

UNIVERSIDAD DE COSTA RICA

ESCUELA DE CIENCIAS DE LA COMPUTACIÓN E INFORMÁTICA

PROYECTO NOSQL

ELABORADO POR:

B52864, CARLOS GAMBOA VARGAS

B53846, JOSUÉ LEÓN SARKIS

B56219, FERNANDO ROJAS MELÉNDEZ

B57594, ANA LAURA VARGAS RAMÍREZ

PROF. LUIS GUSTAVO ESQUIVEL QUIRÓS

GRUPO 03

Ciudad Universitaria Rodrigo Facio, Costa Rica

2017

Acrónimos

A continuación se provee una lista de acrónimos que podrían ser utilizados o tienen relevancia en la comprensión de este proyecto:

1. **DB:** (Database) Base de Datos.
2. **RDBMS:** (Relational Database Management System) Sistema de gestión de bases de datos relacionales.
3. **HDFS:** (Hadoop Distributed File System) Sistema de archivos en el que se implementa HBase.
4. **SQL:** Standard Query Language.
5. **NoSQL:** No Standard Query Language.
6. **API:** (Application Programming Interface) Interfaz de programación de aplicaciones, se compone de un conjunto de rutinas y métodos para crear una capa de abstracción en el desarrollo de software.
7. **ODBC:** (Open Database Connectivity) es un estándar de acceso a las bases de datos.
8. **JDBC:** (Java Database Connectivity) API que permite la ejecución de operaciones sobre bases de datos desde Java.
9. **ACID:** Acrónimo de Atomicity, Consistency, Isolation and Durability: Atomicidad, Consistencia, Aislamiento y Durabilidad.

10. **OLAP**: (On-Line Analytical Processing) Procesamiento Analítico En Línea, es una solución utilizada en el campo de la llamada Inteligencia de negocios para agilizar la consulta de grandes cantidades de datos.
11. **OLTP** (OnLine Transaction Processing) Procesamiento de Transacciones En Línea es un tipo de procesamiento que facilita y administra aplicaciones transaccionales, para consultas, modificaciones y entradas de datos.

Índice general

Acrónimos	i
1. Introducción	1
2. HBase: un sistema de almacenamiento sin esquemas	2
2.1. Modelo de datos	3
2.2. Arquitectura	6
2.2.1. Particionamiento	13
2.2.2. Replicación	14
2.3. Acceso cliente	15
2.3.1. Acceso por medio de un API	15
2.3.2. Conectividad desde un lenguaje de programación	16
2.3.2.1. Java	16
2.3.2.2. C#	19
2.3.2.3. HBase Shell	21
2.4. Detalles de implementación	22

Índice de figuras

2.1. Una posible tabla de HBase.	4
2.2. Componentes de Apache HBase.	7
2.3. La tabla meta se encuentra en el ZooKeeper.	11
2.4. Mapeo de regiones con la tabla meta.	12
2.5. Una región dividida en dos regiones.	13

Listings

2.1.	Configuración XML	16
2.2.	Ejemplo de interacción con la DB	17
2.3.	Generación de código Thrift	19
2.4.	Conexión mediante Thrift	20
2.5.	Conexión por HBase Shell	21
2.6.	Comandos HBase Shell	22

Capítulo 1

Introducción

En la actualidad, una gran cantidad de organizaciones se enfrentan al desafío de un crecimiento masivo en el volumen de sus datos. Mientras la cantidad de datos alcanza los límites definidos de procesamiento de información, se debe a su vez analizar estas entradas y obtener un valor significativo de ellas. Dicha información es conocida como *Big Data*, y ha presentado un desafío en términos de niveles de almacenamiento y procesamiento de datos (Wijaya & Nakamura, 2013).

Para solucionar este problema se han desarrollado distintos *frameworks* especializados en el procesamiento de *Big Data*, como lo son Bigtable, Google File System y MapReduce. No obstante, dichos *frameworks* carecen de un aspecto específico: el acceso aleatorio en tiempo real al *Big Data*. HBase nace para satisfacer esta necesidad.

En el presente documento se expondrá el sistema de almacenamiento sin esquemas: HBase, desarrollado por Apache Software Foundation, y lanzado al mercado en el año 2006. Se abordarán temas como el modelo que utiliza para la organización de la información, la distribución de los procesos entre los dispositivos Maestros y Esclavos; además del acceso desde un cliente; y los requisitos de un sistema que desea implementar una DB en HBase.

Capítulo 2

HBase: un sistema de almacenamiento sin esquemas

Las bases de datos relacionales fueron por mucho tiempo la principal solución para el almacenamiento de datos, sin embargo desde la aparición de información a gran escala o *Big Data* se ha intentado buscar una solución más eficiente al tratamiento de estos (Hou et al., 2015; Point, 2017).

HBase es una base de datos distribuida de este tipo, construida encima del sistema de archivos Hadoop (HDFS). HDFS se especializa en guardar gran cantidad de información de forma eficiente, sin embargo HBase además agrega la posibilidad de acceso rápido de forma aleatoria, mientras que Hadoop solo permite acceso secuencial (Point, 2017).

Inicialmente fue solo una propuesta de Google para que en algún punto se pudiera almacenar datos eficientemente en tablas inmensas, y con el pasar de los años se convirtió, junto a HDFS, en un proyecto de alto nivel de Apache Software Foundation (Point, 2017).

2.1. Modelo de datos

Las bases de datos NoSQL suelen tener grandes rasgos que las diferencian en gran medida de las bases de datos relacionales comunes. Las formas en que modelan la estructura de sus datos varían en gran medida, y estos modelos definen cómo, para qué y por qué se usan. A pesar de que HBase es un sistema de base datos no relacional, utiliza tablas para organizar su información, pero de una forma bastante distinta a como lo hacen los RDBMS, por ejemplo es una base de datos orientada a columnas, a diferencia de la mayoría de bases de datos relacionales que son orientadas a filas (Hou et al., 2015; Karnataki, 2013).

El modelo de datos de HBase está creado con el objetivo de acomodar datos semi-estructurados, es decir, que pueden variar en tamaño, tipo y columnas. La forma en que se organizan los datos en HBase provoca que sea más fácil particionarla en distintos lugares dentro de un *cluster*, por esto mismo la mayoría de veces HBase es implementado y usado en grandes *clusters* de máquinas que guardan muchísima información (Hou et al., 2015; Karnataki, 2013; Li, 2010).

Hbase guarda muchos de sus datos como un arreglo de bytes por dos principales razones. La primera es porque hace las inserciones más fáciles y las consultas más rápidas, ya que no debe preocuparse por el tipo de dato en el momento en que se inserta o recupera, sino solo cuando se quiere interpretar, lo cual hace que sea mucho más eficiente. La segunda razón es que para cantidades de datos tan grandes, los bytes que se ahorran poniendo la información como arreglo de bytes, en vez del formato de cada tipo, al final salvan mucho espacio dentro de la base de datos (Karnataki, 2013; Team, 2017).

Entre los diferentes componentes lógicos que constituyen el modelo de datos de Hbase están los siguientes (Karnataki, 2013; Team, 2017):

1. **Tablas:** Colección lógica de filas almacenadas en diferentes regiones. Estas regiones son administradas por *Region Servers* (Hay que recordar que las tablas que maneja HBase son extremadamente grandes y pueden llegar a ser almacenadas en diferentes servidores).

2. **Filas:** Es una instancia de datos de la tabla, ya que normalmente se accede a la información por fila. Están identificadas por una llave de fila o *rowkey* que es único y es tratado como un arreglo de bytes.
3. **Familias de Columnas:** La información en las filas se agrupa en lo que se llaman familias de columnas. Cada una tiene una o más columnas asociadas. Son guardadas en un archivo juntas, donde se hace el trabajo de compresión cuando sea necesario.
4. **Columnas:** Las familias de columnas están compuestas de columnas, cada una se identifica por un calificador de columna o *Column Qualifier*. Este último está compuesto por la unión del nombre de la familia y el nombre de la columna separados por dos puntos (nombredelafamilia:nombredelacolumna). Pueden haber múltiples columnas en una familia de columnas, y una fila puede variar en la cantidad de columnas que posee.

Row Key	Customer		Sales	
Customer Id	Name	City	Product	Amount
101	John White	Los Angeles, CA	Chairs	\$400.00
102	Jane Brown	Atlanta, GA	Lamps	\$200.00
103	Bill Green	Pittsburgh, PA	Desk	\$500.00
104	Jack Black	St. Louis, MO	Bed	\$1600.00

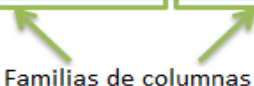

 Familias de columnas

Figura 2.1: Una posible tabla de HBase.
Fuente: (Karnataki, 2013)

5. **Celda:** Guarda la información de la combinación única de una llave de fila (*rowkey*), una familia de columnas y una columna. La información de la celda

se llama valor y se trata como un arreglo de bytes.

6. **Versión:** La información en cada celda está versionada, es decir existen diferentes versiones de su valor, cada una diferenciada por una marca de tiempo. La cantidad de versiones puede variar según la familia de columnas correspondiente pero por defecto es tres.

En la figura 2.1 se puede observar una posible organización de una tabla en HBase. En el ejemplo existen tres familias de columnas: ‘Row Key’, que contiene una sola columna con el calificador ‘Customer Id’. Luego está ‘Customer’ que incluye las columnas con los calificadores ‘Name’ y ‘City’ y finalmente ‘Sales’ con ‘Product’ y ‘Amount’. Por otro lado están las filas, que en cada una de sus celdas contiene información relacionada a las distintas columnas, por ejemplo podemos observar la llave de fila (*Row Key*) de cada una en la columna ‘Customer Id’, la cual es única (Karnataki, 2013).

El hecho de que HBase se enfoque tanto en el almacenamiento por columna no es extraño, ya que es una base de datos orientada a columnas. Los datos se tratan y almacenan a través de estas y de las familias de columnas para generar varias ventajas cuando se tratan grandes cantidades de datos que no salen a relucir en bases de datos que son orientadas a filas. Por ejemplo, es mucho más fácil recuperar datos relevantes porque HBase recupera la información por columnas. Es decir, si se quiere encontrar un valor específico dentro de una tabla el tratamiento de esta información es mucho más simple, porque solo se debe tratar con la información de la columna, mientras que se estuviera en un modelado orientado a filas habría que recuperar fila por fila para obtener los datos. Por esta misma razón es que cuando se hace procesamiento analítico en línea (OLAP), el cual analiza grupos de información muy grandes, HBase muchas veces es preferido (Karnataki, 2013).

También a la hora de hacer cambios en el conjunto de datos entero es preferible tratar la información por columnas, ya que facilita los cambios y agregados que se hagan porque solo hay que tratar con la información a manipular de cada columna, en vez de iterar fila por fila encontrando y cambiando lo que queremos. Finalmente la compresión en este tipo de bases de datos es mucho más eficiente, ya que los distintos

valores en cada columna son pocos en comparación a las distintas combinaciones de filas que puede haber, por lo que a la hora de intentar éstrujar’la información se logra una mejor solución haciéndolo por columnas(Karnataki, 2013; Sandel et al., 2015).

2.2. Arquitectura

La arquitectura de HBase sigue un modelo Maestro-Eslavo (Gómez, 2014). El *cluster* donde se implementa puede consistir en uno o más nodos con sus componentes distribuidos a través de él.

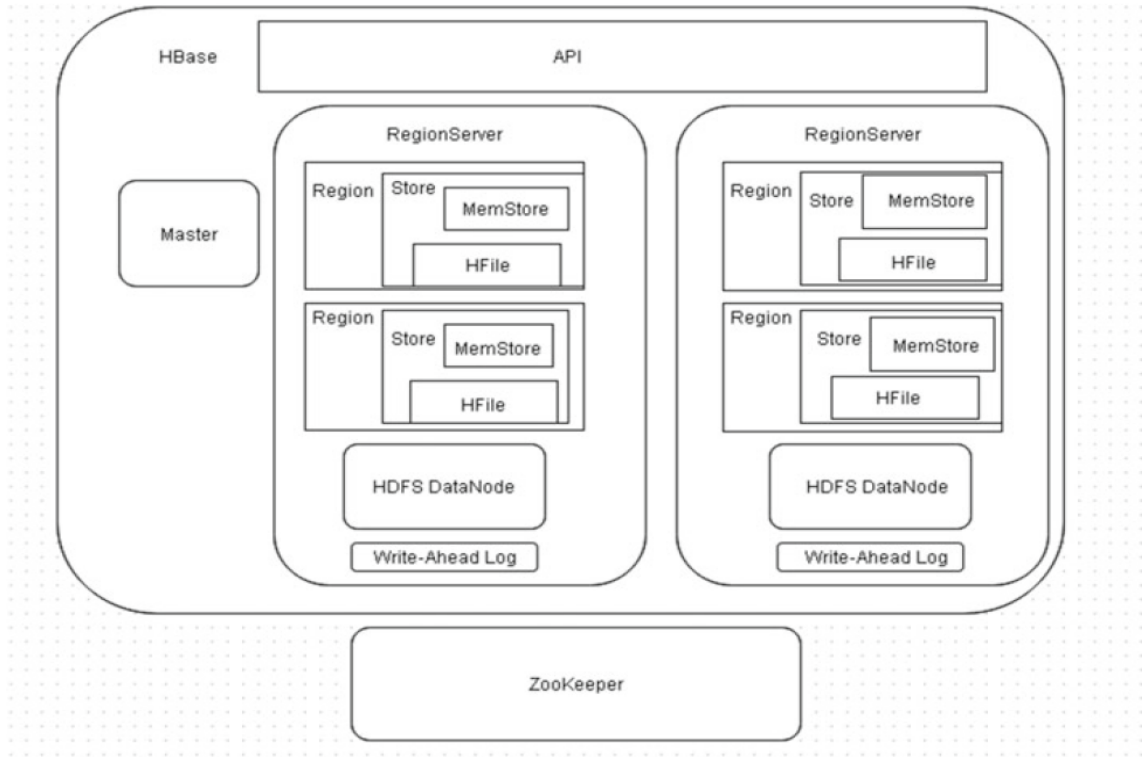


Figura 2.2: Componentes de Apache HBase.

Fuente: (Vohra, 2016)

Sus principales componentes y subcomponentes se pueden observar en la figura 2.2, a continuación se explicará cada uno de ellos.

Los datos del HDFS se organizan en directorios y archivos, estos últimos se dividen en bloques de tamaño uniforme, que además, se distribuyen entre nodos de cluster (Vora, 2011).

HDFS también adopta una arquitectura maestro-esclavo en la que el *NameNode* (maestro) mantiene el espacio de nombres de archivos (metadatos, estructura de directorios, lista de archivos, lista de bloques para cada archivo, ubicación de cada bloque, atributos de archivo, autorización y autenticación). Los *DataNodes* (esclavos) crean, eliminan o replican los bloques de datos reales basados en las instrucciones recibidas del *NameNode* y reportan periódicamente a este con la lista completa de bloques que están almacenando (Vora, 2011).

De esta manera, en un cluster distribuido, el servidor maestro normalmente se encuentra en el mismo nodo que el *NameNode* de HDFS, y los servidores de región o esclavos están en el mismo nodo que un *DataNode*. Para un *cluster* pequeño, ZooKeeper puede ser colocado junto con el *NameNode*, pero para uno grande, este debe ejecutarse en un nodo separado (Vohra, 2016).

Si bien en HBase los datos se almacenan en tablas, estas no son su unidad fundamental. Esta base de datos está diseñada para *Big Data* y en tablas individuales sería difícil de almacenar. Para esto, las tablas se componen de una o más regiones (Vohra, 2016).

Una región (*HRegion*) es un subconjunto de los datos de una tabla, o más explícitamente corresponde a la unidad mínima para agrupar las filas según su llave (Gómez, 2014), o la unidad de escalabilidad horizontal en HBase (Vohra, 2016). Cuando se crea una tabla, consta de una región por defecto. Las regiones poseen una llave de inicio (*startKey*) y de fin (*endKey*), y contienen un rango ordenado y contiguo de filas (Vohra, 2016).

Es importante destacar que las regiones no se superponen, es decir, la misma clave de fila no se almacena en varias de ellas. HBase garantiza una consistencia sólida dentro de una sola fila alojando una región en un solo servidor a la vez. Además, es importante reconocer que una tabla completa no necesariamente es almacenada en la misma región o incluso el mismo servidor, lo cual implica que puede haber varias regiones por tabla (Gómez, 2014), y cada región puede estar en un nodo diferente y puede consistir en varios archivos y bloques HDFS (Vohra, 2016).

HBase se compone de un nodo maestro (*HMaster*), el cual es el encargado de monitorear el estado de los nodos esclavos y a través de él es que se efectúan los cambios en los metadatos. Además, es el que realiza operaciones administrativas como la asignación de regiones a los servidores esclavos y el equilibrio de cargas, lo cual se logra moviendo regiones (Gómez, 2014).

Un *HRegionServer* es un servidor físico que almacena y proporciona datos. Una región es la unidad de programación lógica administrada por un servidor esclavo y cada uno tiene múltiples regiones bajo su control (Cao et al., 2017). Estos servidores

manejan las peticiones de lectura/escritura de los clientes y son responsables de las operaciones regionales, tales como la división y compactación de regiones (Vohra, 2016).

Por otro lado, el modelo de HBase posee otros componentes importantes. Cada familia de columnas corresponde a una tienda o almacén (*Store*) para una región determinada. Un almacén tiene dos partes, una parte en memoria, *MemStore*, y la otra parte en disco, *StoreFiles* (Cao et al., 2017). Los *MemStores* son los encargados de mantener en memoria los datos modificados del sistema y hay uno por cada almacén, estos se utilizan como memoria caché de escritura (Gómez, 2014). Los *StoreFiles* o *HFiles* son las estructuras donde reside la información, y puede haber varios por cada región, estos últimos a su vez se conforman por bloques, cuyo tamaño puede variar entre familias de columnas (Gómez, 2014).

HBase se basa en árboles de fusión de estructura logarítmica (árboles *Log-Structured Merge*, LSM). Este se materializa mediante el uso de *HLog*, *MemStore* y *StoreFiles*. Las modificaciones de datos se almacenan primero en memoria (*MemStore*) y se vacían en disco en intervalos regulares, o si se supera un umbral de memoria, o explícitamente con un comando shell (Vohra, 2016). Cada descarga genera un *StoreFiles* o *HFiles*. Cuando son muy pequeños se combinan en el fondo mediante un proceso llamado compactación para mantener un número de archivos mínimo, lo cual ofrece la ventaja de una búsqueda en disco más rápida, pero genera un costo adicional de E/S y puede provocar fallos en el rendimiento (Cao et al., 2017).

Asimismo, para simplificar el mantenimiento, HBase normalmente funciona en la parte superior de Apache Zookeeper, un servicio de coordinación de alto nivel. ZooKeeper arranca y se encarga de la coordinación del *cluster*, esto porque los clientes solicitan primero la dirección del servidor de región correspondiente de ZooKeeper, antes de interactuar directamente con el servidor de región respectivo para lecturas y actualizaciones (Pallas et al., 2016).

El ZooKeeper puede ser un solo nodo o un conjunto de estos, y proporciona información de estado compartida para los componentes del sistema distribuido. También, proporciona notificaciones de fallo del servidor para que un servidor maestro pueda

realizar una conmutación por error a otro servidor esclavo. Se utiliza además para almacenar metadatos para operaciones como la dirección del maestro y el estado de recuperación (Vohra, 2016).

Además, es necesario añadir que HBase utiliza *Write-Ahead Logs* (WAL) para las inserciones. Todas estas se realizan primero en este registro de escritura anticipada. Allí se registran todos los cambios de datos (*Puts* y *Deletes*) en el almacenamiento basado en archivos. El WAL es una copia de seguridad para cuando un servidor de región se cae. Bajo un funcionamiento normal, los datos almacenados en un WAL no se utilizan porque los datos se almacenan por primera vez en *MemStore* y posteriormente van a un *HFile*. Pero, si un esclavo se bloquea, los cambios de datos se reproducen desde el WAL (Vohra, 2016).

El maestro de HBase coordina el *cluster*, y es el responsable de las tareas de administración del mismo. Se encarga de asignar una o varias regiones a un esclavo durante el arranque del *cluster*, así como también puede decidir que una región va a cambiar de servidor como resultado de una operación de balance, o debido a un fallo en uno de los esclavos que obligue a reubicar todas las regiones que estaban alojadas en el mismo. Es importante recordar que el maestro queda fuera en las operaciones de lectura y escritura, permitiendo a los clientes contactar directamente con los servidores de regiones para solicitar o proporcionar los datos (Gómez, 2014).

El mapeo entre las regiones y los nodos esclavos son almacenadas en una tabla especial de HBase llamada *hbase:meta* o anteriormente conocida como *.META.*, que contiene el listado de regiones que están creadas en un momento determinado en el sistema (Gómez, 2014; Vohra, 2016).

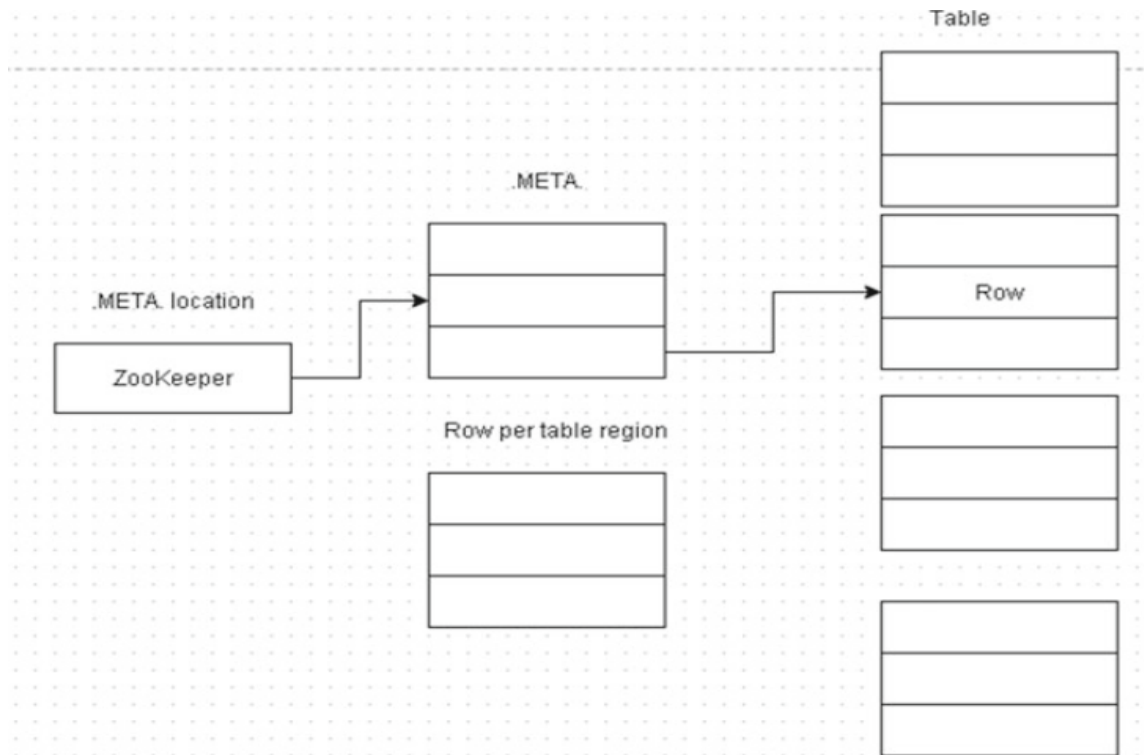


Figura 2.3: La tabla meta se encuentra en el ZooKeeper.

Fuente: (Vohra, 2016)

El hbase:meta es una tabla como las otras y el cliente tiene que buscar desde el ZooKeeper en el que *RegionServer* se encuentra la meta hbase:meta. Antes de HBase 0.96, los lugares de META se almacenaron en una tabla llamada -ROOT-, pero en esta y las versiones superiores -ROOT- se ha eliminado y las ubicaciones de la tabla se almacenan en ZooKeeper, como se muestra en la figura 2.3 (Vohra, 2016).

De este modo, el flujo normal de comunicación es el siguiente: un nuevo cliente cuando intenta acceder a una fila en particular primero contacta con el *cluster* de ZooKeeper. Con esto, se consulta .META. y se obtiene el nombre del servidor a cargo de la región que contiene la fila que busca el cliente (Gómez, 2014).

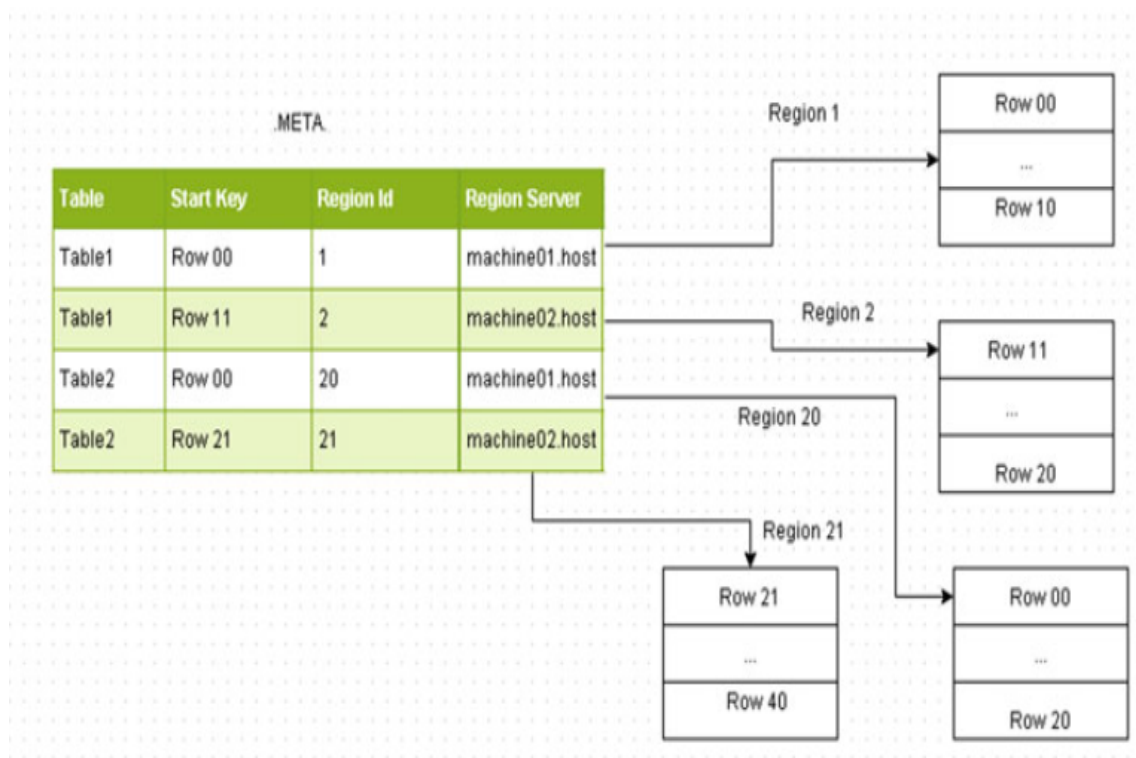


Figura 2.4: Mapeo de regiones con la tabla meta.

Fuente: (Vohra, 2016)

Por ejemplo, la tabla meta mostrada en la figura 2.4 muestra la región y las asignaciones de servidor para *Table1* y *Table2*. La tabla 1 con llave de inicio de fila Key-00 se almacena en una región con una Id de 1, que se asigna al servidor machine01host y ‘*Table1 startKey fila Key-30*’ se almacena en una región con una Id de 2, que se asigna a machine02host. De forma similar, la fila de inicio Key-00 de *Table2* se almacena en una región con una Id de 1, que se asigna a machine01host y ‘*Table2 startKey row Key-41*’ se almacena en una región con un Id de 2, que se asigna a machine02host (Vohra, 2016).

Para evitar tener que obtener la ubicación de la región una y otra vez, el cliente mantiene una memoria caché de las ubicaciones de la región. La caché se actualiza cuando se divide una región o se mueve a otro servidor esclavo debido a políticas de equilibrio o de asignación. El cliente recibe una excepción cuando la memoria caché está obsoleta y la caché se actualiza al obtener información actualizada de la tabla

meta (Vohra, 2016).

La base de datos NoSQL HBase, utiliza dos procedimientos que generan un aumento en el rendimiento del sistema, así como le permiten recuperarse de fallos. Estas técnicas se explicarán a continuación.

2.2.1. Particionamiento

En HBase, las regiones proporcionan partición de datos. Varios clientes que acceden a una tabla pueden utilizar diferentes regiones de la tabla y como resultado no sobrecargarán una partición (región) o un servidor esclavo. Tener varias regiones reduce el número de búsquedas de disco necesarias para encontrar una fila y los datos se devuelven más rápido (baja latencia) a una solicitud de cliente (Vohra, 2016).

Si el número de llaves de fila en una región es demasiado grande, la región se divide en aproximadamente dos mitades iguales, en un proceso llamado auto-sharding. La división se hace en la clave de la fila del medio.

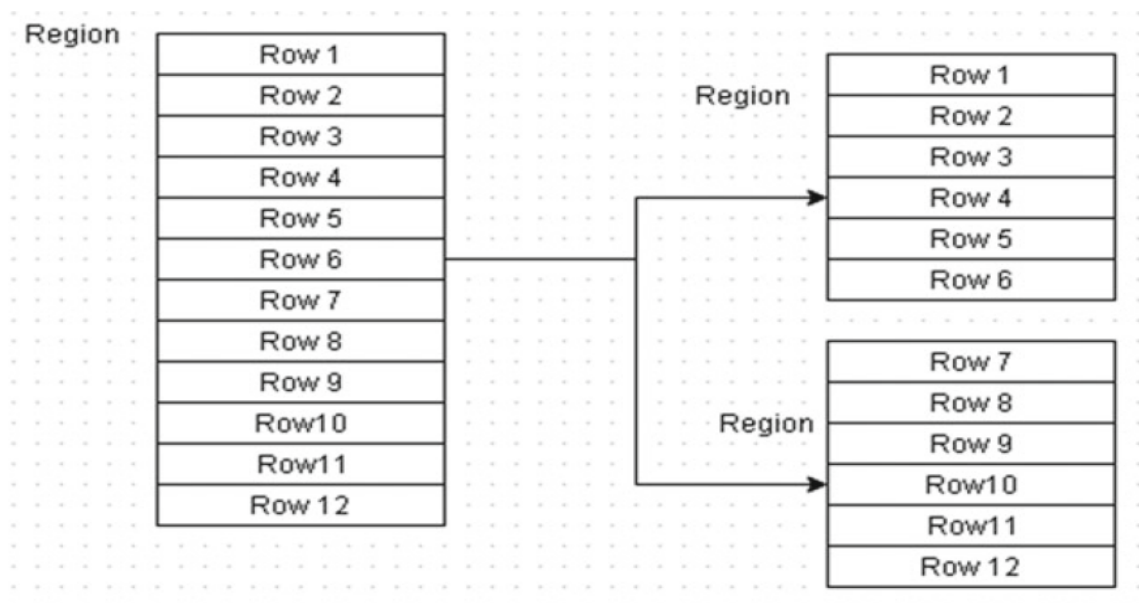


Figura 2.5: Una región dividida en dos regiones.

Fuente: (Vohra, 2016)

En la figura 2.5, la región tiene 12 filas y se divide en dos regiones con 6 filas cada una. Este proceso se aplica cuando se supera un umbral y es manejado por el servidor esclavo, que divide una región y desconecta la región dividida. Posteriormente, las dos regiones divididas se agregan a `hbase:meta`, se abren en el en el servidor y se informan al maestro (Vohra, 2016).

2.2.2. Replicación

La solución más simple para evitar la pérdida de datos es el mecanismo de replicación, es decir, copias redundantes de los datos son guardados por el sistema de modo que en caso de fallo, siempre hay otra copia disponible, con esto se mejora la confiabilidad y el sistema se vuelve tolerante a fallos. De forma predeterminada, HDFS almacena tres copias de cada bloque de datos para garantizar la fiabilidad, disponibilidad y tolerancia de fallos (Vora, 2011).

La política habitual de replicación de un centro de datos es tener dos réplicas en las máquinas del mismo servidor y una réplica en una máquina ubicada en otro. Esta política limita el tráfico de datos entre los servidores. Para minimizar la latencia del acceso a los datos, HDFS intenta acceder a los datos de la réplica más cercana (Vora, 2011).

La localidad es la proximidad de una región a un servidor de región o esclavo. Se logra con la replicación de bloques HDFS en el *cluster*. Un cliente contacta un servidor, y si una región está más cerca de este, se requiere menos transferencia de red para las operaciones del cliente (Vohra, 2016).

La directiva de ubicación de réplica es la siguiente: la primera réplica está en el nodo local, la segunda se encuentra en un nodo aleatorio en otro *rack* y la tercera está en el mismo que el segundo, pero en un nodo diferente; las réplicas subsiguientes se encuentran en nodos aleatorios en el *cluster* (Vohra, 2016).

2.3. Acceso cliente

2.3.1. Acceso por medio de un API

HBase está desarrollado en Java, por lo cual tiene una *API* nativa en este lenguaje para poder acceder programáticamente los datos y manipularlos. Mediante esta *API*, la cual se encuentra en el paquete *org.apache.hadoop.hbase.client*, se obtiene el acceso a la base de datos con mayor rapidez y facilidad (Team, 2017). Existen diferentes clases dentro de esta, las cuales ofrecen distintas funcionalidades para interactuar con los datos. El código del cliente que intenta acceder a la base de datos del modelo HBase mediante esta *API*, primero debe comunicar sus operaciones con ZooKeeper, que guarda la configuración de las consultas en un archivo (xml), ubicado en el mismo directorio del cliente (Point, 2017).

Además, HBase provee acceso desde el HBase Shell, el cual permite los distintos tipos de manipulación de datos necesarios para comunicarse con la base de datos. HBase Shell está desarrollado en el lenguaje de programación JRuby, y busca brindar otro mecanismo de acceso sencillo mediante una gran variedad de comandos (Vohra, 2016).

En los casos en que el cliente no sea Java, se pueden utilizar otros mecanismos de acceso como las librerías Thrift o Rest. Thrift, al igual que HBase desarrollado por Apache, es un *framework* que permite la comunicación entre el cliente y la base de datos de una manera más eficiente. Rest, por su parte, proporciona un servicio de interacción entre el cliente y la base de datos mediante utilidades web por medio de protocolos como HTTP. Específicamente, REST proviene de RESTful, y consiste en la comunicación entre el cliente y el servidor a través de operaciones “stateless”, en las cuales ni el emisor ni el receptor almacena información ya que el proceso de reconocimiento entre partes no es parte del protocolo (Apache.org, 2017).

2.3.2. Conectividad desde un lenguaje de programación

Los distintos API's que permiten el acceso de un cliente a una base de datos implementada en HBase permiten conectividad mediante lenguajes de programación específicos. Existen una gran cantidad de lenguajes de programación, y HBase, mediante estos API's, es compatible con muchos de ellos. No obstante, algunos de los más utilizados y soportados por los principales métodos de acceso disponibles son *Java*, *C#* y *IRuby*.

2.3.2.1. Java

Utilizando la API de Java para HBase, se puede manipular la base de datos de distintas maneras. Para crear, alterar y eliminar tablas, se debe utilizar la clase *admin*. Por otra parte, para acceder directamente las tablas ya creadas, se utiliza una instancia de *Table*. Con la instancia ya creada, para insertar información se debe tener otra instancia, la del objeto *Put()*. De manera similar, para obtener información de una fila en una tabla, se emplea el objeto *Get()*. Finalmente, para eliminar contenido se utiliza *Delete*. Todos estos principales tipos de accesos y muchos otros que ofrece la API utilizan candados dentro de la base de datos para controlar la concurrencia (Point, 2017).

Los siguientes fragmentos de código fuente en Java ejemplifican cómo interactuar con la base de datos (Paraschiv, 2017):

```
1 <configuration>
2   <property>
3     <name>hbase.zookeeper.quorum</name>
4     <value>localhost</value>
5   </property>
6   <property>
7     <name>hbase.zookeeper.property.clientPort</name>
8     <value>2181</value>
9   </property>
10 </configuration>
```

Listing 2.1: Configuración XML

En el listing 2.1 se puede observar una configuración en formato XML el cual se utiliza para conectarse con la base de datos HBase. Como se puede ver, en la primera propiedad se establece que el Zookeeper Quorum es "localhost", ya que el modo de operación con el que se utilizará la base de datos, en este ejemplo, es local. De manera similar, se asigna el puerto "2181", ya que es el determinado para que Zookeeper reciba las conexiones de clientes.

```
1 private TableName table1 = TableName.valueOf("Table1");
2 private String family1 = "Family1";
3 private String family2 = "Family2";
4
5 Configuration config = HBaseConfiguration.create();
6
7 String path = this.getClass().getClassLoader().getResource(
8 "hbase-site.xml").getPath();
9 config.addResource(new Path(path));
10
11 HBaseAdmin.checkHBaseAvailable(config);
12
13 Connection connection = ConnectionFactory.createConnection(config)
14 Admin admin = connection.getAdmin();
15
16 HTableDescriptor desc = new HTableDescriptor(table1);
17 desc.addFamily(new HColumnDescriptor(family1));
18 desc.addFamily(new HColumnDescriptor(family2));
19 admin.createTable(desc);
20
21 byte[] row1 = Bytes.toBytes("row1")
22 Put p = new Put(row1);
23 p.addImmutable(family1.getBytes(), qualifier1, Bytes.toBytes(
24 "cell_data"));
25 table1.put(p);
26
27 Get g = new Get(row1);
28 Result r = table1.get(g);
```

```
29 byte[] value = r.getValue(family1.getBytes(), qualifier1);
30
31 Bytes.toString(value)
32
33 Delete delete = new Delete(row1);
34 delete.addColumn(family1.getBytes(), qualifier1);
35 table.delete(delete);
```

Listing 2.2: Ejemplo de interacción con la DB

Por su parte, en el listing 2.2 se encuentra una serie de instrucciones para manipular la base de datos. En la línea 5 se crea una nueva configuración y luego se le asigna el archivo XML que contiene la configuración específica. La instrucción de la línea 11, llama al método `checkHBaseAvailable(config)` para chequear si la creación de la configuración fue exitosa.

Posteriormente, en la línea 13 se establece la conexión utilizando la configuración ya creada. Para poder ejecutar instrucciones de creación, consulta, modificación y borrado, se debe obtener una instancia `Admin`, como se hace en la línea 14.

Una vez obtenido la instancia `Admin` de la conexión, se pueden crear tablas las cuales requieren que se les asigne un `HTableDescriptor`. Luego, se le debe agregar familias al `HTableDescriptor`, que define la estructura de la tabla, para luego crearla como se aprecia de la línea 16 a la 19.

Después de crear la tabla, se le pueden agregar datos, como se hace a partir de la línea 21, en la cual se crea un arreglo de tipo `byte` para utilizarlo como parámetro de la instancia `Put`, la cual permitirá insertar información en una fila de la tabla. En la línea 23, se le agrega una columna a la instancia de `Put` con la familia1 creada en la línea 2 como familia, el calificador y el valor indicado. Finalmente, en la línea 25 se agrega la fila a la tabla propiamente.

La tabla ahora contiene la fila que se agregó, por lo cual podemos consultar su información, mediante una instancia de `Get`. En la línea 27 se realiza el `Get` y en la 28 se guarda el resultado de la operación `table1.get(g)`, para luego obtener los valores mediante el método `getValue()`, el cual retorna la familia indicada según el calificador

proporcionado, guardándolo en otro arreglo de bytes "value". Al ejecutar el contenido de la línea 31, se obtiene el valor "cell_data" que se insertó anteriormente.

Finalmente, en las últimas líneas (33-35), se realiza un borrado de la fila que se agregó, creando una instancia de Delete y especificando la fila a borrar mediante el rowkey.

2.3.2.2. C#

C es otro lenguaje, orientado a objetos, muy importante y utilizado actualmente, sobretodo por su gran compatibilidad con .NET, uno de los *frameworks* más relevantes para desarrollar aplicaciones del sistema operativo Windows. Establecer una conexión mediante C# con una DB HBase se logra utilizando el *framework* Thrift. Para realizarlo, se debe primero tener instalado los componentes necesarios de Thrift y la DB HBase.

Los siguientes fragmentos de código ejemplifican cómo interactuar con la base de datos (Apache.org, 2017):

```
1 thrift -r --gen csharp hbase.thrift
```

Listing 2.3: Generación de código Thrift

Debido a que Thrift soporta una serie de lenguajes de programación para conectarse con HBase, es necesario generar unas clases para establecer cuál lenguaje de programación se utilizará para comunicarse con HBase y facilitar el proceso. Por lo tanto, se debe ejecutar el comando del listing 2.3 para obtener estos archivos necesarios, que luego se deben importar en el proyecto .NET con el cual se esté trabajando.

```
1 using System;
2 using Thrift;
3 using Thrift.Protocol;
4 using Thrift.Server;
5 using Thrift.Transport;
6
7 namespace CSharpTutorial
8 {
9     public class CSharpClient
10    {
11        public static void Main()
12        {
13            try
14            {
15                TTransport transport = new TSocket("localhost",
16                9090);
17                TProtocol protocol = new TBinaryProtocol(transport);
18                Calculator.Client client = new
19                Calculator.Client(protocol);
20
21                transport.Open();
22                //Operaciones
23                transport.Close();
24            }catch (TApplicationException x)
25            {
26                Console.WriteLine(x.StackTrace);
27            }
28        }
29    }
30 }
31 }
```

Listing 2.4: Conexión mediante Thrift

Una vez generadas las clases de Thrift necesarias, se puede establecer la conexión para empezar a manipular la base de datos como un cliente. En el listing 2.4, se detalla cómo realizar la configuración para luego crear la conexión. A diferencia de Java y su API, no se necesita un archivo XML que contenga esta información (Apache.org,

2017).

De esta manera, en las líneas 15-19 se ejecutan las operaciones necesarias para iniciar la comunicación con HBase. La línea 15 crea una nueva instancia de TTransport para establecer el tipo de conexión y el puerto por el cual se comunicará Thrift con HBase. Luego se crea un protocolo utilizando el TSocket transport recién creado, para poder completar la creación de la conexión del cliente. Finalmente, se ejecuta el método Open() a la instancia de TTransport para que se puedan recibir solicitudes y efectuar operaciones en la base de datos HBase determinada (Apache.org, 2017).

Al igual que con los demás APIs y lenguajes de programación, las operaciones de interacción con la base de datos se pueden realizar en C. La sintaxis, estructura e instrucciones específicas son distintas, pero el proceso en general es similar.

2.3.2.3. HBase Shell

HBase provee la posibilidad de conectarse mediante su shell y ejecutar las operaciones necesarias como cualquier otra API. Si bien los comandos que se ejecutan desde el HBase Shell no corresponden a un lenguaje de programación específico, de igual manera son instrucciones programadas para realizar procedimientos y obtener resultados.

Los siguientes fragmentos de comandos ejemplifican cómo conectarse e interactuar con la base de datos desde el HBase Shell ([Point, 2017](#); [Yadav, 2017](#)):

```
1 cd /usr/localhost/
2 cd Hbase
3
4 ./bin/hbase shell
5
6 HBase Shell; enter 'help<RETURN>' for list of supported commands.
7 Type "exit<RETURN>" to leave the HBase Shell
8 Version 0.94.23, rf42302b28aceaab773b15f234aa8718fff7eea3c, Wed Aug 27
9 00:54:09 UTC 2014
10
11 hbase(main):001:0>
```

Listing 2.5: Conexión por HBase Shell

Para abrir el HBase Shell y empezar a utilizarlo se deben ejecutar los comandos de las líneas 1 y 2 del listing 2.5, ya que para abrir el shell interactivo se debe correr el comando de la línea 4 desde la carpeta donde HBase se encuentra instalado. Una vez ejecutado, si todo se hizo correctamente, debe desplegarse un mensaje de inicio, el cual se puede observar a partir de la línea 6. Una verificación que se puede realizar para comprobar el funcionamiento correcto del shell, es ejecutar el comando "list", el cual debe desplegar todas las tablas existentes en la base de datos.

```
1 hbase> create 'tabla', 'Customer', 'Sales'
2 hbase> put 'tabla','1','Customer:Name','John_White'
3 hbase> put 'tabla','1','Customer:City','Los_Angeles,_CA'
4 hbase> put 'tabla','1','Sales:Product','Chairs'
5 hbase> put 'tabla','1','Sales:Amount','$400.000'
```

Listing 2.6: Comandos HBase Shell

Una vez adentro del HBase Shell, se pueden ejecutar una gran cantidad de comandos para realizar transacciones. En el listing 2.6 se pueden visualizar los comandos necesarios para crear la tabla de la figura 2.1 explicada en el inicio e insertar la primera fila de datos. El primer comando crea la tabla, con sus familias de columnas asociadas y el resto insertan valores a las celdas de la fila. Las columnas se crean cuando se crean nuevas celdas, por ejemplo, no debemos crear la columna Customer:Name antes de insertar información, solo usarla y esta se crea sola (Yadav, 2017).

2.4. Detalles de implementación

Lars (Lars, 2018) indica los siguientes requerimientos que se deben tomar en cuenta a la hora de implementar HBase. En HBase existen dos tipos de máquinas diferentes: los maestros y los esclavos, y para estos existen recomendaciones específicas en lo referente a *Hardware*.

- **Núcleos:** Como los maestros realizan menos tareas que los esclavos, se recomienda que las máquinas de este tipo posean alrededor de 4 a 8 núcleos, mientras que las máquinas de tipo esclavo realizan una mayor cantidad de trabajo, por lo que es necesario entre 4 a 10 núcleos.
- **Memoria:** Las máquinas maestro deben correr el *NameNode*, *Zookeeper* y *HMaster*; por lo que se recomienda una memoria de 24GB. Por otro lado los esclavos deben correr los *DataNodes* y los *HRegionServer*, los cuales requieren más memoria, por consiguiente se deberían poseer una memoria superior a los 24GB.
- **Discos:** Los datos se almacenan en los esclavos, por lo que ellos son los que requieren una capacidad mayor, sin embargo hay que tomar en cuenta que si la base de datos se orienta más a procesos o lectura/escritura, se debe balancear el número de discos con el número de núcleos disponibles. En resumen se recomienda que las máquinas maestro posean 4 discos de 1 TB SATA, RAID 1+0, mientras que los esclavos 6 discos de 1 TB SATA, JBOD.
- **Chassis:** El *chassis* del servidor no es de gran relevancia en HBase, no obstante es recomendable poseer una tarjeta de 2 o 4 puertos Gigabit Ethernet tanto en maestros como esclavos, o si se posee una InfiniBand también es recomendable.

Como se mencionó anteriormente, HBase ocupa de Java para funcionar. Se debe poseer la versión 6 del lenguaje (1.6) proveída por Oracle. Además, para su funcionamiento se debe verificar que el archivo binario de Java sea ejecutable, y a su vez, el camino a dicho archivo se debe encontrar en las variables de ambiente del sistema (Lars, 2018).

En lo referente a *FileStystem* hay varias recomendaciones. En primer lugar se encuentran ext3 y ext4, los cuales funcionan en Linux únicamente y se han probado como unos FileSystem estables y confiables. En segundo lugar se encuentra XFS, el cual también funciona únicamente en Linux y posee características similares a ext4, además que puede formatear un disco entero virtualmente de manera inmediata, sin

embargo afecta operaciones de metadata como lo es borrar gran cantidad de archivos. Por último se encuentra ZFS que es soportado principalmente por Solaris que posee características avanzadas las cuales son de gran ayuda al combinarlas con HBase (Lars, 2018).

Bibliografía

Apache.org (2017). Apache thrift. <https://thrift.apache.org/>.

Cao, C., Wang, W., Zhang, Y., & Ma, X. (2017). Leveraging column family to improve multidimensional query performance in hbase. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, (pp. 106–113).

Gómez, R. H. (2014). Bases de datos nosql: Arquitectura y ejemplos de aplicación. <https://core.ac.uk/download/pdf/44310803.pdf>.

Hou, Y., Yuan, S., Xu, W., & Wei, D. (2015). Transformation of an e-r model into hbase tables: A data store design for ihe-xds document registry. In *Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom), 2015 IEEE 12th Intl Conf*.

Karnataki, V. (2013). Hbase – overview of architecture and data model. <https://www.netwoven.com/2013/10/10/hbase-overview-of-architecture-and-data-model/>.

Lars, G. (2018). *HBase: The Definitive Guide*. O'Reilly Media, Inc.

Li, C. (2010). Transforming relational database into hbase: A case study. In *2010 IEEE International Conference on Software Engineering and Service Sciences*.

Pallas, F., Günther, J., & Bermbach, D. (2016). Pick your choice in hbase: Security or performance. In *2016 IEEE International Conference on Big Data (Big Data)*, (pp. 548–554).

- Paraschiv, E. (2017). *HBase with Java*. Str. Drumul Gazarului, nr. 40, bl. A2, sc. 1, ap. 5: Baeldung SRL.
- Point, T. (2017). Hbase overview. https://www.tutorialspoint.com/hbase/hbase_overview.htm.
- Sandel, R., Shtern, M., & Fokaefs, M. (2015). Evaluating cluster configurations for big data processing: an exploratory study. In *2015 IEEE 9th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Environments (MESOCA)*.
- Team, A. H. (2017). Apache hbase reference guide. http://hbase.apache.org/book.html#_rest.
- Vohra, D. (2016). *Apache HBase Primer*. White Rock, British Columbia, Canada: Apress.
- Vora, M. N. (2011). Hadoop-hbase for large-scale data. In *Proceedings of 2011 International Conference on Computer Science and Network Technology*, volume 1, (pp. 601–605).
- Wijaya, W. M. & Nakamura, Y. (2013). Predicting ship behavior navigating through heavily trafficked fairways by analyzing ais data on apache hbase. In *2013 First International Symposium on Computing and Networking*, (pp. 220–226).
- Yadav, V. (2017). *Working with HBase*, (pp. 123–142). Berkeley, CA: Apress.