mediante usuario y contraseña (Basic, DIgest, NTLM y Negotiate para HTTP y kerberos4 para FTP), continuación de transferencia de archivos, tunneling de proxy http y muchas otras

prestaciones. El principal propósito y uso para cURL es automatizar transferencias de archivos o secuencias de operaciones no supervisadas.

```
apt-get install curl

curl -s http://archive.cloudera.com/debian/archive.key |
apt-key add -

apt-get update
```

Instalar Hadoop y componentes necesarios para el correcto funcionamiento de HBase.

```
apt-get install Hadoop-0.20

apt-get install Hadoop-0.20-namenode

apt-get install Hadoop-0.20-datanode

apt-get install Hadoop-0.20-jobtracker

apt-get install Hadoop-0.20-tasktracker
```

Una vez realizada la instalación, el siguiente paso es la configuración de Hadoop. En primer lugar se debe añadir la ruta de Java. Agregar el texto en el fichero de configuración de Hadoop que se encuentra en la ruta "/etc/Hadoop/conf/Hadoop-env.sh".

```
export JAVA_HOME=/usr/lib/jvm/Java-7-oracle/
export HADOOP_NAMENODE_USER="hdfs"
export HADOOP_SECONDARYNAMENODE_USER="hdfs"
export HADOOP_DATANODE_USER="hdfs"
export HADOOP_TASKTRACKER_USER="hdfs"
export HADOOP_JOBTRACKER_USER="hdfs"
```

Es aconsajeble añadir las IPs de todos las máquinas que se van a utilizar asignándoles un nombre. Esta configuración se debe escribir en el fichero "/etc/hosts".

```
192.168.1.30 replica130 #maestro
192.168.1.31 replica131 #esclavo 1
192.168.1.32 replica132 #esclavo 2
192.168.1.33 replica133 #esclavo 3
192.168.1.34 replica134 #esclavo 4
192.168.1.35 replica135 #esclavo 5
```

Para indicar quien es el maestro hay que editar el fichero "/etc/Hadoop/conf/masters" y añadir la dirección IP del maestro o bien el nombre de la máquina (si se ha configurado en el fichero de "/etc/hosts") de la siguiente manera.

```
replica130
```

Es necesario editar el fichero "/etc/Hadoop/conf/core-site.xml" indicando la dirección IP que tiene el maestro o su equivalente nombre y el puerto en el que estará activo. En este caso el maestro será "replica130" y estará activo en el puerto 8020.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
href="configuration.xsl"?>
<configuration>
     <property>
          <name>fs.default.name</name>
          <value>hdfs://replica130:8020</value>
     </property>
</configuration>
```

En el siguiente paso se debe crear una nueva carpeta, que puede tener cualquier nombre, pero para comodidad es aconsejable el nombre "Hadoop" y proporcionarle los permisos necesarios.

```
mkdir /Hadoop && chown hdfs:hdfs /Hadoop && chmod 777
/Hadoop
```

Editar el fichero "/etc/Hadoop/conf/hdfs-site.xml" con el siguiente texto.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
href="configuration.xsl"?>
<configuration>
     <property>
          <name>dfs.replication</name>
          <value>5</value>
     </property>
     <property>
          <name>dfs.permissions</name>
          <value>false</value>
     </property>
     <property>
          <name>dfs.datanode.max.xcievers</name>
          <value>4096</value>
     </property>
</configuration>
```

Editar también el fichero "/etc/Hadoop/conf/slaves" añadiendo las IPs de los esclavos o el nombre correspondiente (en el caso de que se haya indicado en el fichero "/etc/hosts" la IP y el nombre):

```
replica131
replica132
replica133
replica134
replica135
```

7.4. HBase

Para instalar HBase se puede ejecutar en una consola el siguiente comando o bien se puede descargar el binaro desde la página oficial.

```
sudo apt-get install Hadoop-hbase
```

Al igual que en Hadoop, hay que indicar quien es el maestro para HBase. Para ello se debe editar el fichero "/etc/hbase/conf/hbase-site.xml" añadiendo el siguiente xml.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
href="configuration.xsl"?>
<configuration>
     <property>
          <name>hbase.cluster.distributed</name>
          <value>true</value>
     </property>
     <property>
          <name>hbase.rootdir</name>
          <value>hdfs://replica130:8020/hbase</value>
     </property>
     <property>
          <name>hbase.master</name>
          <value>replica130:60000</value>
     </property>
     <property>
          <name>hbase.zookeeper.quorum</name>
          <value>replica130</value>
     </property>
     <property>
          <name>hbase.regionserver.port</name>
          <value>60020</value>
     </property>
</configuration>
```

Por último, para finalizar la configuración de HBase hay que indicarle la ruta de Java así como otras preferencias y/o configuraciones. Para ello abrir el fichero "/etc/hbase/conf/hbase-env.sh" y añadir el siguiente texto.

```
export JAVA_HOME=/usr/lib/jvm/Java-7-oracle/
export HBASE_REGIONSERVERS=/etc/hbase/conf/regionservers
export HBASE_LOG_DIR=/hbase/logs
export HBASE_PID_DIR=/hbase/pid
export HBASE_MANAGES_ZK=true
```

En este proyecto se ha trabajado con máquinas virtuales por la falta de recursos, ya que no se disponia más que de dos laptops para su realización. Debido a la utilización de máquinas virtuales, es aconsajable configurar una de las máquinas y clonarla tantas veces como maestros y esclavos haya. Tras este paso hay que configurar las direcciones IP en cada una de las máquinas.

Una vez creadas, en la máquina virtual del maestro establecemos las siguientes configuraciones de Hadoop. Se debe editar el fichero "/etc/Hadoop/conf/mapred-site.xml" sustituyendo el texto existente por el siguiente:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
href="configuration.xsl"?>
<configuration>
     <property>
               <name>mapred.job.tracker</name>
               <value>replica130:8021</value>
     </property>
     <property>
          <name>mapred.local.dir</name>
          <value>/Hadoop/mapred/local</value>
     </property>
     <property>
          <name>mapred.map.tasks</name>
          <value>20</value>
     </property>
     <property>
          <name>mapred.reduce.tasks</name>
          <value>4</value>
     </property>
</configuration>
```

En la máquina virtual del maestro instalar los siguientes componentes de HBase:

```
sudo apt-get install Hadoop-hbase-master

sudo apt-get install Hadoop-hbase-regionserver

sudo apt-get install Hadoop-zookeeper-server
```

Ejecutar los siguientes comandos en el maestro:

```
cat /etc/Hadoop/conf/slaves >
/etc/hbase/conf/regionservers

sudo -u hdfs Hadoop fs -mkdir /hbase

sudo -u hdfs Hadoop fs -chown hbase /hbase
```

Para terminar con la configuración del sistema, en las máquinas virtuales de los esclavos hay que editar el fichero "/etc/Hadoop/conf/mapred-site.xml" sustituyendo el texto existente por el siguiente.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
href="configuration.xsl"?>
<configuration>
     <property>
          <name>mapred.job.tracker</name>
          <value>replica130:8021</value>
     </property>
</configuration>
```

Una vez realizada toda la configuración, el último paso para poner todo en marcha es realizar un formateo de los nodos. Por tanto, se debe ejecutar en una terminal del maestro el siguiente comando.

```
Hadoop namenode -format
```

Para iniciar el sistema, arrancar todos los nodos y ejecutar en el maestro el siguiente comando.

```
/user/lib/Hadoop/start-all.sh
```

Capítulo 8. Despliegue de Cassandra

8.1. JAVA

Al igual que en HBase y Hadoop, Cassandra también necesita Java para poder ejecutarse. La versión necesaria en este caso para que Cassandra funcione es la 7. Una vez más, la opción recomendada es la provista por Oracle, que se puede encontrar en la página oficial de Java. Para comenzar la instalación de Java primero se deben añadir los repositorios al sistema para poder descargar las dependencias. Para ello mirar capítulo 7.

Para descargar Cassandra, basta con ir a la página de descarga de Apache y descargarse el binario. Otra alternativa es la descarga e instalación automática añadiendo los repositorios de DataStax Comunity al sistema.

```
echo "deb http://debian.datastax.com/community stable
main" > /etc/apt/sources.list.d/cassandra.sources.list
```

El siguiente paso es descargar la key para validar el repositorio creado anteriormente.

```
curl -L http://debian.datastax.com/debian/repo_key | sudo
apt-key add -
```

Para finalizar con la instalación, descargar e instalar python-cli y Cassamdra. Python-cli es necesario para poder instalar y ejecutar Cassandra.

```
sudo apt-get update

sudo apt-get install python-cql

sudo apt-get install cassandra
```

Si hubiera existido una instalación o versión previa de Cassandra en el sistema, es conveniente realizar una limpieza de las carpetas. Para ello se recomienda ejecutar los siguientes comandos en la consola.

```
sudo rm -rf /var/lib/cassandra/data/system/*

sudo rm -rf /var/lib/cassandra/log/*
```

Es importante asegurarse de que la carpeta "/var/lib/cassandra" y "/var/log/cassandra" tengan permisos de lectura y escritura. Por defecto, en la instalación se le otorgan permisos por lo que no es necesario hacerlo manualmente. Pero en caso de que el arranque de Cassandra falle es interesante comprobar que las carpetas antes mencionadas tienen los permisos adecuados.

La configuración de Cassandra requiere definir el nombre del host. Esta configuracion se debe escribir en el fichero "/etc/hosts".

```
192.168.1.10        cassandra130
192.168.1.11        cassandra131
192.168.1.12        cassandra132
192.168.1.13        cassandra133
```

Una vez establecidos los nombres, modificamos el fichero de configuración para indicarle dónde tiene que buscar a los miembros del clúster y las propias características. En los siguientes cuadros se muestra la configuración de cada una de las máquinas utilizadas en el proyecto. Esta configuración se encuentra en el fichero "/etc/cassandra/cassandra.yaml".

Configuración máquina 1

```
cluster_name: 'pruebas_cassandra'
seed_provider:
     - seeds: "198.168.1.10"
listen_address: cassandra130
rpc_address: 198.168.1.10
endpoint_snitch: RackInferringSnitch
```

Configuración máquina 2

```
cluster_name: 'pruebas_cassandra'
seed_provider:
      - seeds: "198.168.1.10"
listen_address: cassandra131
rpc_address: 198.168.1.11
endpoint_snitch: RackInferringSnitch
```

Configuración máquina 3

```
cluster_name: 'pruebas_cassandra'
seed_provider:
      - seeds: "198.168.1.10"
listen_address: cassandra133
rpc_address: 198.168.1.13
endpoint_snitch: RackInferringSnitch
```

Configuración máquina 4

```
cluster_name: 'pruebas_cassandra'
seed_provider:
     - seeds: "198.168.1.10"
listen_address: cassandra132
rpc_address: 198.168.1.12
endpoint_snitch: RackInferringSnitch
```

Para comprobar que todos los nodos están funcionando correctamente y que todos tienen un token en el anillo se debe ejecutar:

```
nodetool status
```

Capítulo 9. Benchmarking

9.1. Objetivos

El objetivo que se pretende medir con este benchmarking es el rendimiento de los dos sistemas en cuanto a operaciones de lectura, inserciones, escaneo y actualizaciones.

El rendimiento mide la latencia de las solicitudes cuando una base de datos está bajo carga. Este es el nivel más básico, pero muy importante en la evaluación comparativa, ya que la primera pregunta antes de cualquier despliegue es la cantidad de carga del almacén de datos que se llevará por servidor. La latencia de solicitudes está fuertemente relacionada con el rendimiento del sistema. A medida que se intenta aumentar el rendimiento, la latencia crece gradualmente. Eventualmente, cuando el sistema alcanza la saturación, la latencia salta drásticamente y entonces no se puede aumentar el rendimiento.

9.2. Detalles y configuración del benchmarking

Configuración del sistema

Para la realización de este proyecto se han utilizada 4 máquinas virtuales empleando el software VirtualBox. Estas máquinas han sido distribuidas en dos anfitriones (dos máquinas por anfitrión).

El primer anfitrión es un laptop con las siguientes características:

- Procesador: Intel(R) Core(TM) i5 CPU M430 @ 2.27 GHz
- RAM: 4,00 GB (3,86 GB disponibles)

- Disco duro: 454 GB
- Sistema operativo: Windows 7 Home Premium (64 bits)
- Tarjeta de red 10/100.

El segundo anfitrión es un laptop con las siguientes características:

- Procesador: Intel(R) Core(TM) i7-3630QM CPU @ 2.40 GHz
- RAM: 4,00 GB (3,86 GB disponibles)
- Disco duro: 455 GB
- Sistema operativo: Windows 8.1 Home Premium (64 bits)
- Tarjeta de red 10/100

Cada uno de las máquinas virtuales tiene la siguiente configuración:

- RAM: 1,00 GB
- Disco duro: 20 GB
- Sistema operativo: Debian
- Cassandra 0.5.0 (0.6.0-beta2 for range queries)
- HBase 0.90
- Hadoop 0.20.2
- No replication; force updates to disk

Diseño del clúster

En HBase una de las máquinas trabaja como maestro en el clúster. Esta máquina es la encargada de la administración de los procesos.

- Replica130: 192.168.1.10

Las otras tres máquinas trabajan como esclavos.

- Replica131: 192.168.1.11
- Replica132: 192.168.1.12
- Replica133: 192.168.1.13

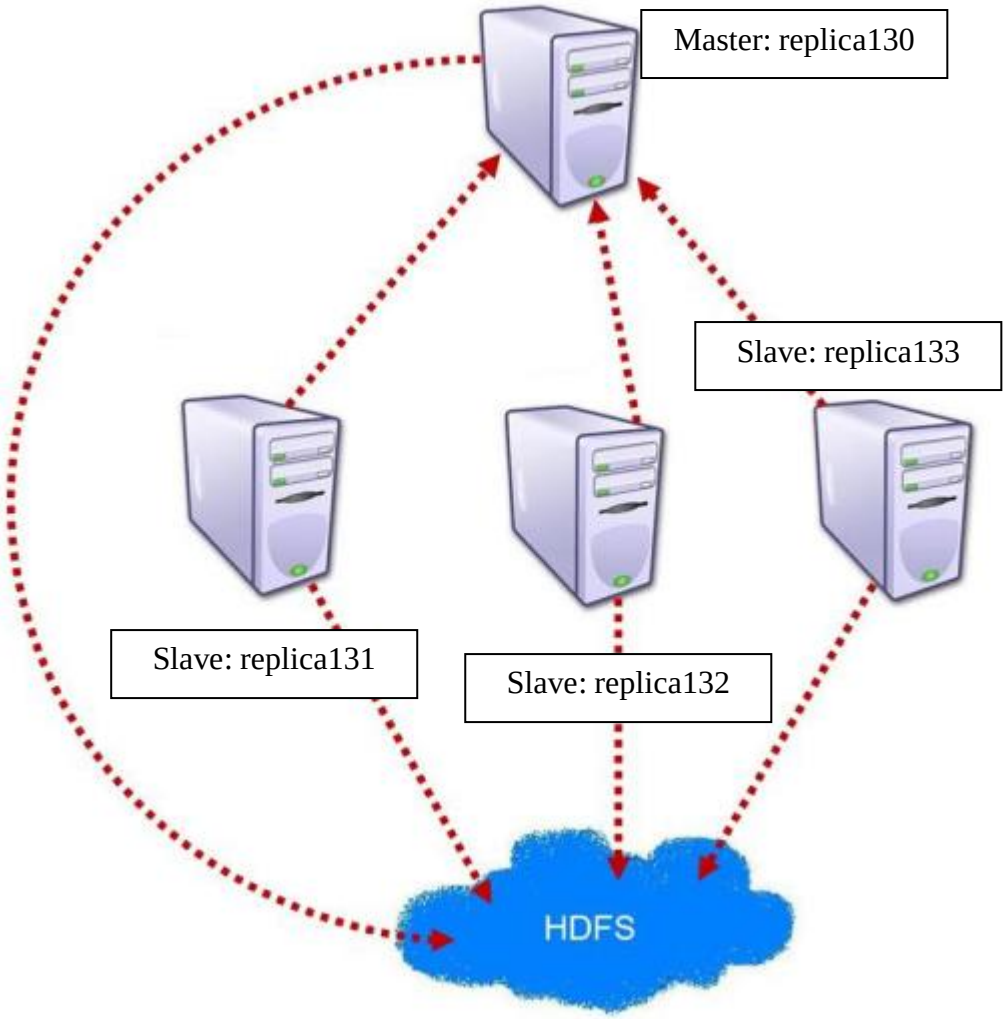


Figura 8. Clúster HBase

Workloads

- 700 000 registros de 1 KB cada uno en una máquina de 20 GB.
- 100 threads.
- Distribución: zipfian/uniform

Tipos de operaciones

- **Insert:** Añade un nuevo registro.
- **Update:** Cambia el valor de un registro existente en la base de datos.
- **Read:** Lee un registro de manera aleatoria.
- **Scan:** Escanea registros en orde, empezando por una clave de registro seleccionada al azar. El número de registros que deben analizarse también es seleccionado al azar entre 1 y 100.

9.3. Ejecución de pruebas

Antes de comenzar a ejecutar las pruebas es necesario crear, tanto en HBase como en Cassandra, las tablas sobre las que se ejecutarán las pruebas.

HBase

En primer lugar es necesario arrancar Hadoop y HBase:

```
Hadoop namenode -format

/usr/lib/Hadoop/bin/start-all.sh

/usr/lis/hbase/bin/start-hbase.sh
```

Una vez iniciados, creamos la tabla sobre la que se creará la carga de trabajo. Para ello, en un terminal, ejecutar lo siguiente:

```
hbase shell

create 'usertable' 'f1'
```

En este proyecto, para poblar la base de datos de HBase se ha utilizado la herramienta Sqoop, importando los datos de una base de datos PostgreSQL ya existente.

```
sqoop import --connect
jdbc:postgresql://192.168.1.10:5432/worlddb -table
usertable --username 'postgres' --password '' --hbase-
create-table --hbase-table usertable --column-family f1
```

- --connect jdbc:postgresql://192.168.1.10:5432/worlddb → indica la conexion a la base de datos y el nombre de la base de datos.
- -table usertable→ nombre de la table a importar
- username 'postgres' –password→ usuario y contraseña para conectar a la base de datos PostgreSQL .
- --hbase-create-table → crear la tabla en HBase en caso de que no exista
- --hbase-table usertable --column-family f → el nombre de la tabla y la columna a crear

Con la base de datos ya preparada se puede proceder a ejecutar YCSB con el siguiente comando.

```
bin/ycsb run hbase -P workloads/workloada -p
columnfamily=f1 -p table=usertable -p recordcount=700000 -
p threadcound=100 -target 1000 -s -t
```

Cassandra

Para iniciar Cassandra, se debe ejecutar el siguiente comando, siempre comenzando por las máquinas que estén actuando como sedes.

```
service cassandra start
```

Para crear la tabla usertable, ejecutar en una terminal el siguiente comando:

```
cassandra-cli --host nombre_de_la_maquina --port 9160
```

De esta manera se iniciará la shell de Cassandra donde se podrá crear la tabla:

```
 create keyspace usertable;

use usertable;

create column family data;

quit;
```

Para ejecutar YCSB sobre Cassandra se debe ejecutar el siguiente comando:

```
bin/ycsb run cassandra10 -P workloads/workloada -p
hosts='192.168.1.10,192.168.1.11,192.168.1.12,192.168.1.13
' -p recordcount=700000 -p threadcound=100 -target 1000
```

9.4. Workload A – Update heavy

En la primera prueba se ha utilizado la siguiente configuración.

```
recordcount=700000
operationcount=700000
workload=com.yahoo.ycsb.workloads.CoreWorkload
readallfields=true
readproportion=0.5
updateproportion=0.5
scanproportion=0
insertproportion=0
requestdistribution=zipfian
```

Actualizaciones - 50%

Throughputs	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
HBase latency (ms)	0,29	0,28	0,29	0,38	0,88	0,30	0,35	0,28	0,25	0,31
Cassandra latency (ms)	3,12	2,10	2,12	2,16	2,26	1,57	3,17	3,15	1,61	1,62

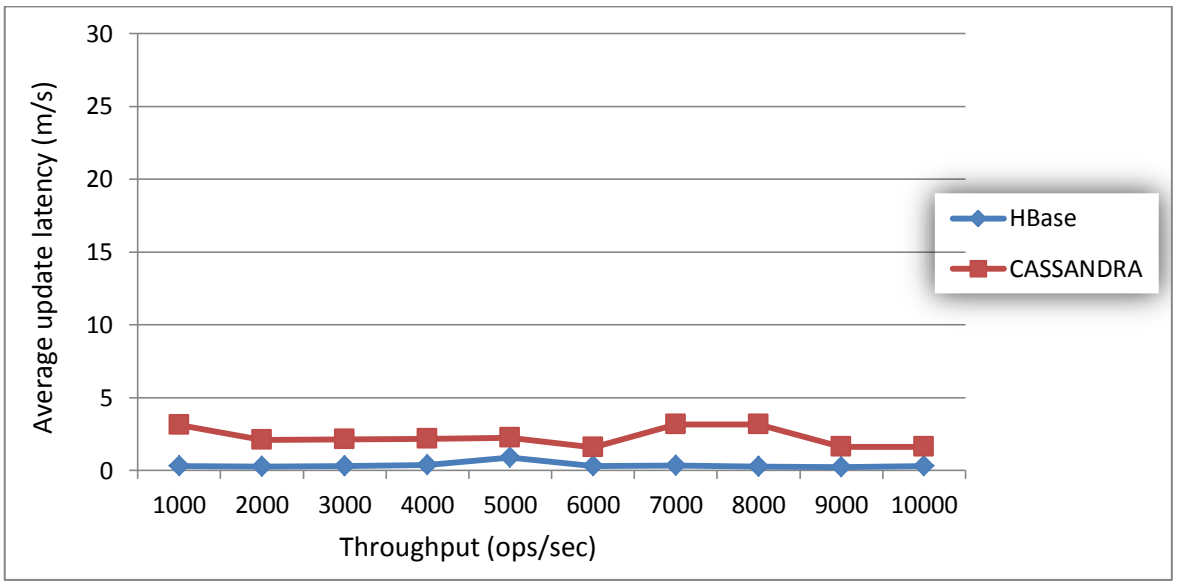


Figura 9. Workload A – Updates

Lectura- 50%

Throughputs	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
HBase latency (ms)	12,73	13,93	19,32	21,21	21,60	11,91	10,93	10,42	10,06	12,19
Cassandra latency (ms)	12,23	10,62	11,38	12,69	15,13	13,29	17,62	17,75	16,94	19,56

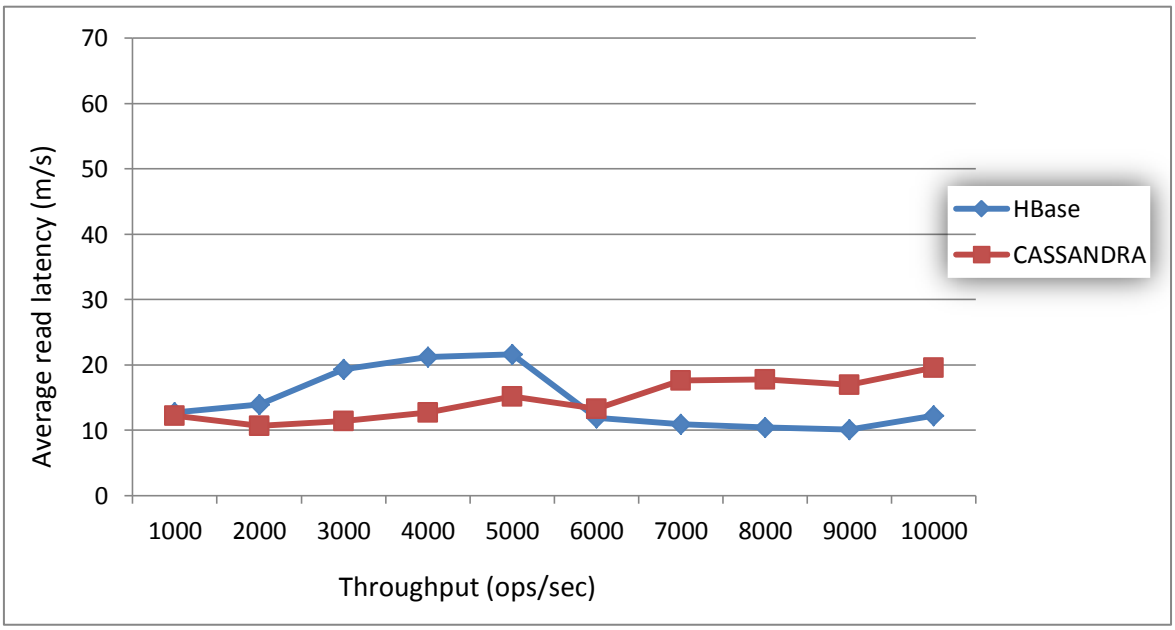


Figura 10. Workload A – Reads

La Figura 10.2 y la Figura 10.3 muestran la latencia frente a las operaciones de lectura y actualización formando las curvas de rendimiento de cada sistema. Cassandra está optimizado para las escrituras. Aunque HBase consigue unos resultados mejores que Cassandra; esto es debido a que HBase almacena las actualizaciones en la memoria volátil. Durante las actualizaciones, el promedio de latencia de las respuestas no superan los dos milisegundos en ninguno de los dos sistemas.

Para las operaciones de lectura sobre HBase, la latencia es a veces más alta debido a la necesidad de reconstruir registros. La latencia de Cassandra aumenta conforme aumentan los throughputs. El promedio de latencia en las lecturas de ambas bases de datos se encuentra entre 14 y 15ms.

9.5. Workload B — Read Heavy

En la segunda prueba se ha utilizado la siguiente configuración.

```
recordcount=700000
operationcount=700000
workload=com.yahoo.ycsb.workloads.CoreWorkload
readallfields=true
readproportion=0.95
updateproportion=0.05
scanproportion=0
insertproportion=0
requestdistribution=zipfian
```

Lectura – 95%

Throughputs	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
HBase latency (ms)	10,93	11,53	11,53	10,37	11,40	12,13	13,04	10,24	11,10	11,80
Cassandra latency (ms)	18,93	19,64	21,75	23,37	21,67	20,41	20,23	20,07	23,24	21,03

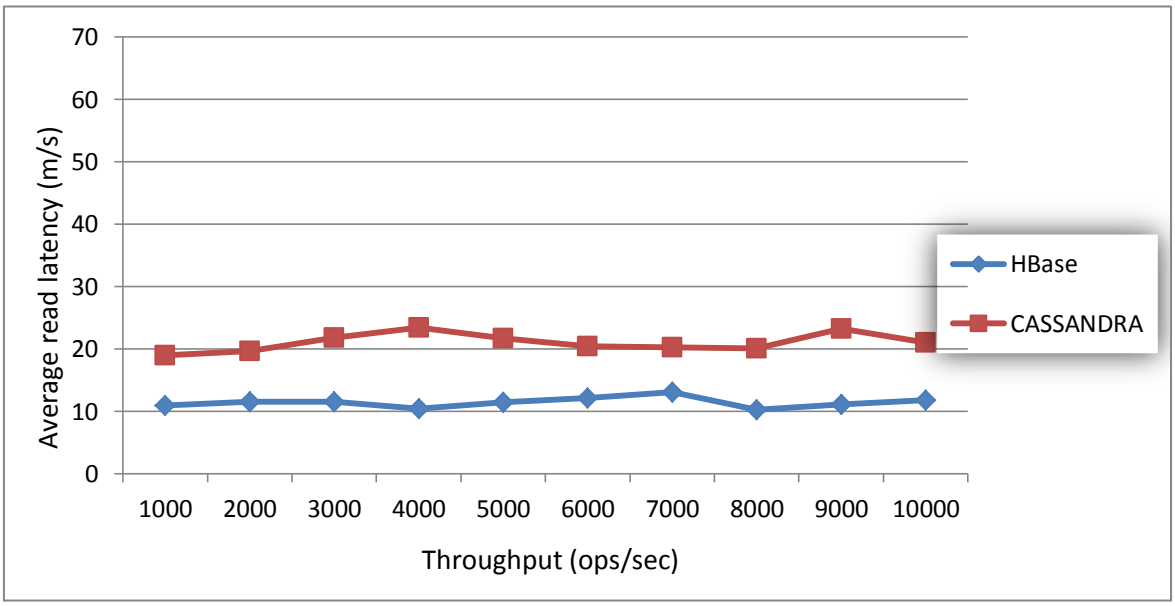


Figura 11. Workload B - Read

Actualizaciones – 5%

Throughputs	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
HBase latency (ms)	0,15	0,15	0,15	0,15	0,15	0,27	0,16	0,16	0,15	0,15
Cassandra latency (ms)	1,97	1,99	3,59	3,62	2,93	1,99	2,00	1,99	2,43	2,99

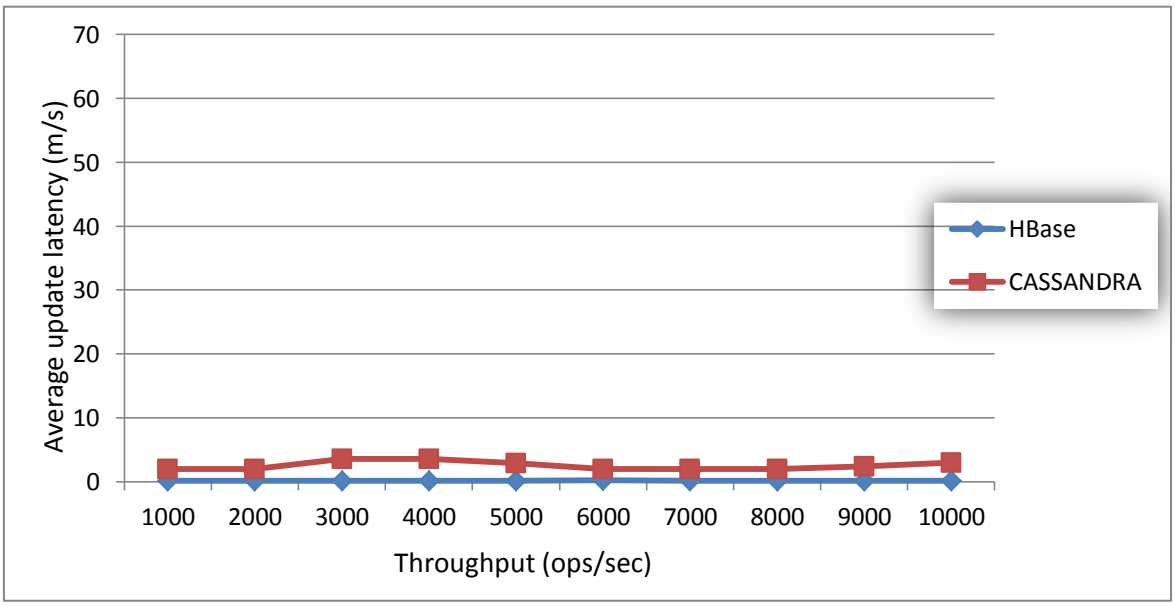


Figura 12. Workload B – Updates

Los resultados del workload B se muestran en las figuras 10.5 y 10.6. En este escenario la pesada carga de trabajo de la lectura proporciona una imagen diferente a la que se observa en el workload A. La latencia de lectura de HBase a veces es más alta debido a la aplicación de almacenamiento de registros estructurados y también porque tiene que reconstruir los fragmentos de registros de varias páginas de disco. HBase vuelca sus memtables a disco en archivos separados, y tiene que buscar cada uno de esos archivos, aunque sólo algunos contienen fragmentos relevantes. De hecho, se observa peor latencia y rendimiento cuando el número de archivos crece, y mejora cuando el número se reduce a través de la compactación. Las operaciones extras que realiza Cassandra para acoplar los registros para la lectura marcan su rendimiento en las lecturas. Cassandra muestra mayor latencia cuando el rendimiento es alto, para ello es necesario llevar al disco cerca de la saturación. Las actualizaciones de ambas bases de datos siguen proporcionando buenos resultados.

9.6. Workload E — Scan

En la tercera prueba se ha utilizado la siguiente configuración.

```
 recordcount=700000
operationcount=700000
workload=com.yahoo.ycsb.workloads.CoreWorkload
readallfields=true
readproportion=0
updateproportion=0
scanproportion=0.95
insertproportion=0.05
requestdistribution=zipfian
maxscanlength=100
scanlengthdistribution=uniform
```

Inserciones – 5%

Throughputs	100	200	300	400	500	600
HBase latency (ms)	1,87	1,56	1,72	0,63	2,01	2,52
Cassandra latency (ms)	4,06	2,10	3,99	4,01	4,04	4,01

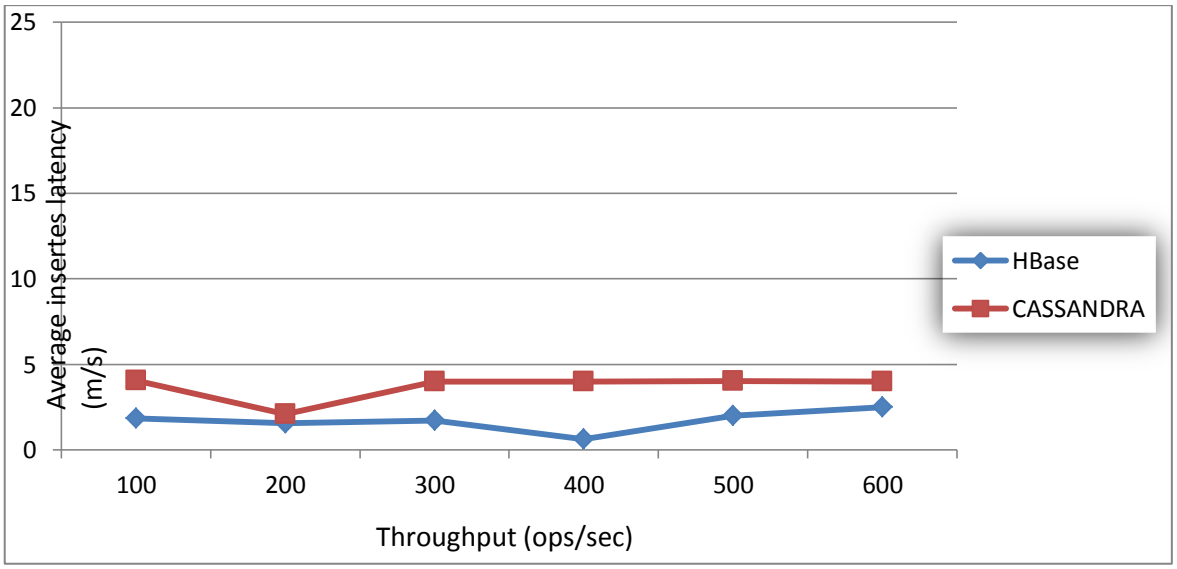


Figura 13. Workload E – Inserts

Escaneos – 95%

Throughputs	100	200	300	400	500	600
HBase latency (ms)	34,90	36,79	43,98	42,89	51,09	56,43
Cassandra latency (ms)	49,37	28,99	32,99	31,54	32,40	31,93

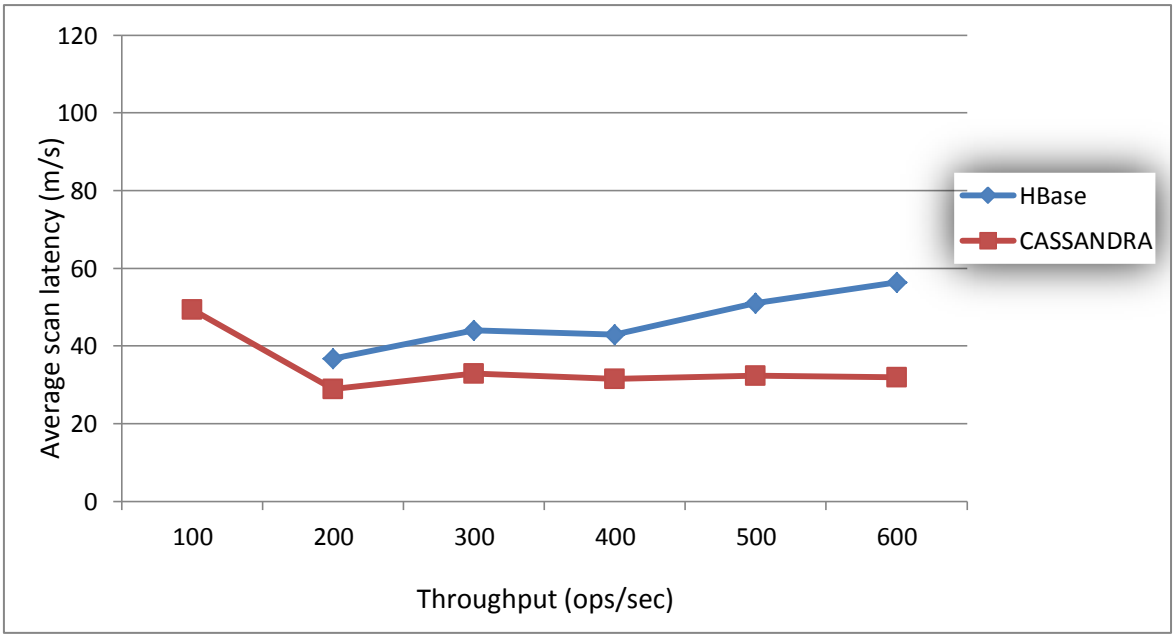


Figura 14. Workload E – Scan

Para esta prueba se utiliza la distribución uniforme la cual consiste en elegir al azar los registros que serán escaneados. Los resultados se muestran en las figuras 10.8 y 10.9. En esta carga de trabajo es interesante realizar las pruebas con distancias cortas para obtener una visión mejor de los resultados. Para los pequeños rangos, las consultas son similares a las búsquedas aleatorias.

En las inserciones se puede observar que al principio Cassandra tiene unos desajustes en la latencia pero luego empieza a ser constante. HBase tiene mejores resultados en las inserciones que Cassandra, debido a que almacena las actualizaciones en RAM. En cambio en los escaneos Cassandra tiene una latencia más baja que HBase, debido a que HBase tiene la necesidad de reconstruir los registros.

9.7. Workload F — Read-modify-write

En la cuarta prueba se ha utilizado la siguiente configuración.

```
recordcount=700000
operationcount=700000
workload=com.yahoo.ycsb.workloads.CoreWorkload
readallfields=true
readproportion=0.5
updateproportion=0
scanproportion=0
insertproportion=0
readmodifywriteproportion=0.5
requestdistribution=zipfian
```

Lectura-actualizaciones-escritura – 50%

Throughputs	100	200	300	400	500	600	700	800	900	1000
HBase latency (ms)	0,01	0,01	0,01	0,01	0,01	0,01	0,01	0,01	0,01	0,02
Cassandra latency (ms)	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,03	0,02	0,03

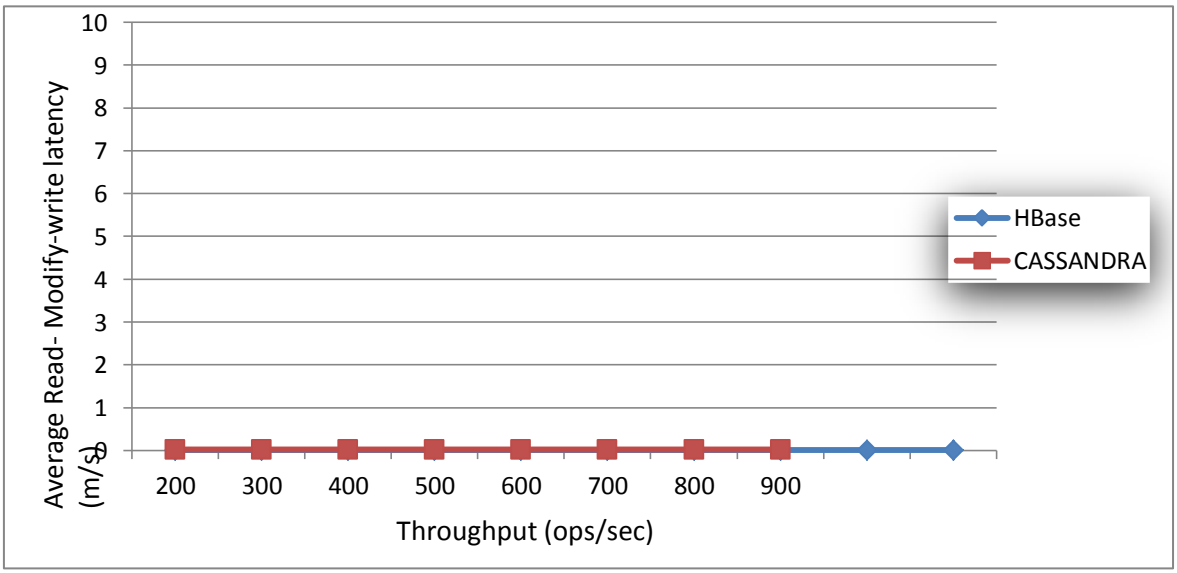


Figura 15. Workload F – Read-modify-write

Lectura – 50%

Throughputs	100	200	300	400	500	600	700	800	900	1000
HBase latency (ms)	9,62	9,60	9,51	11,01	12,92	14,50	14,02	13,76	13,63	14,67
Cassandra latency (ms)	-	16,57	16,94	17,92	19,64	19,94	19,94	23,01	23,13	-

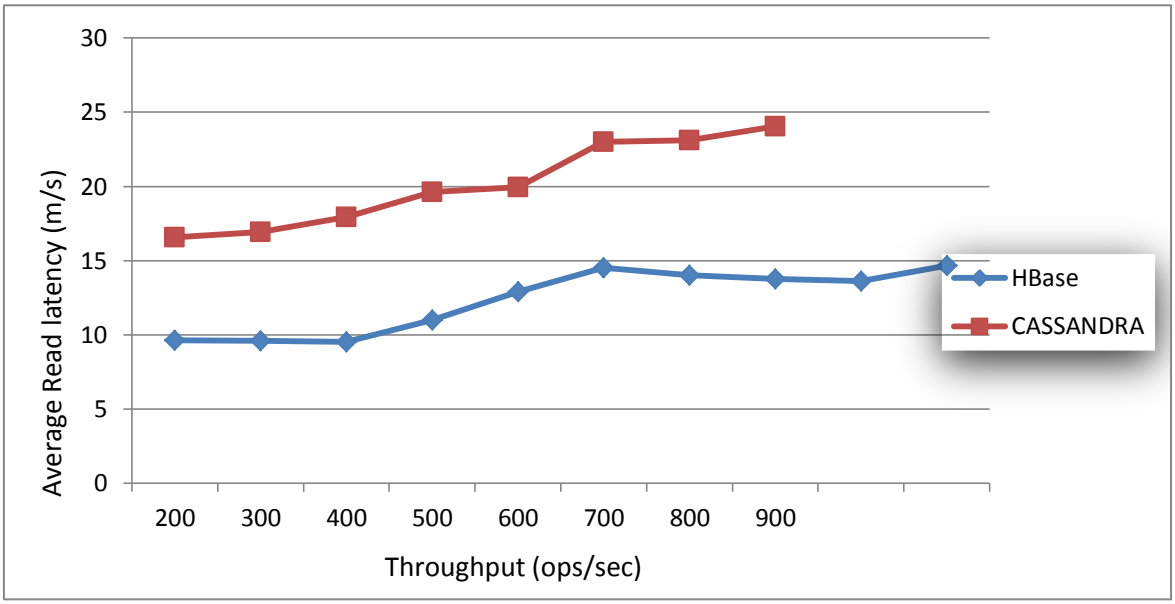


Figura 16. Workload F – Read

Estas pruebas se han realizado con unos números de throughputs bajos para observar más de cerca el comportamiento de cada sistema. Se crea un escenario real de pruebas con operaciones aleatorias de lectura, actualizaciones y escritura. Se puede observar que el comportamiento de ambos sistemas es muy parecido y la latencia es realmente baja por lo que la curva de rendimiento es una línea recta con un valor prácticamente igual a cero.

En la figura 10.13 se puede observar el comportamiento de lectura de los registros. Se aprecia en ambos casos que la latencia aumenta conforma se trabaja con más operaciones por segundo. HBase a pesar de tener un sistema de lectura más complejo obtiene mejores resultados que Cassandra. En todo momento se debe tener en cuenta que HBase es una base de datos optimizada para la lectura y Cassandra para la escritura.

Capítulo 10. CONCLUSIONES

Mi motivación principal para la realización de este proyecto es la adquisición de conocimiento en las tecnologías NoSQL que están revolucionando el mundo de las bases de datos. Además, el proyecto ha supuesto todo un reto debido a la temática en la que se fundamenta; un campo que me era totalmente desconocido y sobre el que he tenido que hacer una amplia investigación para conocer el funcionamiento de los sistema que deseaba analizar.

Durante el desarrollo de este proyecto, se han encontrado diversos problemas que se han tenido que resolver, investigando y profundizando sobre las distintas tecnologías que se han utilizado finalmente, obteniendo así una visión más clara sobre cada uno de los aspectos que han intervenido. La dificultad de este proyecto residía en que hay muy poca información y estudios sobre las herramientas.

HBase tiene una configuración compleja, sobre todo porque para comenzar a utilizar HBase hay que realizar la configuración de Hadoop. Esta última es la herramienta que permite a HBase comportarse como sistema distribuido. La configuración de estas dos tecnologías ha requerido mucho tiempo y paciencia. En cambio la configuración de Cassandra no me ha supuesto tanto esfuerzo, he conseguido terminarla en dos días sin ningún problema. También hay que destacar que las pruebas realizadas sobre HBase con YCSB se completaron en 3 días y, mientras se realizaban las pruebas podía utilizar los ordenadores para realizar otras tareas. En cambio para Cassandra el total de días usados para realizar las pruebas ha sido siete y mientras se ejecutaban las pruebas los ordenadores no eran capaces de realizar otras tareas con rapidez.

Observando los resultados obtenidos que se muestran en el capítulo X del proyecto, se puede concluir que ninguna de las base de datos NoSQL es perfecta. Cada una de ellas tiene sus ventajas y desventajas que se convierten en más o menos importantes en función de nuestras preferencias y el tipo de tarea para él que se quiera utilizar.

También hay que tener en cuenta que una base de datos puede demostrar un excelente rendimiento, pero una vez que la cantidad de registros supera un cierto límite, el comportamiento puede cambiar drásticamente. Esto significa que una solución particular puede ser buena para cargas de datos moderados y cálculos muy rápidos, pero no sería adecuada para los trabajos que requieren una gran cantidad de lecturas y escrituras. Además, el rendimiento de la base de datos depende también de la capacidad del hardware.

Con la realización de este proyecto se ha alcanzado el objetivo que se había propuesto y se ha conseguido esclarecer los puntos fuertes y débiles de ambos sistemas gestores de base de datos. También se ha adquirido un conocimiento profundo sobre las bases de datos NoSQL, YCSB, HBase y Cassandra. Otro de los objetivos que perseguía era automatizar la instalación de todos los componentes mediante un shell script, el cual he podido realizar satisfactoriamente. También es interesante mencionar la herramienta Sqoop de Hadoop. La he utilizado para migrar datos de una base de datos PostgreSQL a HBase y así poblar la base de datos de una manera más rápida.

Espero que esta investigación sea útil tanto para los desarrolladores que traten de elegir una base de datos como para los que necesiten realizar la instalación de alguno de los sistemas aquí estudiados.

10.1. Líneas futuras

Un trabajo muy interesante podría estar orientado a realizar las pruebas en un clúster más grande y con una base de datos más poblada. En este proyecto se ha trabajado sobre una columna tanto de lectura como escritura pero YCSB permite realizar las lecturas de la tabla sobre una o más columnas y sería interesante realizar las pruebas de lectura sobre más de una columna para observar el comportamiento de los gestores de las bases de datos.

En este proyecto por falta de tiempo y recursos sólo me he centrado en estudiar el rendimiento de las bases de datos. Pero hay muchas más características que se deben estudiar antes de decidir o saber cuál puede ser el gestor más acertado.

Características que se pueden estudiar en futuros trabajos:

- **Escalabilidad**: El mayor atractivo en la venta de los sistemas de nube es su capacidad de escalar hacia arriba añadiendo más servidores. Si el sistema ofrece baja latencia en pequeña escala, a medida que aumentamos proporcionalmente el tamaño de la carga de trabajo y el número de servidores, la latencia debe permanecer constante. Si este no es el caso, el sistema podría tener cuellos de botella que surgen a escala. La elasticidad es, básicamente, una propiedad medible de los sistemas de nube que hace referencia al manejo de los datos del servicio cuando se agregan más nodos y servidores mientras el sistema continúa en funcionamiento.
- **Disponibilidad:** una base de datos debe ser altamente disponible ante fallos. La forma más fácil de tener información sobre este aspecto es iniciar un workload sobre el

sistema y eliminar uno de los servidores mientras el workload está ejecutándose y observar cualquier comportamiento erróneo y el impacto de la acción. En un sistema real, muchos otros fallos pueden ocurrir a distintos niveles, incluidos fallos de disco duro, de red…

- **Replicación:** Los sistemas en la nube utilizan réplicas para tener mas disponibilidad y mejor rendimiento. Las réplicas se pueden utilizar en caso de fallo o para aumentar la velocidad de las solicitudes. Algunos sistemas también permite separar solicitudes de escritura.

Otro punto muy interesante que estaba mencionado en el anteproyecto pero por falta de tiempo no se ha podido realizar es incluir un estudio sobre una base de datos SQL para comparar y saber hasta qué punto los comportamientos son diferentes. Y realizar un estudio sobre hasta qué punto compensa utilizar una base de datos NoSQL.

10.2. Bibliografía

[1] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, Russell Sears: Benchmarking cloud serving systems with YCSB, Proceedings of Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC 2010, Indianapolis, Indiana, USA, June 10-11, 2010.

[2] Yahoo! Cloud Serving Benchmark.

https://github.com/brianfrankcooper/YCSB/wiki.

[3] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Michael Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber: Bigtable: A Distributed Storage System for Structured Data. ACM Trans. Comput. Syst. 26(2): (2008)

[4] HADOOP: THE DEFINITIVE GUIDE. Tom White. O'Reilly Media, Inc.,100 Gravenstein Highway North, Sebastopol, 2009.

[5] HBASE: THE DEFINITIVE GUIDE. Tom White. O'Reilly Media, Inc.,100 Gravenstein Highway North, Sebastopol, 2009.

[6] CASSANDRA: THE DEFINITIVE GUIDE. Tom White. O'Reilly Media, Inc.,100 Gravenstein Highway North, Sebastopol, 2011.

[7] http://Hadoop.apache.org/

[8] Sanjay Ghemawat , Howard Gobioff , Shun-Tak Leung, The Google file system, Proceedings of the nineteenth ACM symposium on Operating systems principles, October 19-22, 2003, Bolton Landing, NY, USA.

[9] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swami Sivasubramanian, Peter Vosshall and Werner Vogels, “Dynamo: Amazon's Highly Available Key-Value Store”, in the Proceedings of the 21st ACM Symposium on Operating Systems Principles, Stevenson, WA, October 2007.

[10] Apache Pig.

http://pig.apache.org/

[11] Apache Hive.

http://hive.apache.org/

[12] Apache avro.

http://avro.apache.org/

[13] Apache zookeeperhttp:

http://zookeeper.apache.org/

[14] Apache HBase.

http://hbase.apache.org/

[15] Apche Sqoop.

http://incubator.apache.org/projects/sqoop.html

[16] Brian F. Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein,Philip Bohannon, Hans-Arno Jacobsen, Nick Puz, Daniel Weaver, Ramana Yerneni: PNUTS: Yahoo!'s hosted data serving platform. PVLDB 1(2): 1277-1288 (2008)

[17] Guía completa de YCSB.

https://github.com/brianfrankcooper/YCSB/wiki

[18] HyperGraphDB.

http://www.hypergraphdb.org/index

[19] Arquitectura Apache HBase.

http://www.larsgeorge.com/2009/10/hbase-architecture-101-storage.html

[20] HBase – Overview of Architecture and Data Model.

http://netwovenblogs.com/2013/10/10/hbase-overview-of-architecture-and-data-model/

[21] HBase-flujo de datos.

http://www.toadworld.com/platforms/nosql/w/wiki/357.hbase-write-ahead-log.aspx#HBase_Architecture_101:_Write-Ahead_Log

[22] HBase-réplicas.

http://blog.cloudera.com/blog/2012/07/hbase-replication-overview-2/

[23] Apache Cassandra.

http://apache.org/cassandra/.

[24] PostgreSQL .

https://www.digitalocean.com/community/articles/how-to-create-data-queries-in-postgresql-by-using-the-select-command

[25] Apache Sqoop.

https://blogs.apache.org/sqoop/entry/apache_sqoop_overview

[26] Benchmarking.

http://www.networkworld.com/news/tech/2012/102212-nosql-263595.html

[27] Instalacion Cassandra.

http://www.hispabigdata.es/2013/09/como-instalar-apache-cassandra.html

[28] Apache Zookeper

http://zookeeper.apache.org/

[29] Apache UIMA

http://uima.apache.org/

[30] Apache Mahout

http://mahout.apache.org/

[31] Jaql

https://code.google.com/p/jaql/

[32] Apache Pig

http://pig.apache.org/

[33] Apache Lucene

http://lucene.apache.org/core/