ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

"EVALUACIÓN DE MAPREDUCE, PIG Y HIVE, SOBRE LA PLATAFORMA HADOOP"

INFORME DE MATERIA DE GRADUACIÓN

Previa a la obtención del Título de:

INGENIERO EN CIENCIAS COMPUTACIONALES
ESPECIALIZACIÓN SISTEMAS MULTIMEDIA

Presentado por:

Franklin Ricardo Parrales Bravo

Marco Genaro Calle Jaramillo

Guayaquil, Ecuador

2010

AGRADECIMIENTO

A Dios primeramente por habernos brindado de salud y recursos para salir adelante en nuestra vida y en la elaboración del presente trabajo investigativo. A Msc. Cristina Abad Robalino, Directora del Proyecto, por su ayuda y colaboración para la realización de este trabajo.
Al Ing. Marcelo Loor, por su constante apoyo y dirección.
A nuestros amigos y familiares.

DEDICATORIA

A Dios por darnos la salud y sabiduría necesaria para terminar nuestra carrera universitaria. A nuestros padres, por habernos dado sustento y cariño para salir adelante. A nuestros hermanos y familiares.

TRIBUNAL DE SUSTENTACIÓN

Msc. Cristina Abad R.

PROFESORA DE LA MATERIA DE GRADUACIÓN

Ing. Marcelo Loor

PROFESOR DELEGADO POR EL DECANO

DECLARACIÓN EXPRESA

"La responsabilidad del contenido de este Proyecto de Graduación, nos corresponde exclusivamente; y el patrimonio intelectual de la misma, a la Escuela Superior Politécnica del Litoral"

(Reglamento de exámenes y títulos profesionales de la ESPOL)

Franklin Ricardo Parrales Bravo

Marco Genaro Calle Jaramillo

RESUMEN

A nivel de programación sobre una determinada plataforma, siempre hay dos aspectos importantes a evaluar, la eficiencia del código generado y la facilidad de lectura y escritura sobre un determinado lenguaje. Pig y Hive son dos lenguajes que nos ayudan a ver desde una perspectiva más fácil la programación sobre Hadoop, plataforma para procesamiento masivo de datos. Por el contrario, el lenguaje Java, nativo en Hadoop, permite que la ejecución de los programas sea más optimo pero a costa de su dificultad de programación.

Nuestra investigación demuestra mediante un análisis experimental comparativo, la eficiencia (en términos de tiempo) de tres programas escritos sobre estas plataformas para dar a conocer cuál de ellos es el más óptimo para diferentes tareas.

El objetivo de nuestro proyecto es el de evaluar el rendimiento del framework Hadoop y de herramientas implementadas sobre el mismo como lo son Hive y Pig, todo esto con el propósito de determinar cuál de las herramientas anteriormente mencionadas es más adecuada para el procesamiento masivo de datos.

Para lograr dicho objetivo, se han elaborado tres consultas sobre logs obtenidos de Web servers (Apache), con el propósito de trabajar con información real: 1) conteo de direcciones IP, 2) identificar fechas de eventos de advertencia o error, y 3) identificación de la pagina que más veces no ha sido hallada, y la hora a la que más veces no se la halló. Dichas consultas fueron elaboradas para cada una de las herramientas a evaluar.

Con el fin de que el proceso de consultas sea eficiente y no consuma recursos propios, se usó los servicios computacionales que provee Amazon Web Services, para el desarrollo de este caso de estudio.

Cada una de las consultas anteriormente descrita, ha sido analizada y comparada por rendimiento en las tres herramientas, para clústeres de varios tamaños (2, 4, 6, 10, 15 y 20 nodos). Además, fueron comparadas sobre como respondían en términos de tiempo al variar la cantidad de MB de los Apache Logs, todo esto para alcanzar nuestro objetivo planteado.

Finalmente se hace una conclusión sobre qué aspectos considerar a la hora de elegir una herramienta para el procesamiento masivo de los datos.

ÍNDICE GENERAL

INTRODUCCIÓN

El presente proyecto realiza procesamiento de log (bitácoras) de gran tamaño, producto de las actividades generadas por usuarios que consultan datos en las páginas alojadas en un servidor de Web (Apache). Dicho procesamiento es útil cuando se desea extraer el comportamiento de los usuarios sobre un determinado servidor, también nos permite conocer que errores se han producido en el servidor y otro tipo de información que nos sirve para hacer un posterior análisis forense de actividades realizadas (1).

Cuando se desea explorar actividades en servidores Web como Apache, se presenta el problema de que muchos de sus logs ocupan una cantidad inmensa de datos que van desde los Megabytes hasta los Terabytes. Es por ello que se hace indispensable el uso plataformas para el procesamiento masivo de datos distribuido en varios computadores para lograr un mayor aprovechamiento de recursos y generar reportes en un menor tiempo.

Hadoop es una implementación en Java de MapReduce (2), el cual es un paradigma para el procesamiento masivo de datos propuesto por Google. Su forma de programar llega, dependiendo del caso, a ser un poco dificultoso, sobre todo por la necesidad de saber cómo paralelizar adecuadamente las tareas, y en algunos casos, definir formatos de entrada propios (InputFormat) para cargar los datos y ser procesados.

Por otra parte, existen herramientas implementadas sobre Hadoop que facilitan su uso. Entre las más conocidas están Hive y Pig.

Hive (3) es una herramienta que me permite consultar, almacenar y procesar datos mediante un lenguaje de consultas parecido al SQL denominado Hive QL (con sus siglas HQL). Su uso está orientado a realizar Data WareHousing para información de empresas.

Pig (4) es otra herramienta implementada sobre Hadoop que proporciona un lenguaje de alto nivel (Pig latin) de flujo de datos, para fácilmente procesar grandes fuentes de información. Esta herramienta administra los procesos MapReduce necesarios para completar la tarea de consulta. El lenguaje Pig latin (5) con el que se escriben las sentencias sobre pig, se caracteriza por poseer: fácil programación, oportunidades de optimización y extensibilidad pues el usuario puede añadir funcionalidad extra a las que ya vienen con el API de Pig.

El objetivo de nuestro proyecto es el de evaluar el rendimiento del framework Hadoop y de herramientas implementadas sobre el mismo como lo son Hive y Pig, todo esto con el propósito de determinar cuál de las herramientas anteriormente mencionadas es más adecuada para el procesamiento masivo de datos.

El resto de este documento está organizado de la siguiente manera. En el Capítulo 1 se describen las generalidades del proyecto. El Capitulo 2 abarca el diseño de las tareas a ejecutar sobre Hadoop, Pig y Hive. Por otra parte, el Capítulo 3 trata sobre detalles en la implementación de las consultas sobre Apache logs. Finalmente, en el Capítulo 4 se presenta los resultados en términos de tiempo de ejecución, al evaluar las tareas de consultas sobre apache logs en clúster de computadoras. También se añade su respectivo análisis con sus respectivos gráficos para dar las conclusiones y recomendaciones sobre cual herramienta se debe elegir para una determinada tarea.

1. GENERALIDADES

1.1 Análisis del problema

Cuando se desea implementar una solución de programación siempre hay dos aspectos importantes a considerar al elegir entre una u otra plataforma. El primer aspecto es la optimización del código, pues en muchos de los casos, lo que se desea es que nuestras soluciones sean eficientes (es decir, que cumplan los objetivos que se trazaron y que lo realicen en un tiempo adecuado); todo esto con el propósito de evitar los costos que implica una demora en generar resultados. Por otra parte, el segundo punto a evaluar al escoger una herramienta es la facilidad que nos proporciona esta para implementar la solución planteada. Esto es vital cuando se quiere ahorrar el tiempo que involucra el aprender a dominar un determinado lenguaje. Muchas veces estos dos puntos expuestos contrastan entre sí, de tal forma que cuando se desea facilitar la escritura de las sentencias de una plataforma, se tiene que sacrificar el costo de optimización.

Estos dos aspectos anteriormente descritos son los que se han evaluado en el presente trabajo, en cuanto al procesamiento masivo de datos con Hadoop. En nuestro caso, se ha seleccionado un problema real: la tarea de procesar masivamente logs generados por un servidor Apache. Se lo seleccionó, puesto que se presta para procesarla en forma distribuida, y porque en muchos de los casos el tamaño de los archivos puede estar en el orden de los Gigabytes o más. Para ello, existe un framework de procesamiento masivo y escalable de datos llamado Hadoop (6), el cual proporciona una estructura escalable; esto nos permite gozar del beneficio de procesar grandes cantidades de datos de una manera eficiente, como los datos que existen en un log de actividades. A su vez, existen otras herramientas implementadas sobre Hadoop como Pig y Hive, que nos ayudan a ver desde una perspectiva más fácil, la programación para ejecutar tareas distribuidas que se ejecuten en un clúster de computadores.

Surge la necesidad de evaluar las tres herramientas para demostrar mediante un análisis comparativo de tiempos de ejecución, cuál de ellos es el más óptimo para una determinada tarea.

1.2 Alcance

El objetivo del proyecto es comparar tres programas de consulta sobre logs de Apache implementadas en Pig, Hive y Hadoop. Las consultas están definidas de la siguiente forma:

Primera consulta: Determinar la cantidad de veces que aparece cada dirección IP en el log.

Segunda consulta: Conocer la hora a la que se ha generado la mayor cantidad de errores en el servidor.

Tercera consulta: Obtener la página o recurso que más veces ha generado errores en el servidor y saber a qué hora este ha producido la mayor cantidad de errores.

Cada una de las consultas, fue implementada en las tres herramientas y evaluada en clústeres levantados bajo demanda con el servicio EC2 (7) (Elastic Computing Cloud) de Amazon Web Services de 2, 4, 6, 10, 15 y 20 nodos. Además, fueron comparadas sobre como respondían en términos de tiempo al variar la cantidad de MB de los Apache Logs, se tomaron en cuenta los siguientes tamaños: 128 MB, 256 MB, 512 MB y 1GB. Todo esto para alcanzar nuestro objetivo planteado. Para cada clúster se hicieron como mínimo 30 observaciones para cerciorarnos que los resultados representen al promedio de tiempo. Se lo realizó así, para concluir con alta confiabilidad estadística sobre cuál herramienta es la más eficiente, para ello se ha escogido usar la distribución T de student, pues nos permite obtener la información planteada poseyendo un mínimo de 30 datos.

Finalmente, el objetivo más ambicioso del proyecto, luego del análisis exhaustivo de resultados obtenidos del proceso anterior; es determinar cuál de las plataformas es la más adecuada para realizar tareas de procesamiento masivo de datos. También se hace las respectivas recomendaciones sobre cuál de estas herramientas se debe elegir para una determinada tarea.

2. DISEÑO DE LA SOLUCIÓN

2.1 Datos de entrada

Para nuestro análisis, se ha considerado el uso de logs de actividades hechas en un servidor Web con Apache versión 2.2.3 instalado. Se ha elegido dicho tipo de información, puesto que es de vital importancia: 1) Conocer el estado de nuestro servidor, 2) Evaluar cómo se comportan las aplicaciones y servicios alojados en el mismo. Entre el conocimiento que se obtiene de un log, se hallan: la capacidad de depuración, estadísticas y el monitoreo del servidor (1).

Además, los archivos de log nos ayudan a tener un punto de partida a la hora de solucionar problemas, relacionados a algún demonio o servicio de alta o baja disponibilidad. En sitios de gran demanda de consulta, estos archivos suelen llegar desde el orden de los gigabytes hasta terabytes; creando la necesidad de procesarlos masivamente en clúster de computadoras, para lograr adquirir una mayor eficiencia al obtener información (8).

Para más detalles sobre el formato de los datos en los logs de servidores Web Apache, lea la Sección “3.1 Análisis de datos de entrada.”

2.2 Paradigma MapReduce

MapReduce es un modelo de programación distribuida que permite el procesamiento masivo de datos a gran escala de manera paralela. Fue planteado originalmente por Google (2) como alternativa escalable y tolerante a fallos para el procesamiento masivo de datos. En este modelo, el usuario debe plantear el diseño de su solución como dos funciones: Map y Reduce.

La operación Map se aplica a los datos de entrada y los agrupa en una lista ordenada tipo clave/valor, luego procesa este resultado dentro de la función Reduce que agrupa los valores por medio de las claves (2).

A continuación, se detalla mediante figuras 1 y 2 el flujo lógico y físico del proceso MapReduce; el ejemplo que utilizan dichas figuras es el de contabilizar cuantas veces aparece un determinado animal en un listado dado.

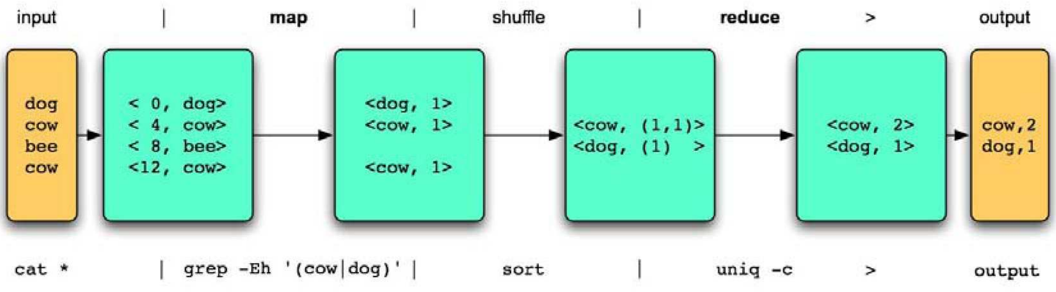

Figura 1. Flujo Lógico de procesos MapReduce (9)

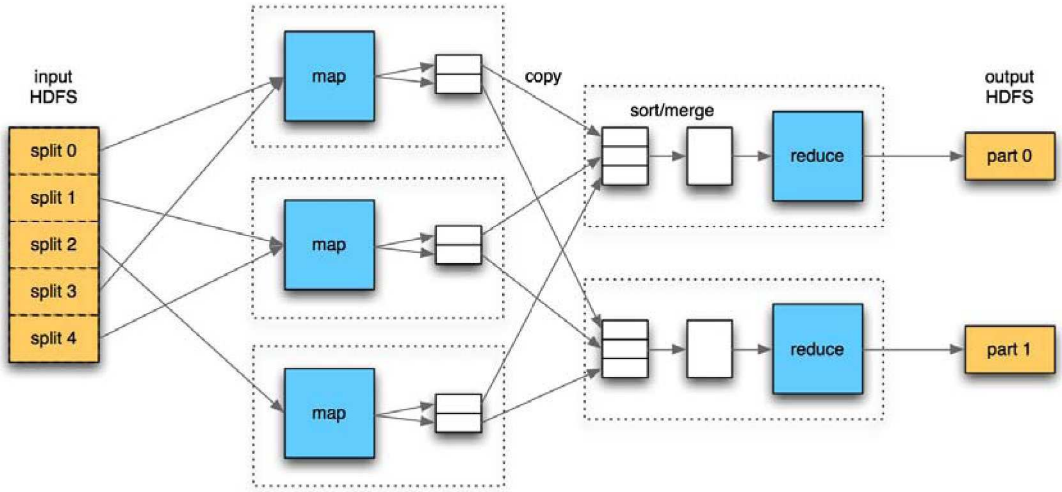

Figura 2. Flujo físico de procesos MapReduce (9)

2.3 Framework para procesamiento distribuido

Para realizar el procesamiento de datos generados por la actividad realizada en un servidor de apache, elegimos a Apache Hadoop (6), pues, al ser una implementación libre de MapReduce, nos provee las facilidades necesarias para ejecutar nuestros procesos de una manera eficiente, escalable y distribuida.

Hadoop es un framework que permite ejecutar aplicaciones sobe una gran cantidad de computadores. Para su implementación utiliza el paradigma MapReduce y un sistema de archivos distribuido propio denominado HDFS (Hadoop Distributed File System).

Además del mencionado framework, se han usado dos herramientas implementadas sobre el mismo, las cuales nos ayudan a facilitar la escritura de sentencias; estos son: Hive y Pig, cuyas ventajas ya han sido explicadas en la sección de introducción del presente trabajo de investigación.

2.4 HDFS (Hadoop Distributed File System)

HDFS es un sistema de archivos distribuido basado en la idea planteada por Google en su paper sobre Google File System (10); la misma que sigue una arquitectura cliente/servidor, basada en un único nodo servidor denominado Nameserver. Dicho nodo maneja el Namespace del sistema de archivos y regula el acceso a los mismos. Además posee varios clientes llamados Datanodes o simplemente nodos, que administran el almacenamiento de los nodos que contienen.

Estructuralmente, un archivo es dividido en bloques, y estos son almacenados en un conjunto de Datanodes. Todos los bloques excepto el último son del mismo tamaño (64MBs por defecto).

Cada bloque de archivos explicado en el párrafo anterior es replicado para proveer al sistema la tolerancia a fallos, la cual es una característica importante de un sistema distribuido (11). Sin embargo, el tamaño de bloque de archivos y el número de réplicas puede ser configurado para cada archivo.

Uno de los factores que distingue a este sistema de archivo con respecto a los tradicionales sistemas de archivos distribuidos, es la optimización de la colocación de las réplicas en los diferentes Datanodes; justamente lo hace para lograr la tolerancia a fallos del sistema.

Por otra parte, también posee soporte a la estructura jerárquica de directorios, lo cual nos permite crear, borrar, mover y renombrar tanto directorios como archivos. Sin embargo, cabe recalcar que los archivos que se eliminan, serán en realidad ubicados temporalmente a una 'papelera' localizada en /trash.

HDFS es accesible mediante un API para java pero también se puede explorar su contenido mediante el uso de un navegador Web (9).

2.4 Plataforma de Computación en las Nubes

Para evaluar el rendimiento de las tres herramientas (Hadoop, Hive y Pig), se utiliza la plataforma de Computación en las Nubes (Cloud Computing) que nos ofrece los servicios de Amazon Web Services (12); específicamente el Elastic Computing Cloud (EC2) (7) para levantar un clúster Hadoop, y el Simple Storage Service (S3) (13) para el almacenamiento de los datos de entrada y de salida. Se han usado dichos servicios, pues nos facilita el uso de todo el hardware requerido para realizar nuestra tarea de evaluación, con un costo relativamente bajo. Además nos proporciona un conjunto de herramientas para revisar el estado de cada nodo del clúster.

2.5 Flujo de procesos

La Fig. 3 muestra el diseño de flujo de procesos. Se obtienen primero los datasets de entrada, en nuestro caso son los logs de actividades de los Web servers Apache de la Facultad de Ingeniería en Electricidad y Computación de la ESPOL. Luego, se lo cargó a las tres herramientas para procesamiento masivo de datos. A continuación, se ejecutan las consultas propuestas en el presente caso de estudio en su sección de “1.2 Alcance”; dichas consultas se las ejecuta en cada una de las tres herramientas propuestas. Finalmente, se obtienen los reportes y se hace medición de tiempo de respuesta para su posterior comparación y análisis.

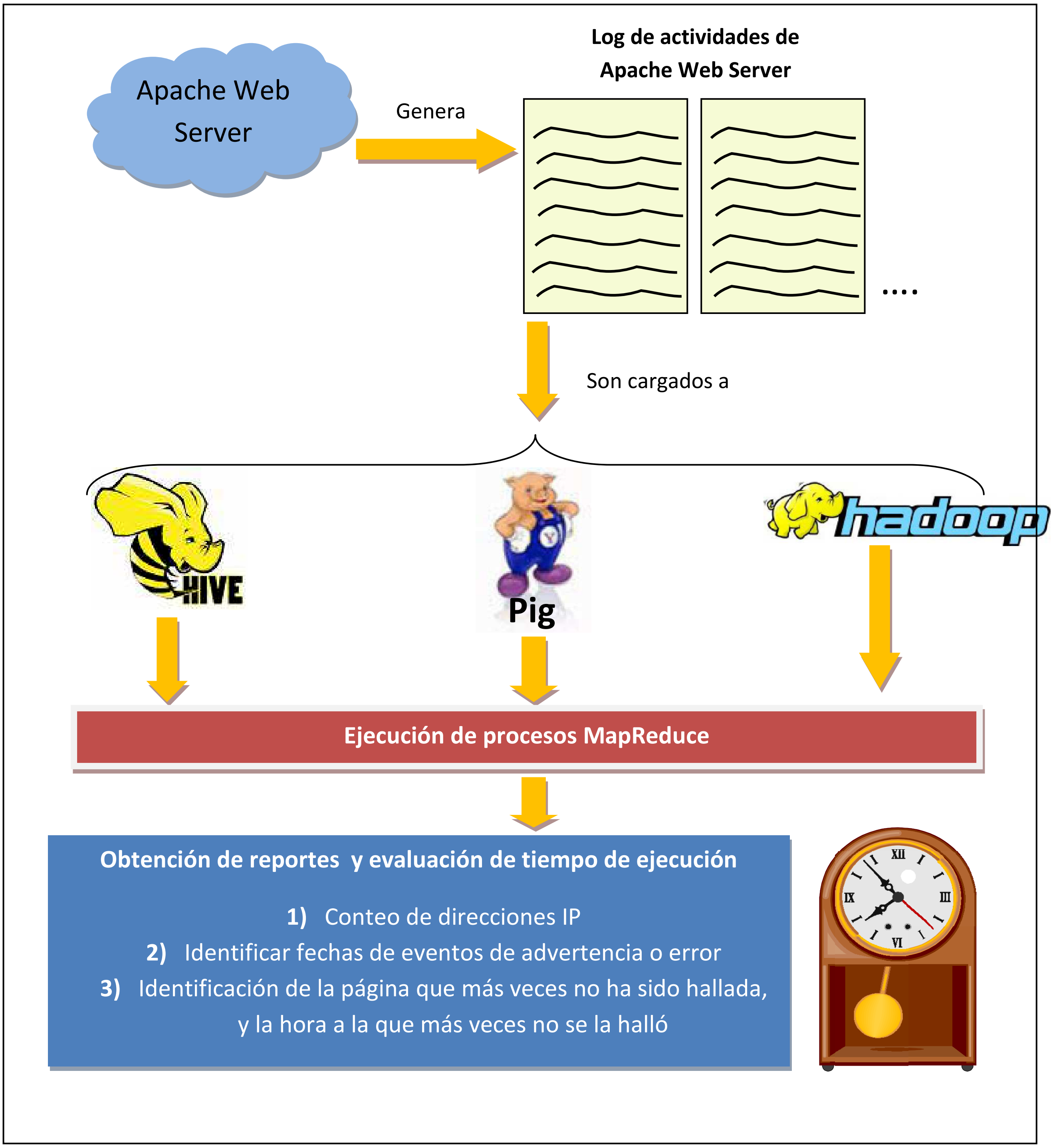

Figura 3. Flujo de procesos de evaluación de herramientas Hadoop, Pig y Hive sobre Apache Logs

3. IMPLEMENTACIÓN

En este capítulo, se describe como están estructurados los datos de entrada, con los que se desea evaluar las tres herramientas de procesamiento masivo de datos propuestas.

Los formatos de tabla de resultados, fueron establecidos de acuerdo a la necesidad de cada consulta a evaluar sobre las tres herramientas propuestas. En la segunda y la tercera consulta, se ha planteado el uso de dos fases de proceso MapReduce; contrastando con la primera, en la que solo se lleva a cabo un solo procesamiento.

3.1 Análisis de datos de entrada.

Los log de apache con los que se trabajo, poseen el siguiente formato (14):

```
127.0.0.1 - frank [10/Oct/2000: 13:55:36 -0700] "GET / HTTP/1.0
apache_pb.gif" 200 2326
```

Cada parte de esta entrada de registro se describe a continuación:

Dirección IP (`127.0.0.1`**):** Esta es la dirección IP del cliente (host remoto) que hizo la petición al servidor. La dirección IP no es necesariamente la dirección de la máquina en la que el usuario está sentado. Si existe un servidor proxy entre el usuario y el servidor, esta dirección será la dirección del proxy, en lugar de la máquina de origen.

Pieza solicitada (`-`**):** El "guión" en el resultado indica que la pieza de la información solicitada no está disponible. En este caso, la información que no está disponible es la identidad RFC 1413 del cliente determinado por identd en la máquina del cliente. Esta información es muy poco fiable y nunca debería ser usada, excepto en las redes internas muy controladas.

Usuario (`frank`**):** Este es el identificador de usuario, de la persona que solicita el documento determinado por la autenticación HTTP (el mismo valor que se proporciona a los scripts CGI en la variable de entorno REMOTE_USER). Si el código de estado de la solicitud es 401, entonces este valor no se debe confiar, porque el usuario no está autenticado. Si el documento no está protegido con contraseña, esta entrada será "-" al igual que la pieza solicitada.

Estampa de tiempo (`[10/Oct/2000: 13:55:36 -0700]`**):** El tiempo que el servidor terminado de procesar la petición. El formato es: [día / mes / año: hora: minuto: segundos zona], además, se tiene la siguiente información: día = posee 2 dígitos, mes = posee 3 letras, año = posee 4 dígitos, hora = posee

2 dígitos, minuto = posee 2 dígitos, segundo = 2 posee dígitos, zona = (`+ '|` - ') posee 4 dígitos. Es posible mostrar la hora en otro formato, sin embargo el formato que describimos aquí es el que se usó en nuestro caso de estudio.

Método y recurso solicitado (`"GET / apache_pb.gif HTTP/1.0"`**):** La línea de la solicitud del cliente se da entre comillas dobles. La línea de petición contiene una gran cantidad de información útil. En primer lugar, el método utilizado por el cliente es GET. En segundo lugar, el cliente solicita el recurso / apache_pb.gif, y tercero, el cliente utiliza el protocolo HTTP/1.0. También es posible registrar una o más partes de la línea de solicitud de forma independiente.

Estado (`200`**):** Este es el código de estado que el servidor envía de vuelta al cliente. Esta información es muy valiosa, porque revela si la petición de una respuesta satisfactoria (códigos a partir de 2), una redirección (los códigos a partir de 3), un error causado por el cliente (códigos a partir de 4), o un error en el servidor (los códigos a partir de 5). La lista completa de códigos de estado posibles se pueden encontrar en la especificación de HTTP (RFC2616 sección 10).

Tamaño (`2326`**):** La última entrada del registro indica el tamaño del objeto devuelto al cliente, sin incluir las cabeceras de respuesta. Si el contenido no fue devuelto al cliente, este valor será "-".

3.2 Cargar datos de logs de Apache

El log de apache especificado aquí, es alimentado al sistema distribuido usando desde el servicio S3 de los Amazon Web Services (descrito en la Sección 2.4). Se transfieren estos archivos hacia el Hadoop Distributed File System (HDFS), para luego ser procesados por un programa en Java (Hadoop) y comandos elaborados en Pig y en Hive que también se transfiere al nodo maestro del servicio.

Todo el proceso se lo ha ejecutado creando clúster varios nodos con los servicios Amazon EC2, con una imagen Amazon Machine Image (AMI) Fedora de Cloudera que soporta la versión de PIG 0.5 y JAVA 1.6.0.10.

A continuación se presenta la porción más importante del código en lenguaje PigLatin, Hive QL y Java nativo con Hadoop de la aplicación. Este segmenta los datos de los log de apache y los carga en el sistema para su posterior procesamiento.

En Hive Query Language:

```
CREATE TABLE apachelog (
ipaddress STRING, identd STRING, user STRING,finishtime STRING,
requestline string, returncode INT, size INT)
ROW FORMAT SERDE
'org.apache.hadoop.hive.serde2.dynamic_type.DynamicSerDe'
WITH SERDEPROPERTIES (
'serialization.format'='org.apache.hadoop.hive.serde2.thrift.TCTLSep
aratedProtocol',
'quote.delim'='("|\\[|\\])',
'field.delim'=' ',
'serialization.null.format'='-')
STORED AS TEXTFILE;
LOAD DATA LOCAL INPATH 'examples/files/access_log' OVERWRITE INTO
TABLE apachelog;
```

Como se puede observar, en Hive se ha hecho uso de `SERDEPROPERTIES` con el fin de definir el formato de las cadenas que van a ser cargadas los datos, mediante una secuencia de expresión regular. Sobre el manejo básico de Hive Query Language, se ha usado la información disponible en su respectiva Wiki (15).

A continuación, presentamos el código para leer un log de Apache Web Server en Pig Latin (para Pig):

```
register /usr/lib/pig/contrib/piggybank/java/piggybank.jar;
DEFINE LogLoader
org.apache.pig.piggybank.storage.apachelog.CombinedLogLoader();
apachelog = LOAD 'apache_log' USING LogLoader as (remoteAddr,
remoteLogname, user, time, method,uri, proto, status, bytes,
referer,userAgent);
```

Asimismo, se ha hecho el uso de la librería Piggybank (16), que posee un conjunto de funcionalidades implementadas por comunidad de desarrollo de software libre para el manejo de ciertas tareas; en nuestro caso, la lectura de Apache Logs.

Por último, presentamos la parte más relevante de nuestro código implementado en Java (java native con hadoop):

```
public static class MapClass extends MapReduceBase implements Mapper
{
    private final static LongWritable ONE = new LongWritable(1);

    private static Pattern p = Pattern
        .compile("([^ ]*) ([^ ]*) ([^ ]*) \\[([^]]*)\\]
\"([^\"]*)\"" +
                        " ([^ ]*) ([^ ]*).*");

    private static DateTimeFormatter formatter = DateTimeFormat
        .forPattern("dd/MMM/yyyy:HH:mm:ss Z");

    private IntWritable minute = new IntWritable();

    public void map(WritableComparable key, Writable value,
        OutputCollector output, Reporter reporter) throws
IOException {

      String line = ((Text) value).toString();
      Matcher matcher = p.matcher(line);
      if (matcher.matches()) {
        String timestamp = matcher.group(4);
        minute.set(getMinuteBucket(timestamp));
        output.collect(minute, ONE);
      }

    }
```

Al final se obtuvieron los datos requeridos por cada consulta y en los formatos indicados en este capítulo. Dichas consultas fueron:1) IP que más ha realizado petición de recursos sobre el servidor. 2) La hora a la que se han producido más errores en el servidor. 3) El recurso que más errores ha generado en el servidor y la hora a la que más veces lo ha hecho.

3.3 Información de la IP que más ha visitado el servidor

Para obtener la información referente a la primera consulta, primero se han cargado los datos contenidos en los logs de Apache, luego se los ha procesado para obtener un documento de texto, el mismo que contiene dos columnas de información: la primera columna en la que se indica las direcciones IP que han visitado el servidor y la segunda columna que me enumera cuántas veces lo ha hecho. Los campos están separados por tabulaciones con el siguiente formato:

Dirección ip	Número de veces
190.131.22.103	123

Tabla 1. Formato para información de dirección ip que más ha visitado el servidor

En la Figura 4, se puede observar el flujo de procesos para obtener dicha información.

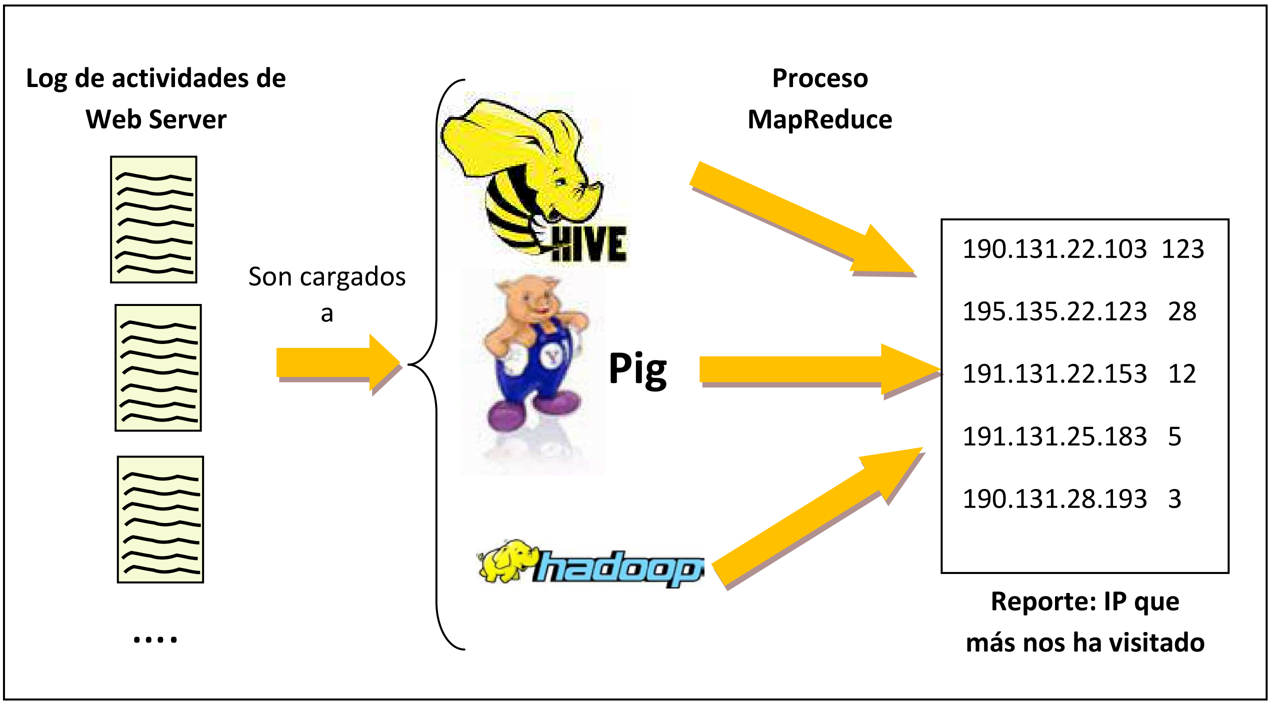

Figura 4 Flujo de procesos de la primera consulta.

A continuación presentamos las partes más relevantes del código de esta consulta implementados en Hive Query Language, Pig latin y Java(Hadoop).

Hive QL

```
SELECT ipaddress, COUNT(1) AS numrequest FROM apachelog GROUP BY
ipaddress SORT BY numrequest DESC LIMIT 1;
```

Pig Latin

```
groupP = GROUP apachelog BY remoteAddr;
ipcounter = FOREACH groupP GENERATE group, COUNT(apachelog) AS cnt;
ipcounter = ORDER ipcounter BY cnt DESC;
```

Java (Hadoop)

Map

```
public void map(LongWritable key, Text value,
OutputCollector<Text,IntWritable> output,
      Reporter reporter) throws IOException {
        StringTokenizer linea =new
StringTokenizer(value.toString(),"\n");
        StringBuilder comilla = new StringBuilder();
        String separador=comilla.append('"').toString();
    while (linea.hasMoreTokens()) {
      String registro = linea.nextToken();
      String [] aux=registro.split(separador);
      StringBuilder ip=new StringBuilder();
      int indice=aux[0].indexOf(' ');
      System.out.println(indice);
      ip.append(aux[0].substring(0,indice));//ip
      output.collect(new Text(ip.toString()),one);
    }
```

```
   }
```

Reduce

```
public void reduce(Text key, Iterator<IntWritable> values,
OutputCollector<Text, IntWritable> output, Reporter reporter) throws
IOException {
             int sum = 0;
             while (values.hasNext()) {
                   sum += ((IntWritable)values.next()).get();
             }
             output.collect(key, new IntWritable(sum));
        }
```

3.4 Información de la hora a la que se han producido la mayor cantidad de errores en el servidor

Como se lo definió en la introducción del presente trabajo investigativo, y en la sección de flujo de datos del Capítulo 2, para obtener dicho reporte se ha dividido el proceso en dos partes.

La primera parte, consiste en sacar un reporte de los registros del log de apache, en los cuales el status de la petición realizada al servidor es diferente de 200. Dicho reporte posee el siguiente formato:

finishtime
06/Dec/2009:04:13:07 -0500

Tabla 2. Formato con información de la fecha y hora en la cual se ha producido errores

Luego, con los datos obtenidos de la primera parte, se la vuelve a procesar. En esta fase, se separa la fecha en las columnas: día, mes, año, hora, minuto y segundo, para posteriormente agrupar esta información por horas y contabilizar cuando se produce la mayor cantidad de errores.

Hora	Número de veces
08	523

Tabla 3. Formato con información de la hora en la cual se ha producido errores

A continuación presentamos la Figura 5, en la cual se describe gráficamente la última parte de esta consulta.

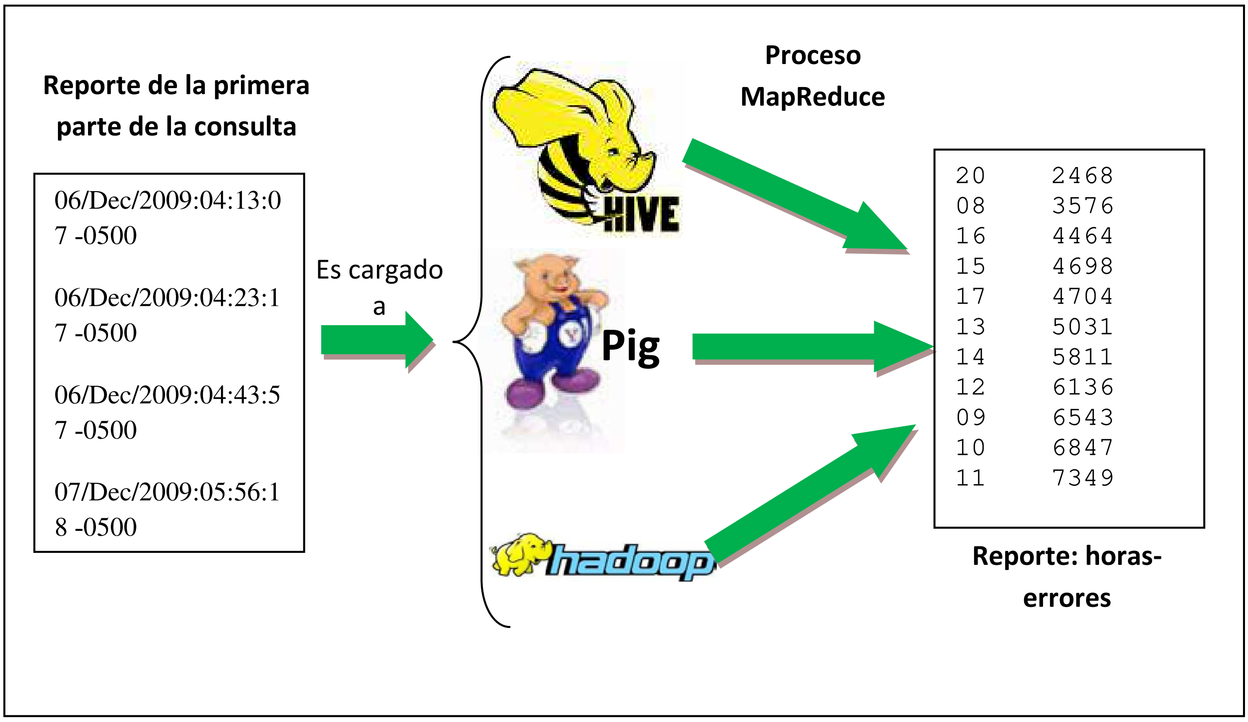

Figura 5. Flujo de procesos de la segunda parte de la segunda consulta, en lo cual se obtiene el listado de horas-errores encontrados

A continuación presentamos las partes más relevantes del código de esta consulta implementados en Hive Query Language, Pig latin y Java(Hadoop).

Hive QL

```
SELECT finishtime FROM apachelog WHERE returncode<>200;
SELECT hora,COUNT(1) AS numerrores FROM apachelogAnalisys GROUP BY
hora SORT BY numerrores DESC;
```

Pig Latin

```
B= FILTER apachelog BY status!='200';
C= FOREACH B GENERATE time;
```

Y la segunda parte de la consulta es:

```
apachelogAnalisys2 = GROUP apachelogAnalisys BY hora;
apachelogAnalisys3 = FOREACH apachelogAnalisys2 GENERATE group,
COUNT(apachelogAnalisys) AS cnt;
apachelogAnalisys3 = ORDER apachelogAnalisys3 BY cnt DESC;
```

Java (Hadoop)

Map

```
public void map(LongWritable key, Text value,
OutputCollector<Text,IntWritable> output,
      Reporter reporter) throws IOException {
        StringTokenizer linea =new
StringTokenizer(value.toString(),"\n");
        StringBuilder comilla = new StringBuilder();
        String separador=comilla.append('"').toString();
    while (linea.hasMoreTokens()) {
      String registro = linea.nextToken();
      String [] aux=registro.split(separador);
      StringBuilder hora_error=new StringBuilder();
      int indice=aux[0].indexOf(':')+1;
      hora_error.append(aux[0].substring(indice,indice+2));
      indice=aux[2].indexOf(' ');
      if(indice!=-1){
        if(aux[2].substring(1,4).compareTo("200") != 0)
            output.collect(new Text(hora_error.toString()),one);
      }
    }
  }
```

Reduce

```
public void reduce(Text key, Iterator<IntWritable> values,
OutputCollector<Text, IntWritable> output, Reporter reporter) throws
IOException {
             int sum = 0;
             while (values.hasNext()) {
                   sum += ((IntWritable)values.next()).get();
             }
             output.collect(key, new IntWritable(sum));
       }
```

3.5 Información de la página o recurso que más veces ha producido error y a qué hora lo ha generado más veces

Con un esquema parecido a la consulta anterior, también involucra dos procesos: la primera parte de la consulta, consiste en cargar los datos del log de Apache en las tres herramientas a evaluar; luego se ha realizado la consulta, acerca de cuál recurso o pagina es la que más veces ha generado mensajes de error. El informe que me devuelve dicho proceso esta especificado en la siguiente tabla:

Página	Número de veces
/templates/fiec_inicio_template/css/template_css.css	2776

Tabla 4. Formato con información, de la página o recurso que ha producido más errores

En la segunda parte del proceso, se lee los datos generados por la primera parte del mismo; luego se realiza la operación join (intersección) entre recurso que más errores ha generado y los datos leídos del log de apache, en los cuales el estado es diferente de 200. A la información obtenida, la separamos por día, mes, año, hora, minutos y segundos. Finalmente se obtiene un reporte indicando a qué hora es a la que se ha producido la mayor cantidad de errores. Dicho reporte tiene el mismo formato del que se obtuvo en el segundo proceso de la segunda consulta. Ver Tabla 5.

Hora	Número de veces
08	523

Tabla 5. Formato con información de la hora en la cual se ha producido errores

A continuación presentamos las partes más relevantes del código de esta consulta implementados en Hive Query Language, Pig latin y Java(Hadoop).

Hive QL

```
SELECT apachelog.finishtime FROM (SELECT requestline,COUNT(1) AS
numerrores FROM apachelog WHERE returncode<>200 GROUP BY requestline
SORT BY numerrores DESC LIMIT 1)tmp JOIN apachelog
ON(tmp.requestline=apachelog.requestline)WHERE
apachelog.returncode<>200;
```

Y la segunda parte de la consulta:

```
SELECT hora,COUNT(1) AS numerrores FROM apachelogAnalisys GROUP BY
hora SORT BY numerrores DESC;
```

Pig Latin

```
B = FILTER apachelog BY status!='200';
C = GROUP B BY uri;
D = FOREACH C GENERATE group,COUNT(B) as cnt;
D = ORDER D BY cnt DESC;
E = LIMIT D 1;
DUMP E;-- solo para presentar los datos
F = JOIN E BY group,B BY uri;
G = FOREACH F GENERATE time,uri;
```

Y la segunda parte de la consulta:

```
apachelogAnalisys2 = GROUP apachelogAnalisys BY hora;
apachelogAnalisys3 = FOREACH apachelogAnalisys2 GENERATE group,
COUNT(apachelogAnalisys) AS cnt;
apachelogAnalisys3 = ORDER apachelogAnalisys3 BY cnt DESC;
apachelogAnalisys4 = UNION E,apachelogAnalisys3;
```

<u>Java (Hadoop)</u>

Map

```
public void map(LongWritable key, Text value,
OutputCollector<Text,IntWritable> output,
      Reporter reporter) throws IOException {
        StringTokenizer linea =new
StringTokenizer(value.toString(),"\n");
        StringBuilder comilla = new StringBuilder();
        String separador=comilla.append('"').toString();
    while (linea.hasMoreTokens()) {
      String registro = linea.nextToken();
      String [] aux=registro.split(separador);
      StringBuilder pag_error=new StringBuilder();
      int indice=aux[1].indexOf(' ')+1;
      String recurso=aux[1].substring(indice,aux[1].length());
      int indicefin=recurso.indexOf(' ');
      pag_error.append(aux[1].substring(indice,aux[1].length()-
recurso.length()+indicefin));//recurso
      indice=aux[2].indexOf(' ');
      if(indice!=-1){
        if(aux[2].substring(1,4).compareTo("200") != 0)
            output.collect(new Text(pag_error.toString()),one);
      }
    }
  }
```

Reduce

Es el mismo que el de la sección anterior.

4. EVALUACIÓN Y RESULTADOS

4.1 Decisiones de versionamiento

Las versiones utilizadas en este caso de estudio fueron, Hadoop 0.18, Pig 0.5, Hive 0.4.0. Todas estas versiones indicadas, trabajan establemente en la imagen Fedora de Cloudera que nos provee Amazon Machine Image (AMI), con la que ejecutamos las pruebas como se lo indicó en la sección 3.2.

4.2 Tiempos de ejecución

Las variables para cada consulta fueron las siguientes:

- Cantidad de nodos en el clúster
- Tiempo de respuesta en segundos

Estos resultados muestran el comportamiento con un log de 1GB de tamaño. Los valores obtenidos pueden variar con tamaños de logs mayores. Cabe recalcar que los tiempos medidos en todas las consultas, fue solo el de procesamiento de la misma, excluyendo el tiempo de carga de datos en el servicio S3 que nos provee Amazon Web Services. Lo excluimos puesto que lo que se quiere evaluar es la eficiencia de las herramientas al procesar consultas.

4.2.1 Consultas sobre Java nativo en Hadoop

Para las tres consultas realizadas sobre Java nativo de Hadoop, se obtuvieron los siguientes resultados de tiempo medido en segundos:

Nodos	2	4	6	10	15	20
Consulta 1	93	59	40	31	28	27
Consulta 2	72	51	36	27	24	21
Consulta 3	71	51	37	28	24	20

Tabla 6 Tiempo promedio en segundos de las consultas sobre Java nativo de Hadoop

La respectiva gráfica de cantidad de Nodos vs. Tiempo se la puede ver en la siguiente figura:

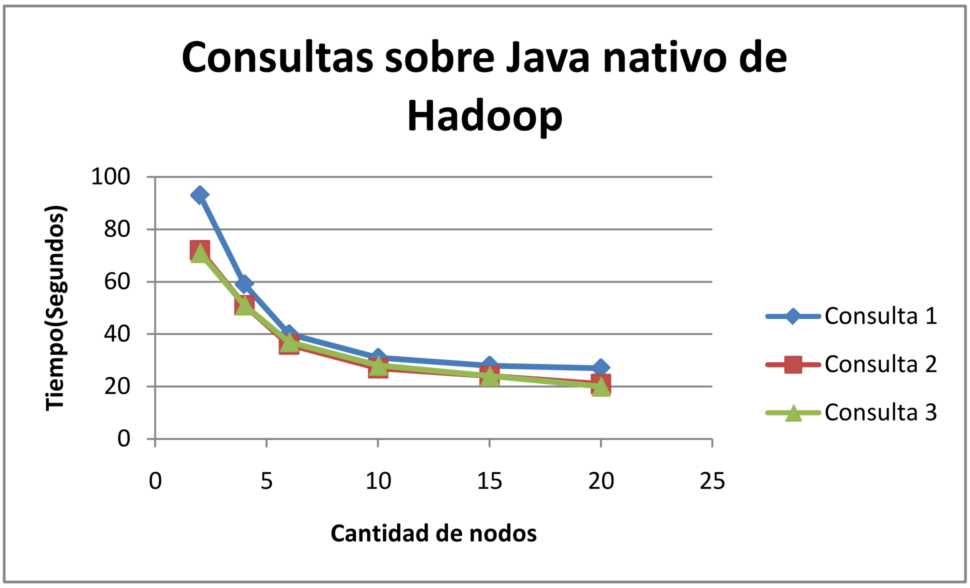

Figura 6 Gráfica de cantidad de nodos en el clúster versus tiempo tomado para las consultas realizadas sobre Hadoop

4.2.2 Consultas sobre Pig

Para las tres consultas realizadas sobre Pig, se obtuvieron los siguientes resultados de tiempo medido en segundos:

Nodos	2	4	6	10	15	20
Consulta 1	172	126	93	76	75	74
Consulta 2	133	74	63	53	52	69
Consulta 3	279	201	171	154	154	142

Tabla 7 Tiempo promedio en segundos de las consultas sobre Pig

La respectiva gráfica de cantidad de Nodos vs Tiempo se la puede ver en la siguiente figura:

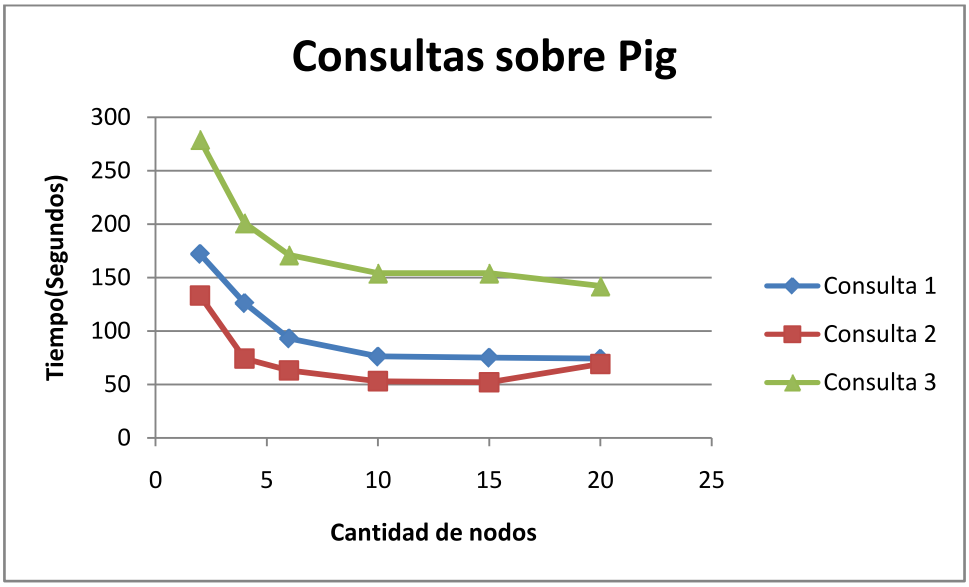

Figura 7 Gráfica de cantidad de nodos en el clúster versus tiempo tomado para las consultas realizadas sobre Pig

4.2.3 Consultas sobre Hive

Para las tres consultas realizadas sobre Hive, se obtuvieron los siguientes resultados de tiempo medido en segundos:

Nodos	2	4	6	10	15	20
Consulta 1	4414.7513	2852.0139	1588.4087	1548.1025	1499.1968	1470.5606
Consulta 2	4352.8812	2834.0337	1561.4006	1415.2823	1384.61	1359.4257
Consulta 3	8898.2346	5776.0148	3117.9123	3087.789	2889.1818	2880.6259

Tabla 8 Tiempo promedio en segundos de las consultas sobre Hive

La respectiva gráfica de cantidad de Nodos vs Tiempo se la puede ver en la siguiente figura:

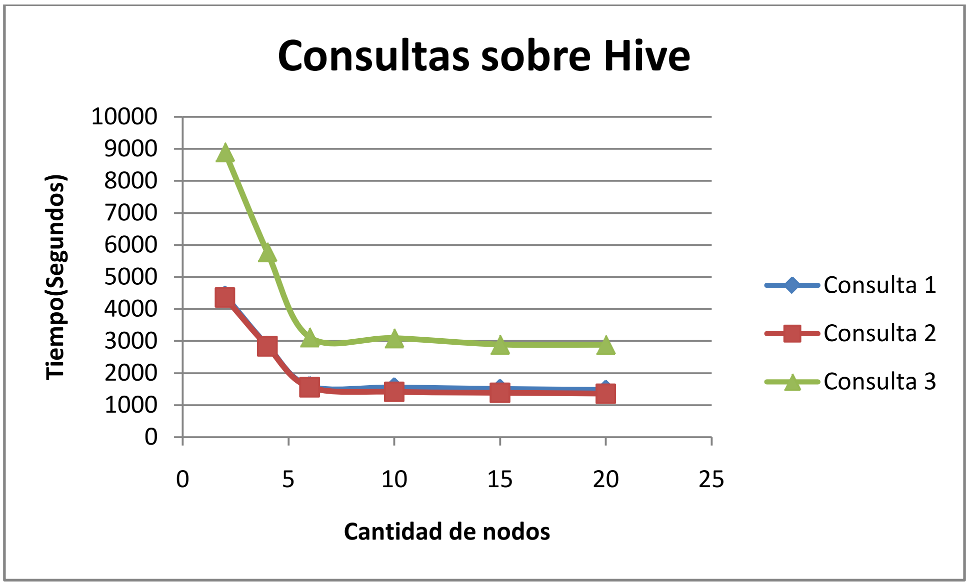

Figura 8 Gráfica de cantidad de nodos en el clúster versus tiempo tomado para las consultas realizadas sobre Hive

4.2.4 Comparación de herramientas en la primera consulta

La siguiente figura nos muestra la comparación de tiempo de ejecución de la primera consulta en las tres herramientas evaluadas.

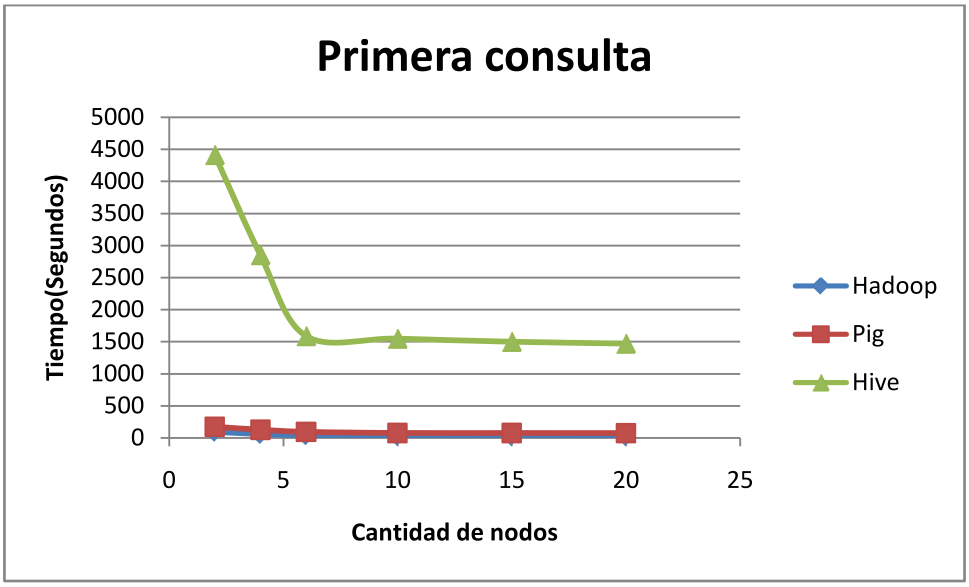

Figura 9 Gráfica de cantidad de nodos en el clúster versus tiempo tomado para la primera consulta realizada sobre Hadoop,Pig y Hive.

Como se puede ver en la gráfica, el tiempo que toma la herramienta Hive en procesar dicha tarea, es muy alto en comparación con las otras dos herramientas; por esta razón, presentamos a continuación el gráfico pero comparando solo Hadoop y Pig.

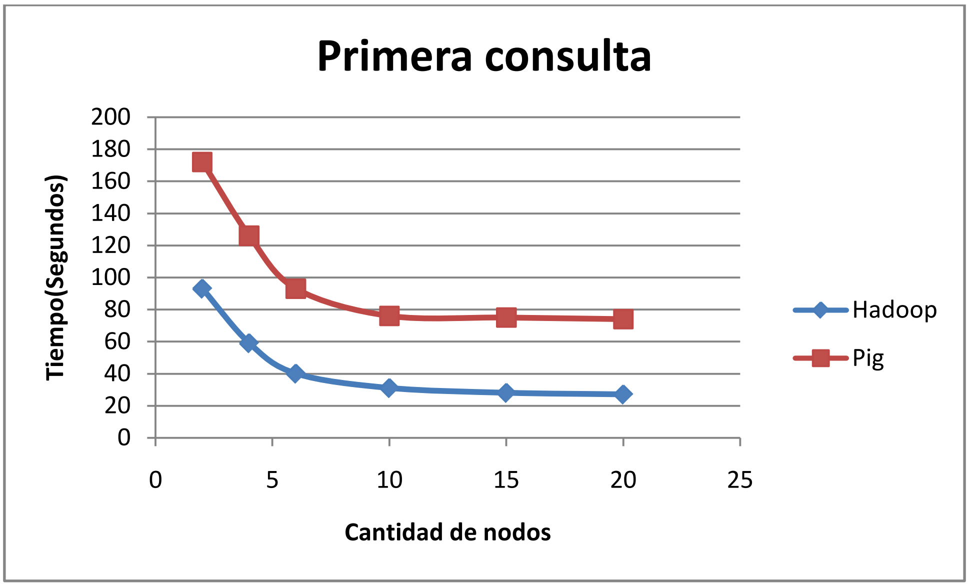

Figura 10 Gráfica de cantidad de nodos en el clúster versus tiempo tomado para la primera consulta realizada sobre Hadoop y Pig.

Claramente se puede ver la superioridad de Hadoop en tiempo de respuesta con respecto a Pig. La diferencia del tiempo que toma Pig en responder una tarea, es cercana al doble del tiempo que toma la misma tarea en Java (Hadoop). Esta situación se da, puesto que Pig revisa las sentencias escritas en PigLatin, para luego transformarlas a sentencias MapReduce, y el código MapReduce nativo generado generalmente no es tan eficiente como el código que se puede escribir directamente en Java. Esto supone un aumento de tiempo (cercano al 100% en esta consulta) de lo que tomaría ejecutar un proceso escrito directamente en Java.

Sin embargo, para la mayoría de los trabajos de minería de datos, lo que se requiere es que el grupo de trabajo que va a ejecutar dicho proceso, use una herramienta de fácil implementación como lo son Pig y Hive. Sin embargo, la figura 10 nos muestra la superioridad de Pig con respecto a Hive.

4.2.5 Comparación de herramientas en la segunda consulta

La siguiente figura nos muestra la comparación de tiempo de ejecución de la segunda consulta en las tres herramientas evaluadas.

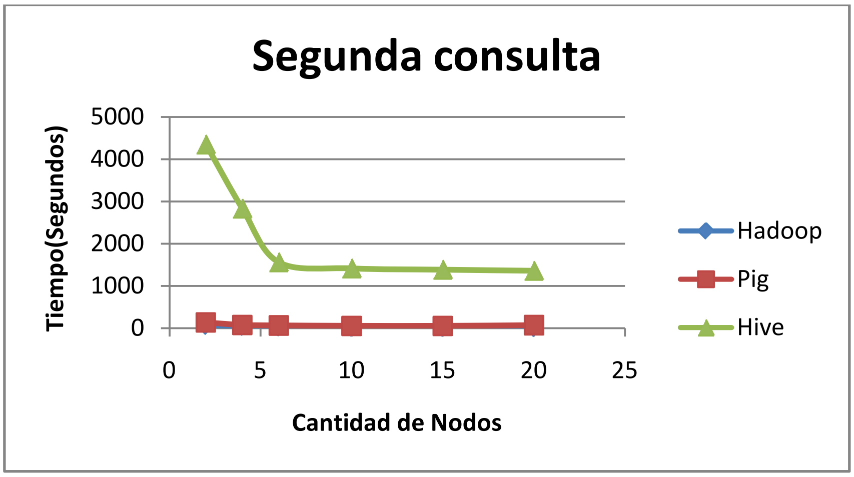

Figura 11 Gráfica de cantidad de nodos en el clúster versus tiempo tomado para la segunda consulta realizada sobre Hadoop, Pig y Hive.

A continuación, presentamos con más claridad la comparación de tiempos de ejecución entre Pig y Hadoop.

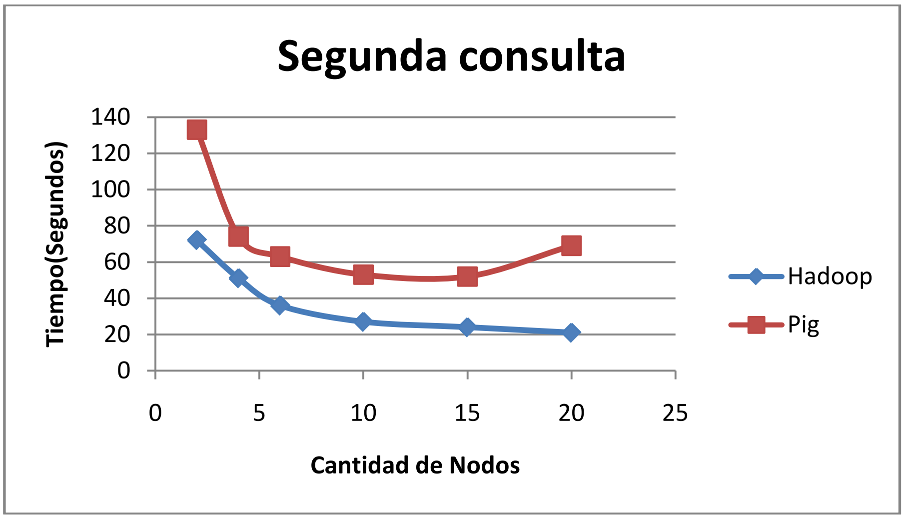

Figura 12 Gráfica de cantidad de nodos en el clúster versus tiempo tomado para la segunda consulta realizada sobre Hadoop y Pig

Por lo que se puede observar, al comparar los tiempos entre las tres herramientas en la segunda consulta propuesta; es que aporta el criterio expuesto en la sección 4.2.4.

4.2.6 Comparación de herramientas en la tercera consulta

La siguiente figura nos muestra la comparación de tiempo de ejecución de la tercera consulta en las tres herramientas evaluadas.

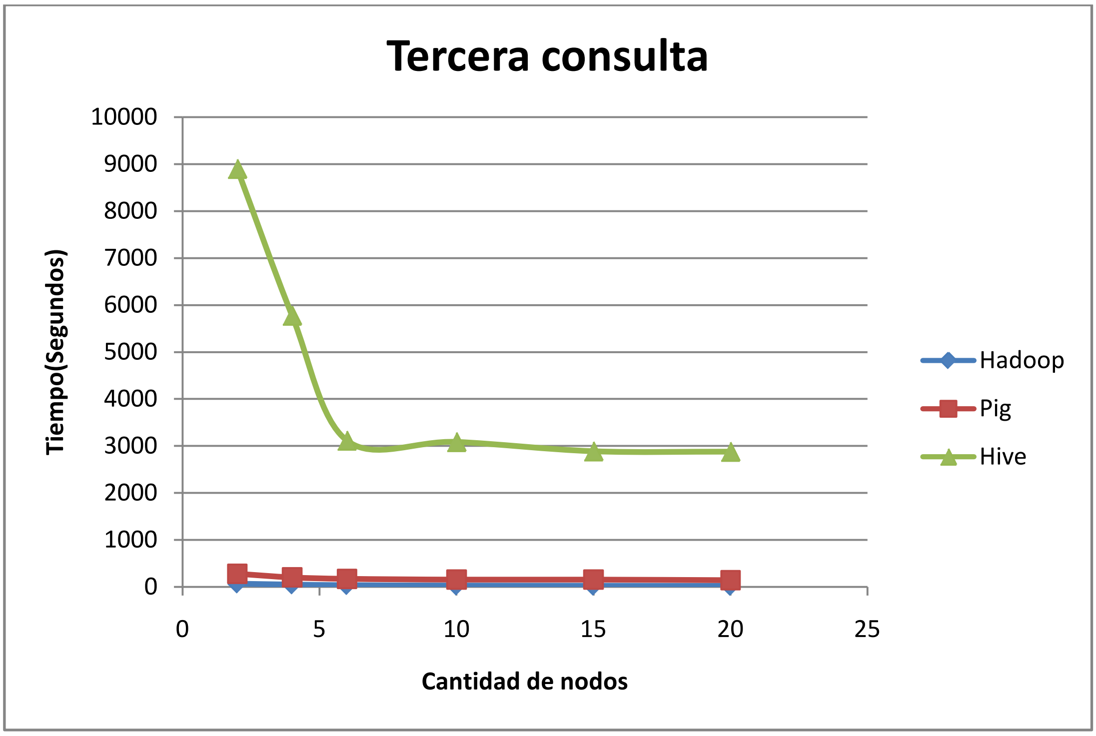

Figura 13Gráfica de cantidad de nodos en el clúster versus tiempo tomado para la tercera consulta realizada sobre Hadoop, Pig y Hive

Asimismo como en las secciones 4.2.4 y 4.2.5, presentamos con más claridad la comparación de tiempos de ejecución entre Pig y Hadoop

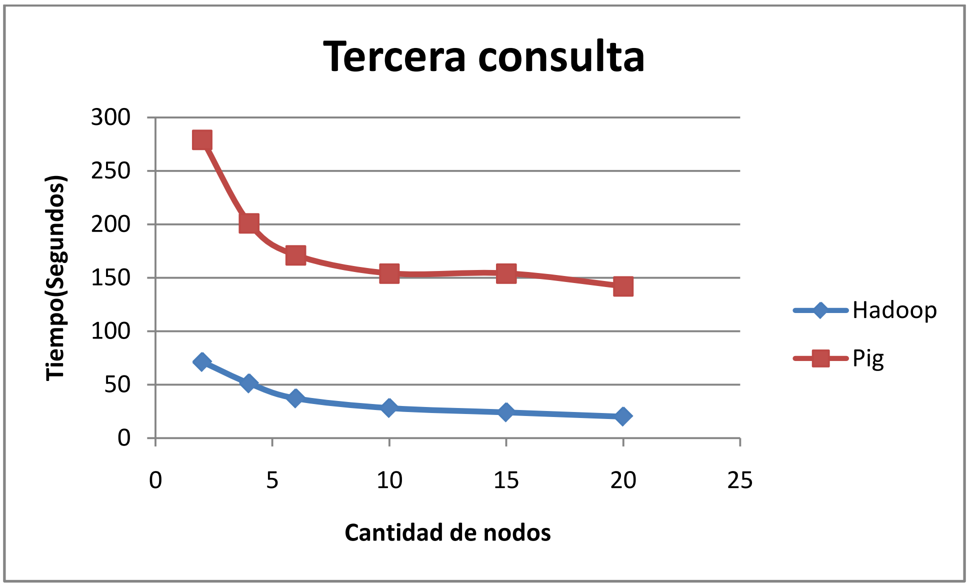

Figura 14 Gráfica de cantidad de nodos en el clúster versus tiempo tomado para la tercera consulta realizada sobre Hadoop, Pig y Hive

Pese a que el criterio expuesto en la sección 4.2.4 sobre Hadoop y Pig es reafirmado en este caso; el aumento de tiempo de Pig con respecto a Hadoop en esta consulta es aproximadamente al 250%. Esto se debe a que el código escrito en Java nativo de Hadoop puede ser optimizado; mientras que al escribir sentencias en PigLatín, se le limita al programador dicha optimización; pues es Pig quien se encarga de traducir las sentencias en tareas MapReduce.

4.2.7 Comparación de herramientas con respecto al número de nodos

Para determinar si el tamaño del clúster influye en la eficiencia en términos de tiempo, a las tres herramientas evaluadas, se ha hecho un análisis inferencial mediante el uso de la distribución T de student.

En nuestra investigación, se han realizado 30 observaciones por cada variación de tamaño de clúster lo cual nos da 179 grados de libertad.

Los datos obtenidos fueron los siguientes:

Hadoop	Correlación	Significancia estadística
Consulta 1	-0.686623	1.39E-06
Consulta 2	-0.7078144	2.55E-07
Consulta 3	-0.7147704	1.41E-07

Tabla 9 Coeficientes de correlación entre la eficiencia de tiempo y el número de nodos del clúster para la herramienta Hadoop (Java nativo).

Pig	Correlación	Significancia estadística
Consulta 1	-0.7864096	9.63E-14
Consulta 2	-0.6801039	2.28E-06
Consulta 3	-0.7843727	1.23E-13

Tabla 10 Coeficientes de correlación entre la eficiencia de tiempo y el número de nodos del clúster para la herramienta Pig.

Hive	Correlación	Significancia estadística
Consulta 1	-0.7225381	7.16E-11
Consulta 2	-0.7378843	1.74E-08
Consulta 3	-0.7274289	4.61E-08

Tabla 11 Coeficientes de correlación entre la eficiencia de tiempo y el número de nodos del clúster para la herramienta Hive.

De las tablas 9,10 y 11 se puede observar claramente que la eficiencia en tiempo de procesamiento en las tres herramientas, están correlacionados con el tamaño del clúster en donde se procesan. El valor de significancia estadística es alto (p=99%).

Sin embargo, cabe recalcar que el valor del coeficiente de correlación es negativo, esto implica que mientras aumenta el número de nodos del clúster, disminuye el tiempo de procesamiento.

4.2.8 Comparación de herramientas con respecto al tamaño de Apache Log

Para realizar esta parte de nuestro trabajo investigativo, se ha decido evaluar las consultas usando un clúster de 10 nodos. La razón para escoger dicho tamaño, es porque en nuestras consultas procesadas sobre las tres herramientas, pudimos observar que al ejecutar los trabajos en un clúster de 10 nodos, se logra una excelente eficiencia en tiempo. Es decir, que si se añaden mas nodos (más de 10), el decremento en tiempo por la tarea es de pocos segundos; llegando a ser despreciable.

A continuación, presentamos las tablas de datos de cada consulta con los tiempos medidos en segundos:

Tamaño del Log (MB)	128	256	512	1024
Hive	879	886	907	1548
Pig	69	70	71	76
Java Nativo (Hadoop)	22	22	24	31

Tabla 12 Tiempo promedio en segundos de la primera consulta

Tamaño del Log (MB)	128	256	512	1024
Hive	868	888	894	1415
Pig	41	41	42	53
Java Nativo (Hadoop)	18	20	22	28

Tabla 13 Tiempo promedio en segundos de la segunda consulta

Tamaño del Log (MB)	128	256	512	1024
Hive	1785	1792	1808	3087
Pig	88	88	89	154
Java Nativo (Hadoop)	16	18	20	28

Tabla 14 Tiempo promedio en segundos de la tercera consulta

Para realizar las respectivas observaciones, se ha sobrepuesto todas las curvas de tiempo en un solo gráfico, el cual lo presentamos a continuación:

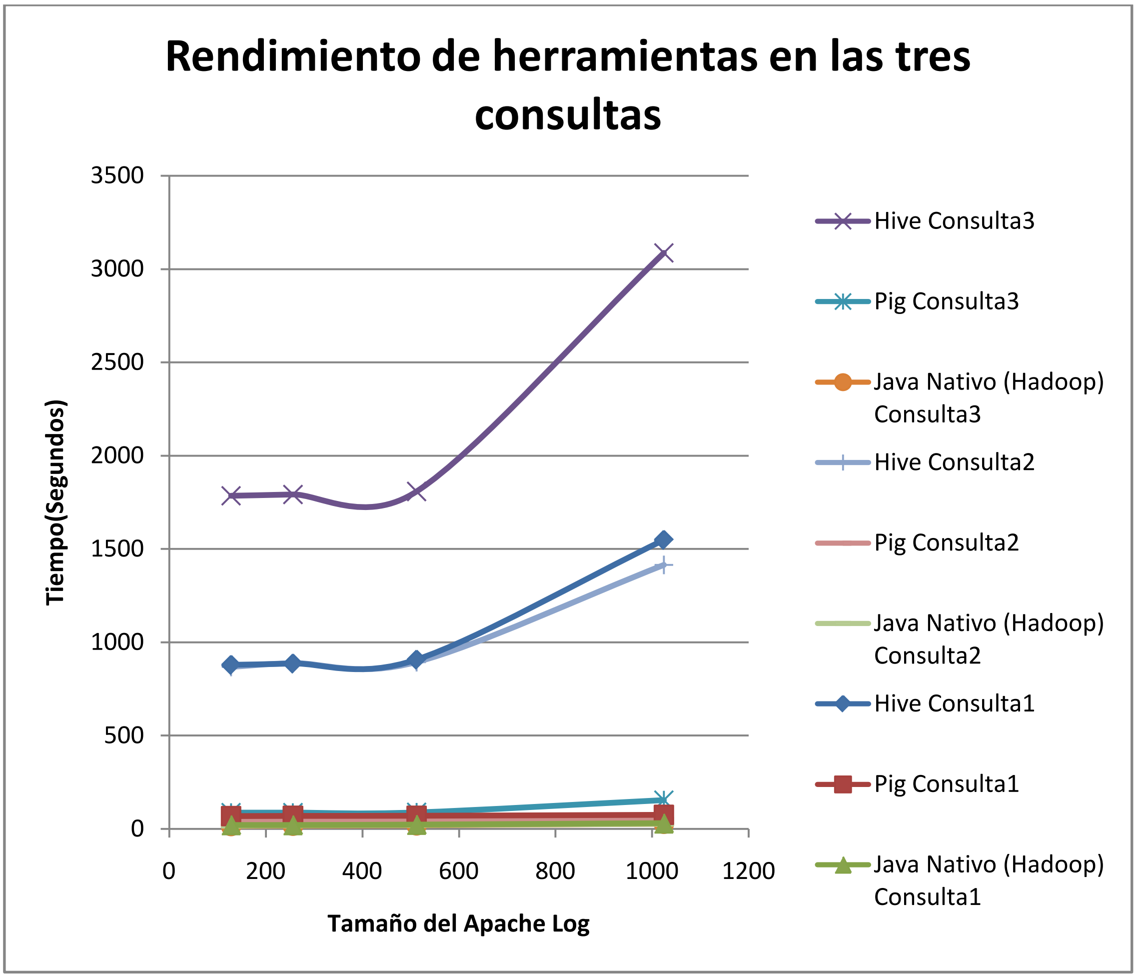

Figura 15 Gráfica de tamaño del Apache Log versus tiempo tomado para las tres consultas realizada sobre Hadoop, Pig y Hive

El comportamiento de Hive no es muy bueno con respecto a las otras dos herramientas evaluadas. Por otra parte, para la primera y segunda consulta, el comportamiento de Pig es bastante parecido al de Java nativo (Hadoop); sin embargo, en la tercera consulta se puede observar una ligera diferencia. Esto se lo puede ver de una forma más clara en la siguiente figura:

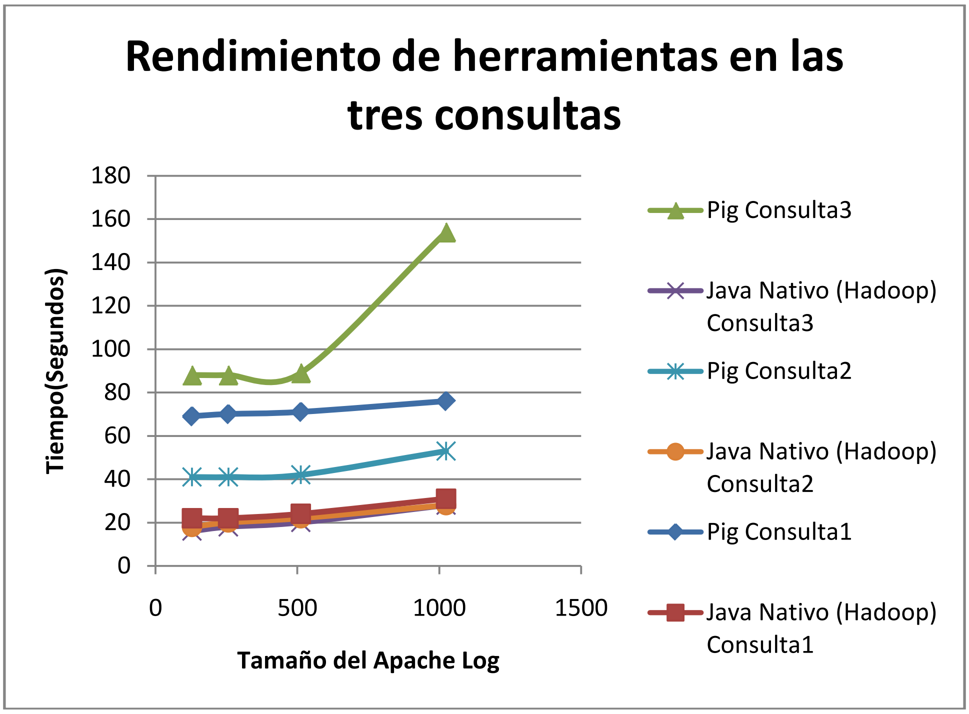

Figura 16 Gráfica de tamaño del Apache Log versus tiempo tomado para las tres consultas realizada sobre Hadoop y Pig.

Esta diferencia de Pig con respecto a Java nativo (Hadoop) reafirma el criterio expuesto sobre Pig en la sección 4.2.4.

Para determinar si el tamaño de los Apache Logs, influye en la eficiencia en términos de tiempo, a las tres herramientas evaluadas, se ha hecho un análisis inferencial mediante el uso de la distribución T de student.

Elegimos dicha distribución estadística, puesto que me permite obtener resultados estadísticamente confiables con un mínimo tamaño de muestra de 30 observaciones. En nuestra investigación, se han realizado 30 observaciones por cada variación de tamaño de Apache Log lo cual nos da 118 grados de libertad (120 observaciones).

Los datos obtenidos fueron los siguientes:

Hadoop	Correlación	Significancia estadística
Consulta 1	0.6093751	0.0003512
Consulta 2	0.843469	4.85E-06
Consulta 3	0.7084398	1.18E-02

Tabla 15 Coeficientes de correlación entre la eficiencia de tiempo y el tamaño de Apache Log para la herramienta Hadoop (Java nativo).

Pig	Correlación	Significancia estadística
Consulta 1	0.2567449	0.1708
Consulta 2	0.4447525	1.38E-02
Consulta 3	0.1553797	4.12E-01

Tabla 16 Coeficientes de correlación entre la eficiencia de tiempo y el tamaño de Apache Log para la herramienta Pig.

Hive	Correlación	Significancia estadística
Consulta 1	0.8654805	6.71E-07
Consulta 2	0.78687	2.51E-04
Consulta 3	0.8690335	4.72E-07

Tabla 17 Coeficientes de correlación entre la eficiencia de tiempo y el tamaño de Apache Log para la herramienta Hive.

De las tablas 15,16 y 17 se observa claramente que la eficiencia en tiempo de procesamiento en Java nativo de Hadoop como en Hive están correlacionados con el tamaño del Apache Log a procesar. El valor de significancia estadística es alto, el cual es del 99% de confiabilidad (p=0.01%).

Sin embargo, cabe recalcar que la herramienta Pig, tiene un coeficiente de correlación bajo, cuyo valor más alto para las consultas evaluadas fue de r=0.44 con significancia del 98% (consulta 2). Esto quiere decir que la eficiencia de Pig está bajamente correlacionada al tamaño de Apache Log.

CONCLUSIONES Y RECOMENDACIONES

Conclusiones

1. Desde el punto de vista de usabilidad, la herramienta que al momento más facilidad nos da para realizar trabajos de procesamiento masivo de datos en paralelo es Pig; pues el mismo hace uso de PigLatín para escribir sentencias. Además el aprendizaje de dicha herramienta nos resultó bastante rápido. Sin embargo hay que notar que es necesario que quien lo está aprendiendo tenga un nivel de experiencia programando en algún lenguaje.
2. También nos dimos cuenta de que si el usuario que desea realizar una tarea de procesamiento masivo de datos, no está interesado en la latencia de los mismos; y además desea que los datos cargados en el sistema queden almacenados en tablas, entonces Hive es la solución pues casi usa el mismo tipo de sentencias de SQL salvo ligeras modificaciones para el manejo de archivos.
3. Sin embargo, si un usuario desea realizar procesos distribuidos de la manera más óptima en tiempo de ejecución, es necesario el uso de la programación MapReduce en Java nativo de Hadoop y maximizar el excelente uso de recursos, para obtener el resultado más óptimo, pero sacrificando: 1) facilidad en escritura de código y 2) el tiempo empleado en la implementación de la solución.

4. Por otra parte, y de lo que pudimos ver, Pig tiene diferencias en tiempo de ejecución con respecto a Hadoop, por lo que hay que evaluar realmente que es lo que se desea sacrificar, poco tiempo de diferencia en procesamiento o facilidad en escritura de código, que en la mayoría de los casos la facilidad es vital pues de ella depende el costo de implementación.
5. Es importante hacer notar que la consulta 3, cuyo contenido esta descrito en la sección 3.5, es mucho más lenta en Hive y Pig que las otras consultas, pero no así en Java nativo, ya que en este caso la consulta 1 es ligeramente más lenta que las otras.
6. El beneficio que brinda este proyecto, es verificar cual herramienta es más eficiente, mediante los resultados de tiempo de ejecución, en las tres consultas realizadas sobre logs de actividades de servidor apache, en las tres herramientas propuestas para el procesamiento distribuido de datos.

Recomendaciones

1. El resultado de este análisis es mucho más beneficioso si se tiene la mayor cantidad de características del tipo de tarea que el usuario desea realizar.
2. La minería de datos de archivos de logs de apache usando la plataforma de Hadoop, Hive y Pig como herramientas para procesamiento masivo de datos, nos ha sido útil para poder generar información vital si es que se desea analizar problemas de seguridad y recursos demonios (servicios de alta o baja disponibilidad).
3. A menos que sea de vital importancia el tiempo, es mejor elegir a Pig como herramienta de procesamiento masivo de datos puesto que brinda facilidad de código y la diferencia de tiempo con respecto a la implementación más óptima en una determinada tarea es mínimo.
4. En nuestras consultas procesadas en las tres herramientas, pudimos observar que al ejecutar los trabajos en un clúster de 10 nodos, se logra una excelente eficiencia en tiempo. Es decir, que si se añaden mas nodos (más de 10), el decremento en tiempo por la tarea es de pocos segundos; llegando a ser despreciable. Por ello recomendamos que para tareas sobre logs del orden de Gigabytes, usar solamente diez nodos dependiendo de la tarea ya que sería un desperdicio de recursos.

Bibliografía

1. **Astorga, Agustín.** La importancia de las bitácoras. Disponible en http://www.allbusiness.com/information/information-services/773243-1.html. 19 de Enero de 2004 .

2. **Ghemawat, J. Dean, S.** *MapReduce: Simplified Data Processing on Large Clusters.* San Francisco, CA-EE.UU : En memorias del Sixth Symposium on Operating System Design and Implementation (OSDI 2004), Diciembre, 2004.

3. **The Apache Software Foundation.** Hive. [En línea] 22 de Enero de 2010. Disponible en http://wiki.apache.org/hadoop/Hive.

4. **The Apache Software Foundation.** Pig. Disponible en http://hadoop.apache.org/pig/. 22 de Enero de 2010.

5. **C. Olston, B. Reed, U. Srivastava, R. Kumar and A. Tomkins.** *"Pig Latin: A Not-So-Foreign Language for Data Processing".* Vancouver, Canada : s.n., June 2008. Vol. ACM SIGMOD 2008 International Conference on Management of Data.

6. **The Apache Software Foundation.** Hadoop. Disponible en http://hadoop.apache.org. 22 de Enero de 2010.

7. **The Apache Software Foundation.** EC2. Disponible en http://aws.amazon.com/ec2/.22 de Enero de 2010.

8. **Rodrigo Zamora Nelson.** La importancia de los logs en apache. Disponible en http://rodrigo.zamoranelson.cl/?p=322. 22 de Enero de 2010.

9. **White, Tom.** *Hadoop: the definitive guide.* s.l. : O'Reilly, 2009.

10. **Ghemawat, S., Gobioff, H., y Leung, S.** *"The Google File System".* Lake George NY-EE.UU : En Memorias del 19th ACM Symposium on Operating Systems Principles, Octubre, 2003.

11. **Tanenbaum, Andrew S.** *Sistemas Distribuidos: Principios y Paradigmas.* México : Prentice Hall, 1996. ISBN 0-13-219908-4.

12. **Amazon.** Amazon Web Services. Disponible en http://aws.amazon.com/.22 de Enero de 2010.

13. **Amazon.** S3. Disponible en http://aws.amazon.com/s3/.22 de Enero de 2010.

14. **Apache HTTP Server Documentation Project.** Estructura de log de apache. Disponible en http://httpd.apache.org/docs/1.3/logs.html#accesslog. 22 de Enero de 2010.

15. **The Apache Software Foundation.** Hive: Getting Started. Disponible http://wiki.apache.org/hadoop/Hive/GettingStarted. 22 de Enero de 2010

16. **The Apache Software Foundation.** PiggyBank. Disponible en http://wiki.apache.org/pig/PiggyBank. 22 de Enero de 2010.

17. **Amazon.** What is AWS?. http://aws.amazon.com/what-is-aws/.26 de Agosto de 2009.