System Log capture & Processing for better analysis
Detail Architecture and Software Design

July 7, 2013

Group 3 – Participants

Anirban Roy Choudhury (EEPITM-01-001)

Siddharth Verma (EEPITM-01-023)

Vikram Sahu (EEPITM-01-030)

Satyajit Mohanty (EEPITM-01-020)

Table of Contents

1 Introduction

The products of the company are installed at different clients and each product as and when any event occurs in the system, it keeps sending logs to the company HQ. The logs come with loads of information and the information typically is about the health of the system like the current firmware installed on the system, the OS running in the system as well as information about different types devices attached to the system etc. This logs information have to be processed immediately and lot of analysis need to be done as there can be a need to take corrective action as well as pro-active action can be taken to help the client business to run smoothly without any road blocks. The need is to process the logs and make it available for analysis in a structured manner and the expectation is to finish the processing of each log within 15 minutes from the time of arrival at the HQ. Typically the current system takes about 20-60 minutes to pull the information from the logs and make it available for any analysis and by the time the data is uploaded to a structured data set, there is a latency of 24 hours. With more and more system being installed at the clients, lots of logs started coming to the HQ for processing and some came in with older text format because of legacy system and newer ones started delivering with XML format, it quickly grew into a 3V problem (big data) that is very high volume, lot of variety and came in with lot of velocity. So the current BI system could not cater to this problem.

The need was without compromising on the performance even if the volume is growing and spending not much money, to process the logs within 15 minutes from the arrival, so the adhoc analysis can be done quickly as well as the required data can be sent for reporting and dashboards to keep the business running.

2 BIG Idea

The idea was to experiment using the available hardware and Hadoop framework (CDH3) to process the logs, use Pentaho Open source for ETL as well as for traditional reports, dashboards along with OBIEE and Datameer for data mining and adhoc analysis.

Other than the scalability and cost saving need, here are some of the other needs that are planned to be addressed in this effort

2.1 Business Impact - Functional

1. What has been tested with what? This will provide a confidence to the people stating that where a system is fully tested or not before it is released for GA.
2. On what software configuration as well as HW configuration the system is providing the best performance
3. Recommendation to the new clients about the systems that they may need based on the systems that have been implemented at different existing clients which are based on the different types of workload they face at the different time period.
4. Invoicing to the client based on the volume usage, where company has provided the system as an infrastructure service.
5. Negotiating with HDD vendors for better price based on their performance when attached to a system

6. Recommending existing client to move away from legacy system to new system and projecting the savings they will make by moving to new system.
7. This will help the company to manage one line of code and technology and less manpower to manage the show.
8. When integrated with Traditional BI, it will have the necessary data on time to provide the data related to Hourly project booking , purchase order, billing on reserve, billing off reserve, deferred revenue, non-deferred revenue etc.
9. Create a segmentation of clients based on the number of systems that have been installed at the site, amount of data they produce, consume and the different types of workloads.
10. Bookings and Installed Base data provides us with valuable information about the systems:
    - Platform mode, OS version, HW options, Storage type and capacity, SW licenses etc.

But this information reflects only what was true at the time of purchase. It is also very valuable for us to know what is happening at the customer site as time goes by, typical information that will be helpful

- How are our products being used?
- How frequently does the customer make upgrades?
- What is the application environment at the customer site?
- Do they use virtualization?
- At what rate are they adding storage?
- Which protocols are they using the most/least?
- What SW are they using and how often?
- What storage efficiency features are they using and how much savings are they achieving from that?
- How does taking advantage of our storage efficiency features affect their subsequent buying patterns?

11. This will marry the non structured data with structured data, position the company in the big data space and create a niche in the market.

2.2 Methodology / Approach

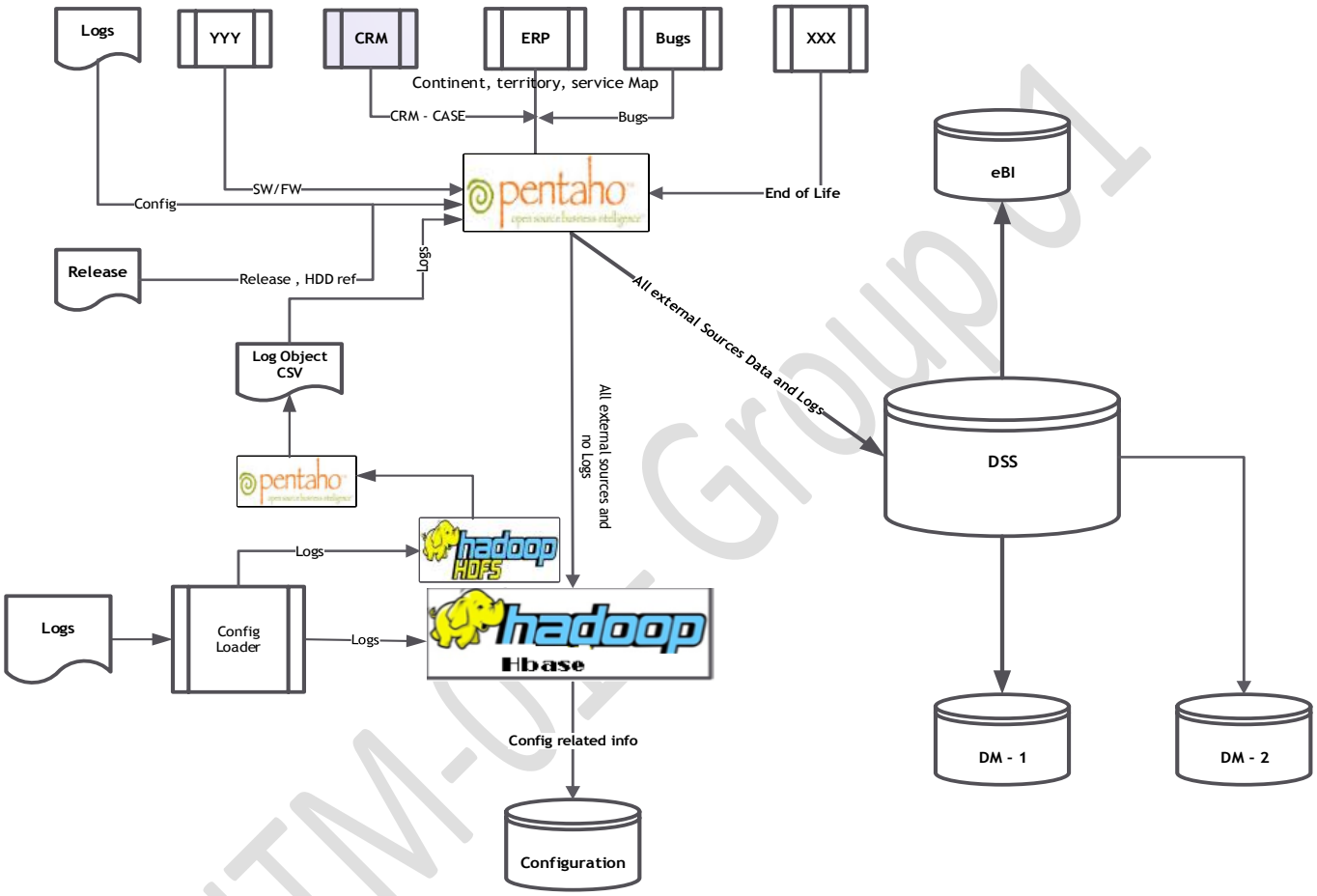
It will be a combination of waterfall and scrum with a combination of framework from Kimball and Inmon, following dependent datamart to hub and spoke architecture. The schemas will as well be a combination of STAR and SNOWFLAKE.

Tools and technologies

STAGE ODS	Datawarehouse Datamart	Reporting / Dashboards / Ad Hoc Analysis / Mining
Fame work – Hadoop distributed file system – CDH3 Flume, Sqoop, Pig, Hive, Hbase, Java, Perl	Oracle - DB Pentaho - ETL	Pentaho – BI OBIEE – BI Datameer

3 BIG Idea –Architectural Overview

3.1 Big Picture



DW - Next Generation

3.2 Data loading from Logs to HDFS (STAGE/ODS)

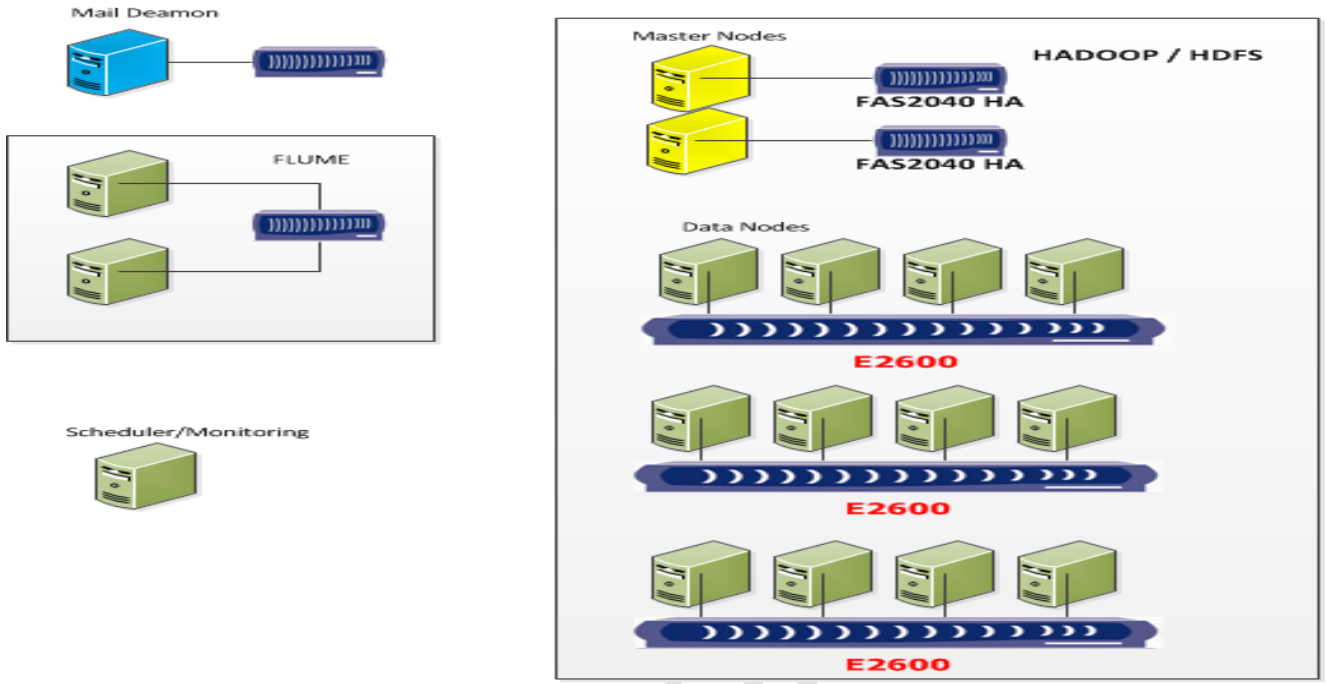
In this section, we will be illustrating the architecture for Hadoop implementation and how the different components for CDH3 will be playing a role to make the data available in HDFS without compromising on the non functional requirement like performance, scalability etc. as defined above.

Understanding CDH3 - Cloudera Hadoop version 3 Components

CDH3 includes several components that you can install and use with Apache Hadoop:

- ***Flume*** — A distributed, reliable, and available service for efficiently moving large amounts of data as the data is produced. This release provides a scalable conduit to shipping data around a cluster and concentrates on reliable logging. The primary use case is as a logging system that gathers a set of log files on every machine in a cluster and aggregates them to a centralized persistent store such as HDFS.
- ***Sqoop*** — A tool that imports data from relational databases into Hadoop clusters. Using JDBC to interface with databases, Sqoop imports the contents of tables into a Hadoop Distributed File System (HDFS) and generates Java classes that enable users to interpret the table's schema. Sqoop can also export records from HDFS to a relational database.
- ***Hue*** — A graphical user interface to work with CDH. Hue aggregates several applications which are collected into a desktop-like environment and delivered as a Web application that requires no client installation by individual users.
- ***Pig*** — Enables you to analyze large amounts of data using Pig's query language called Pig Latin. Pig Latin queries run in a distributed way on a Hadoop cluster.
- ***Hive*** — A powerful data warehousing application built on top of Hadoop which enables you to access your data using Hive QL, a language that is similar to SQL.
- ***HBase*** — provides large-scale tabular storage for Hadoop using the Hadoop Distributed File System (HDFS). Cloudera recommends installing HBase in a standalone mode before you try to run it on a whole cluster.
- ***Zookeeper*** — A highly reliable and available service that provides coordination between distributed processes.
- ***Snappy*** — A compression/decompression library. You do not need to install Snappy if you are already using the native library, but you do need to configure it.

3.2.1 System and Hardware Architecture



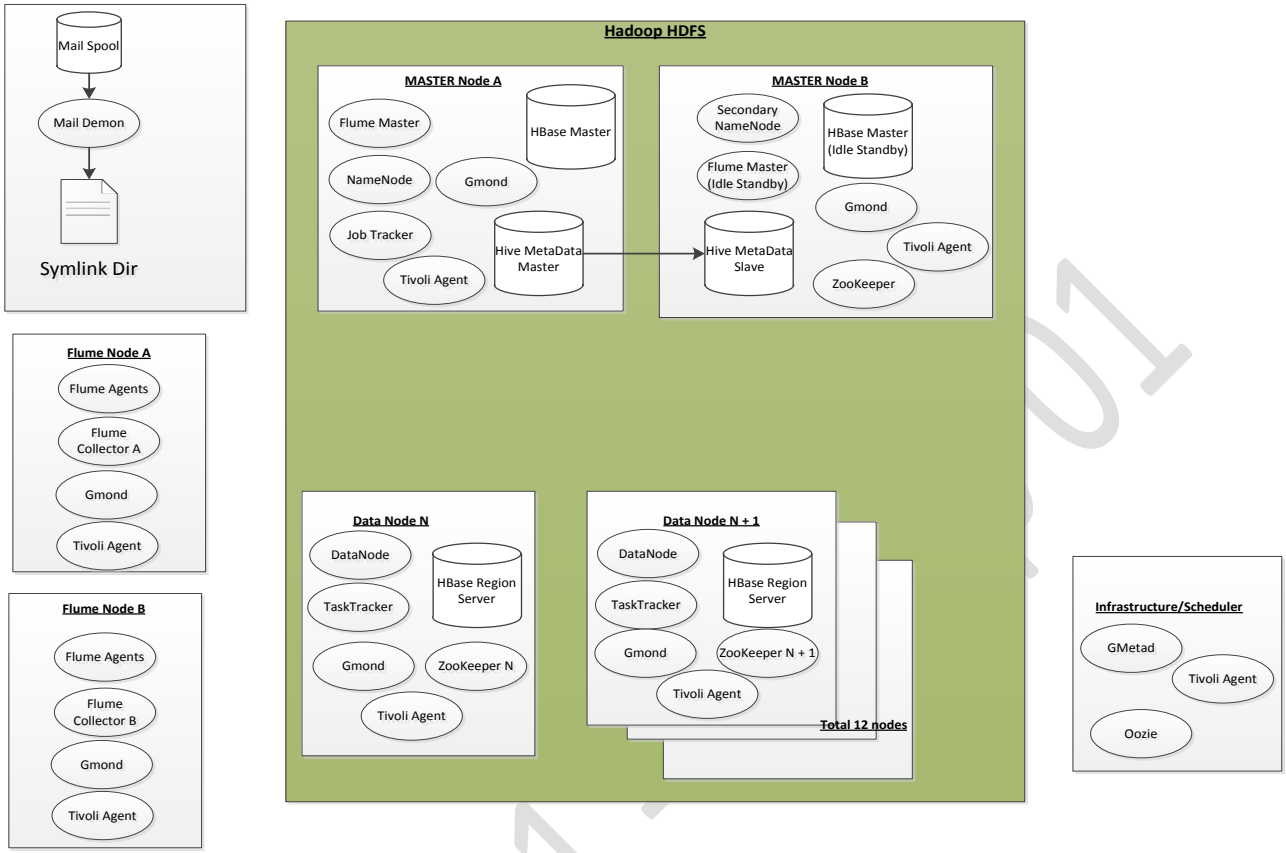
The hardware components illustrated in the diagram above are described in the tables below:

Master Nodes	Data Nodes	Flume Nodes / Scheduling Monitoring Node
Class A machines (RX200)	Class B machines (RX300)	Class B machines (RX300)
2 Sockets x 6 core CPU	2 Sockets x 6 core CPU	2 Sockets x 6 core CPU
72GB RAM	48GB RAM	48GB RAM
Dual 10 Gigabit NICs	Dual 10 Gigabit NICs	Dual 10 Gigabit NICs
Dual Power	Dual Power	Dual Power
OS: RHEL5.6	OS: RHEL5.6	OS: RHEL5.6
Cloudera distribution CHD3u1	Cloudera distribution CHD3u1	Cloudera distribution CHD3u1
File System: ext3	File System: ext3	File System: ext3
Storage: FAS2040 HA	Storage: E2600	Storage: Local Disks (2TB+ per machine)

Networking for the Hadoop Cluster:

- Top of Rack 10Gigabit Switches
- 10 Gigabit Interconnects
- 2 layer network configuration for Production and Stage with 2 rack configuration

Software Architecture for Hadoop Cluster



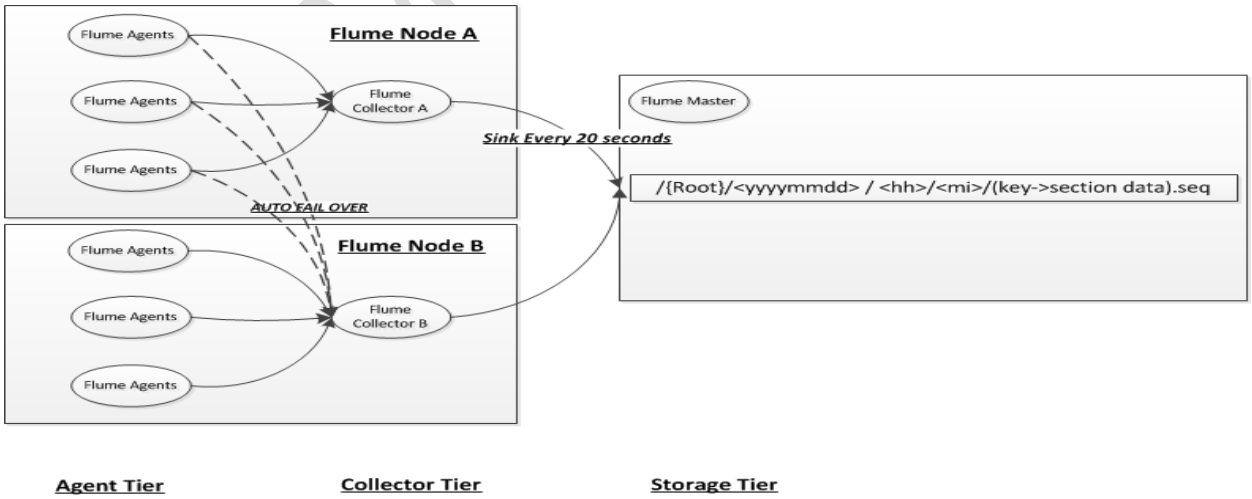
The architecture for software components include:

- **NameNode** -- The 'coordinator' process for the HDFS cluster. Serves as the primary meta-store where clients can find the list of data nodes that contain a given HDFS block.
- **Secondary NameNode** -- Keeps a 'checkpoint' log of the NameNode metadata, so that in the event of complete NameNode failure, a new NameNode can be restored. Note that this is not automatic (despite the implication of the name 'Secondary NameNode'). The process for creating a new NameNode from this metadata is a manual one.
- **Data Node** -- The individual HDFS data server processes, which are responsible for the blocks stored on each node. Clients communicate directly with each DataNode once they have been 'pointed there' by the NameNode.
- **Job Tracker** -- The 'coordinator' process for MapReduce processing. Responsible for organizing MapReduce jobs run on the cluster.
- **Task Tracker** -- The individual task processes for MapReduce processing. They communicate with the JobTracker to run individual map and reduce tasks, and shuffle data around the cluster.
- **HBase Master** -- The 'coordinator' process for HBase. Maintains the mapping of all table key spaces and the regions servers that serve them.
- **HBase Region Server** -- The individual worker processes for HBase. Responsible for serving and storing data for tables in addition to store file management including splits and compactions.

- **Zookeeper** -- Zookeeper is a distributed lock mechanism used by HBase and Flume. Zookeeper uses a quorum of Zookeepers to coordinate the granting of locks. Zookeepers are also responsible during automatic failover process for electing new Flume and HBase masters in the event that an active master goes down.
- **Hive Metastore (MySQL)** -- Hive uses an RDBMS for its Metastore, which stores relational information about the 'tables' that Hive knows about in the cluster. We use MySQL for this Metastore in a high availability configuration of master / slave configuration.
- **Flume Master** – The Flume masters are responsible for managing Flume flow configurations an Flume Node management.
- **Flume Agent** – Flume Agents are Flume nodes that play the role of data sources. In this architecture, Flume Agents are responsible for detecting new LOGs and initiating ingestion processing.
- **Flume Collector** – Flume Collectors are Flume nodes that receive Flume events from Flume Agents. Flume Collectors deliver events to Sinks. In this architecture Flume Collectors are configured with Sinks that write LOGS data into HDFS.
- **Ganglia** -- Collects and displays monitoring data on all machines in the cluster. The Infrastructure / Scheduler node runs the main 'metastore', while all other nodes run a 'monitor' process that feeds data to the metastore. These monitor processes collect both low-level system data, as well as higher-level Hadoop/HDFS statistics.

Below is the detail design that will be implemented in the flume decorator:

3.2.2 Ingestion Design



We will be deploying 2 Flume nodes for high availability design. Flume agents/collectors will be configured so that if a machine or process is down, a secondary Flume node's agents/collectors can continue to process incoming LOGs.

Flume DirSource / FileSource

The Flume sources for LOGS Phase 0 are based on the Flume TailDirSource and TailSource source implementations, but area adapted to meet the unique needs for LOGS.

DirSource

The DirSource is configured source for the Flume Agent. Its implementation must:

- Monitor a directory for new and deleted files
- Create the FileSource container
- Notify the FileSource that a new file is to be processed when a new file in the directory being monitored is detected.
- When a New File is detected write a NEW log record to the LOGS Lifecycle Log
- Remove files from the processing queue when they are deleted.
- Maintain the buffer of concurrent files to process. That is, if the FileSource is already processing a configurable number of files, then the DirSource is responsible for caching any new file paths detected and releasing them to the FileSource when the FileSource has available capacity.

FileSource

The FileSource is managed by the DirSource. FileSource is mainly responsible for sectioning LOGs into events and sending them to collectors for further processing. The life cycle in the FileSource for processing an incoming LOG is the following:

Step 1: Using the LOGS file name, determine the format of the LOGS and instantiate the appropriate Cursor implementation to handle the file format. Possible formats are:

a) 7-Zip (.7z)
    a. Parsed using net.sf.sevenzipjbinding.SevenZip library. The zip file is streamed and extracted sequentially.
b) GZip Tarball (.tgz)
    a. Parsed using java.util.zip.GZIPInputStream and org.apache.tools.tar.TarInputStream. Compressed file and attachments are streamed sequentially.
c) Single Header File
    a. Parsed using java.nio.channels.FileChannel
d) SMTP Mime with file attachments
    a. Parsed using java.nio.channels.FileChannel. Sections and attachments are read and processed sequentially.

Step 2: Mark the current time. This will be the Event timestamp for all events for this section.
Step 3: Parse the LOGS_ID from the filename.
Step 4: Create a timestamp that is the Event timestamp plus 30 seconds. Convert this timestamp to a String formatted as YYYY-MM-DD/HH/MM and set the "localTime" attribute on the event. This value will be used for output bucketing when the event gets written to HDFS.
Step 5: Initialize a Sequence counter to 1. To be used to count the sections.
Step 6: Parse each section in the LOGS
Step 7: For each section

- Create a new Flume Event with the Event timestamp from Step 2.
- Set the event body to the raw LOGS section.
- Set the " logs_id" attribute of the event to the value parsed in Step 3.
- Set the "localTime" attribute to the value calculated in Step 4.
- Set the "sectionSequence" attribute to the value of the Sequence counter.
- Set the "sectionName" attribute to the name of the section.
- If the section is the last section for the LOGS, set the "lastSectionFlag" attribute on the event to "true".
- Append the event to the Flume processing queue
- If it is not the last section of the event, increment the sequence counter and repeat Step 6.

Step 8: Once all sections are successfully processed for LOGS, close the file cursor and notify sweeper daemon so that it can clear the LOGS from the source location

3.2.3 Flume Sink

There will be at least one Flume Sinks configured for the Phase 0 flow. One for all LOG sections and one for the LOG file sections. Both Flume Sinks will be configured as "escapedCustomDfs" sinks, the difference being the configured output path in HDFS and the output format.

Phase0 will require two custom output formats. Both output formats will write Hadoop SequenceFiles with LZO compressed values. BLOCK or RECORD compression is TBD based on testing.

Text-Sequence-File-Output-Format

The Text-Sequence-File-Output-Format implements Flume's OutputFormat whose format() method is responsible for writing individual events to the sink's output stream. For Text-Sequence-File-Output-Format the format method will:

1. Create a SequenceFile.Writer for the output stream if one does not exist.
    a. The Writer should be created using the LzoCodec.class and either BLOCK or RECORD compression (TBD)
2. Mark the current position of the output stream using its getLength() method.
3. Determine the name of the file being written (actual file names and locations are TBD)
4. Set the "filename" attribute of the Event to the name of the file being written.
5. Set the "offset" attribute of the Event to the current position determined in step 2.
6. Convert the Event to AVRO JSON using the org.apache.avro libraries.
7. Append the AVRO JSON formatted event to the SequenceFile.Writer using Text key and Value. The key is the value of the "logs_id" event attribute concatenated with "_" and the value of the "sectionName" event attribute. The value is the AVRO JSON formatted event.
8. Write an INGESTED log record to the LOGS Lifecycle Log if the "lastSectionFlag" attribute on the event is true

Note: The Text-Sequence-File-Output-Format will be configured to bucket its output based on the value of the "localTime" attribute generated when the LOGS was first steamed into Flume. The format of the output directory will be "YYYY-MM-DD/HH/MM ". The intent of this bucketing of LOGs into a date partition slightly in the future is to facilitate configuration and loading of complete LOGs, so using a date in the future allows for some lag time for ingesting LOGS sections. Without the use of bucketing, the probability of processing partial LOGs is faily high and while the design accounts for handling partial LOGs it is desirable to limit partial LOGS processing as much as possible.

Log-Binary-Sequence-File-Output-Format

The Log-Binary-Sequence-File-Output-Format implements Flume's OutputFormat whose format() method is responsible for writing individual events to the sink's output stream. For Log-Binary-Sequence-File-Output-Format the format method will:

1. Test the event's "sectionName" attribute and if it is not an LOG_FILE section then ignore the event.
2. If the event is an LOG_FILE section test the event's "partialSection" attribute and if the value is true then ignore the event.

3. Create a SequenceFile.Writer for the output stream if one does not exist.
    a. The Writer should be created using the LzoCodec.class and either BLOCK or RECORD compression (TBD)
4. Extract the body from the event.
5. Decode the event body using org.apache.commons.codec.binary.Base64.
6. Append the value of the "logs_id" event attribute as a Text key and the decoded bytes as an Bytes Writable value to the SequenceFile.Writer.

3.2.4 ConfigLoader MapReduce Design

The ConfigLoader job is responsible for taking LOGS sections copied into the cluster by Flume. It parses the sections to extract necessary configuration elements and then loads the relevant configuration data into the HBase LOGS_META and LOGS_CURRENT tables.

The ConfigLoader is a Hadoop Streaming job because the Parsers are leveraging pre-existing Perl libraries for processing LOGS sections. As such there is no Java based Job or Configuration object for the mapper in this MapReduce job.

Job Arguments

The arguments for the ConfigLoader job are:

- Input Path – The location in HDFS of the files to be processed (passed by Pentaho Scheduler)
- Output Path – The location of the Job's output files (passed by Pentaho Scheduler)
- Input Format –Sequence-File-As-Text-Input-Format-With-Split-Info
- Output Format – Sequence-File-Output-Format
- Mapper – Parse.pl
- Reducer – Config-Loader-Reducer
- Replay – Company YYYYY.loader.replay=<true|false> -- Inidcates to the loader whether or not this is a "replay" job and if it is, then do not overwrite the last logs id processed.

Additional configuration variables passed into the command line include:

- Lifecycle Log Path – A path to a flat file storing standardized log records for tracking the lifecycle of an LOGS.
- Error Log Path – A path to a flat file where detailed error & debug info can be written.
- Number of failed tasks allowed – (mapred.map.max.attempts)) A parameter passed to Hadoop which specifies the number of tasks which need to fail before the whole job fails.
- The number of skipped records allowed (mapred.skip.map.max.skip.records and mapred.skip.map.auto.incr.proc.count) Specifies the number of allowable BAD records to skip and still consider a map task successful.

Sequence-File-As-Text-Input-Format-With-Split-Info

This input format will extend the standard Sequence-File-As-Text-Input-Format class. In particular, it should override the "next" method so that bad records are logged and not sent to the mapper. This functionality will be accomplished with a while loop that repeatedly calls the underlying "next" method until an acceptable record is returned.

The input format should also override the method that calculates the incoming splits so that the mapping between these splits and mapper tasks can be logged. Logging this information will allow us to determine the specific input split that caused a map task to fail. Or if the task doesn't fail, this information may be useful for tracking individual sections which may be corrupted or malformed.

Parser.pl

The Parser.pl script is the Mapper for the ConfigLoader job. It is responsible for:

1. Extracting the LOGS_ID and the Section by parsing the input key. All emitted values will have the LOGS_ID as the key.
2. For all sections extract the "fileName" and "offset" from the AVRO JSON formatted input value and emit the section name, fileName and offset.
3. If the section is one of: HEADERS, X-HEADERS, SYSCONG-A, SYSTEM_SERIAL_NO, CLUSTER_INFO, CLUSTER_MONITOR then invoke the specific parser for that section.
    a. The values that need to be extracted from these sections are the following (a separate design note will be generated for the specific of each section parser)
        i. cfo_node_state
        ii. system_id
        iii. sys_serial_no
        iv. partner_system_id
        v. partner_serial_no
        vi. logs_gen_date
        vii. logs_gen_zone
        viii. logs_type
        ix. cluster_id
        x. system_version
        xi. system_model
    b. Each parser will emit all parsed fields keyed on LOGS_ID. The Loaders will be responsible for determining which fields to use during the loading stage. Emitted values will be similar to the following format:

```
"SYSCONFIG_A_SYSTEM":{
"sys_bp_partno":null,
"system_id":"0101174535",
"sys_bp_rev":null,
"p0_system_id":"0101174161",
"is_clustered":"T",
"sys_serialno":"1071112",
"hostname":"rhv-entfs-001",
```

```
"sys_version_pro_tz":"PST",
"sys_bp_serial_no":null,
"sys_version_pro_sec":"1263299807",
...
}
```

Throughout the process of parsing sections, the Perl script will log detailed error and debug information to the error log. In addition, the script should use the path to the lifecycle log to record the result of parsing each section. See the lifecycle management section for more details.

Config-Loader-Reducer

The Config-Loader-Reducer's primary role is to assemble the LOGS's business key and configuration values before writing this data out to HBase. The reduce method flow is:

1. Using the values passed into the reduce method calculate the business key
2. Write the following values to the LOGS_META HBase table's SYSTEM column family using LOGS_ID as the RowID with the following column qualifiers:
    a. business_key
    b. cluster_id
    c. cluster_member_id
    d. system_id
    e. serial_number
    f. system_version
    g. system_model
    h. partner_system_id
    i. partner_serial_no
    j. logs_type
    k. logs_gen_date
    l. logs_received_date
    m. time_zone
    n. hostname
    o. partner_hostname
3. Write the following values to the LOGS_META_CURRENT HBase table's META column family using LOGS_ID as the RowID the following column qualifiers:
    a. Section_name
    b. NumberOfParts
    c. Parts
    d. Position
4. Write the LOGS_ID into the "last_ logs_id" column in the LOGS_META_CURRENT table's SYSTEM column family using the business key as the RowID. Only update the "Last logs id" if the business key already exists otherwise create new records with business key and "last_ logs_id"
5. For all sections extract the section-Sequence and last-Section-Flag.

a. If the sorted set of sequence numbers can be incremented with no missing sequences and the highest sequence number also has a corresponding last-Section-Flag value equal to true
    i. Write a LOADED_ALL log record to the LOGS Lifecycle Log.
b. If the list of sections is incomplete for any reason (i.e. there is a gap in the sequence numbers or the last-Section-Flag is false), but all of the required sections were found:
    i. Write a LOADED log record to the LOGS Lifecycle Log.
    ii. Emit a record for each value passed into the reduce method. For each value, emit the value as is and the key associated with that value (i.e. the input key to the reduce task under which this value was grouped). Together this key-value pair will form a single record.
c. If any of the required sections were missing:
    i. Write a LOAD_FAILED log record to the LOGS Lifecycle Log.
    ii. Emit a record for each value passed into the reduce method. For each value, emit the value as is and the key associated with that value (i.e. the input key to the reduce task under which this value was grouped). Together this key-value pair will form a single record.

3.2.5 Scheduler

Pentaho Scheduler will be used to configure and schedule the ConfigLoader Jobs. If the ConfigLoader job fails for any reason, create a new file under a directory path that uniquely identifies the job and check the status of the previous two Config-Loader workflows. If the two previous jobs also failed, send an email alert.

Job One – Main Flow

The following action steps will be required to execute a run, which initially will be configured to run once every 1 minute

Step 1: Execute the Config-Loader shell script passing as arguments:

1. The input path which is the HDFS Location of Flume ingested raw sections. The formatted date value for this directory should be calculated as NOW – 6 minute
2. The output path, a temporary output location.

Step 2: Using the output path from Step 1, copy the output files to a directory whose formatted date value is calculated as NOW + 1 hour. The directory paths will be at the hour granularity. Note that this step is intended to handle the case where incomplete LOGs are emitted from the primary ConfigLoader flow. The output files are copied to a common location that is used as input into the Alternate Flow, Job Two described below.

Step 3: Delete the output path from Step 1.

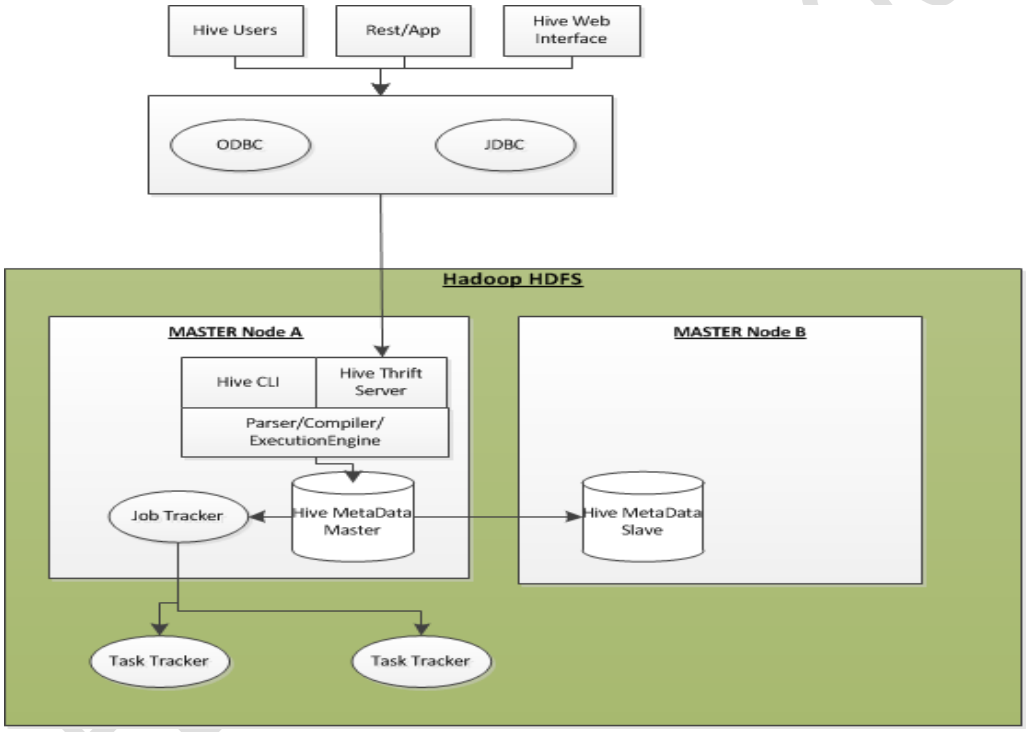
Job Two – Alternate Flow

The following action steps will be required to execute a run, which initially will be configured to run once per hour:

Step 1: Execute the Secondary Config Loader job passing as arguments:

1. The input path of previous incomplete LOGs. The formatted date value for this directory should be calculated as NOW – 1 hour
2. The output path, a temporary output location.
3. Step 2: Using the output path from Step 1, copy the output files to a directory whose formatted date value is calculated as NOW + 1 hour. The directory paths will be at the hour granularity
4. Step 3: Delete the output path from Step 1.

3.2.6 Hive Software Design



We will be setting up a remote Metastore for Hive. All Hive clients will make a connection to the Metastore server which in turn queries the relational database (MySQL) for metadata. The Metastore server and client communicate using the Thrift Protocol. To start the Hive Thrift server execute the flowing command: hive --service metastore

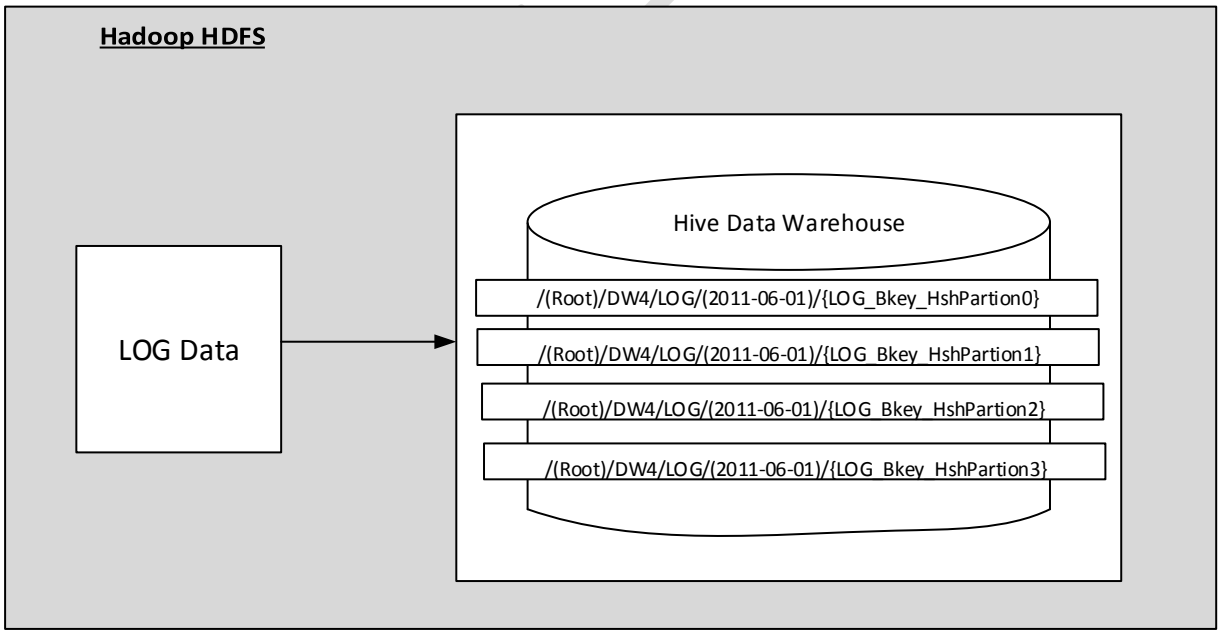
3.2.7 Hive Schema Design

log_data Table

The log_data table is the main Hive table for all parsed and loaded LOG files.

```
CREATE EXTERNAL TABLE log_data
        (system_key STRING,
         logs_id STRING,
        date_time STRING
        sys_model STRING,
        sys_version STRING,
        sequence_id STRING,
        message STRING,
        tags MAP<STRING,STRING>
        )
PARTITIONED BY (dt STRING, sys_hash STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' COLLECTION ITEMS TERMINATED BY ':'
MAP KEYS TERMINATED BY '='
STORED AS INPUTFORMAT "com.hadoop.mapred.DeprecatedLzoTextInputFormat"
OUTPUTFORMAT "org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat";
```

The following diagram illustrates how the partitions look in terms of the directory structure.



log_event_map Table

The log_event_map table is used to join log_data to determine the object type the event is for, mainly to be used for the daily LOG event CSV export to legacy DW.

CREATE TABLE log_event_map (object_type STRING, event_name STRING)

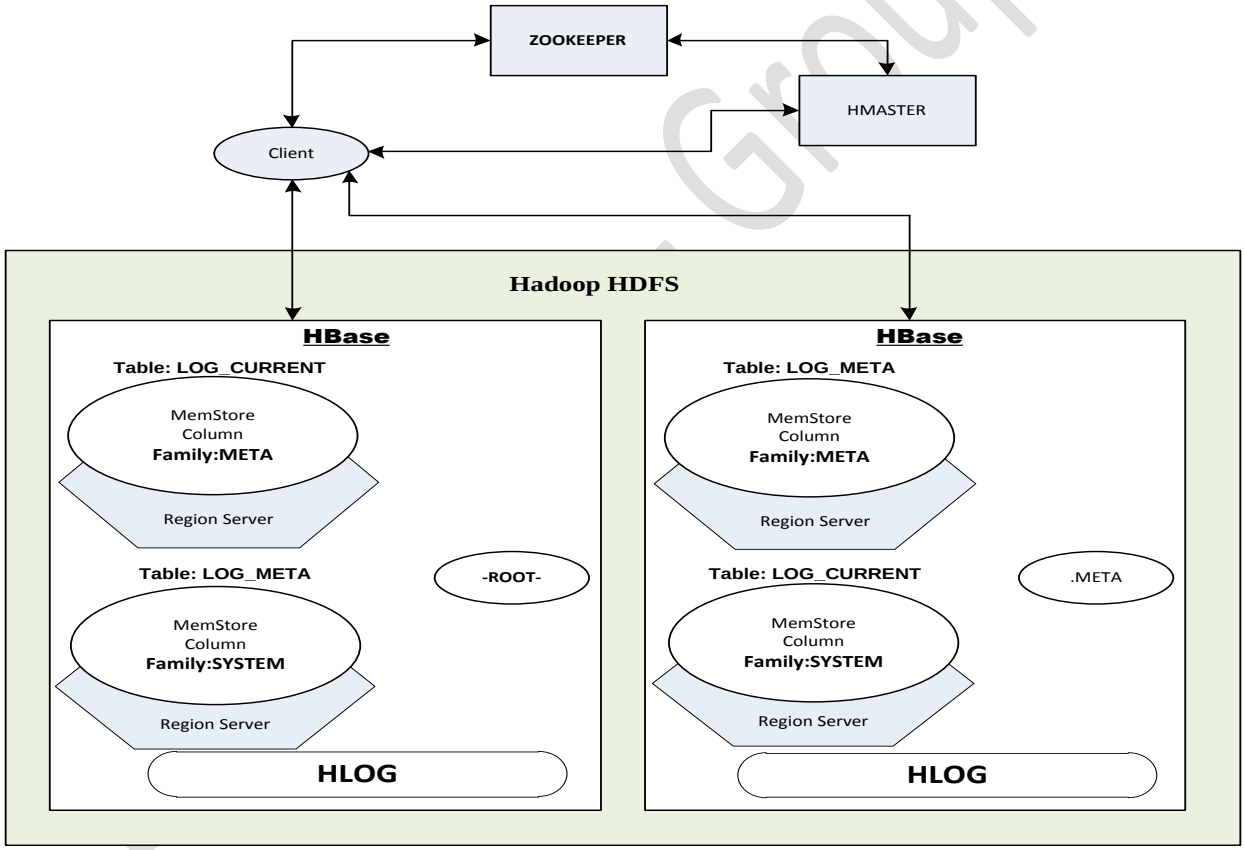
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';

Business Key Hash UDF

A Hive user defined function will need to be developed and installed in order to facilitate generation of the business key hash. See the LOG Design section's LOGKeyHashPartitioner for a description of the hashing logic for the business key.

3.2.8 HBase Software Design

The following illustrates the HBase software component design.



Key points to note about the HBase software components are:

- A ZooKeeper cluster maintains a quorum to elect the HBase Master. In the event of an HBase Master failure, the ZooKeeper will elect the standby HBase Master to take over automatically and with minimal interruption of service.
- All client access to HBase is coordinated via the ZooKeepers for initial connectivity and resolution of the HBase Master.
- There will be one HBase Region Server per node. Each Region Server manages one or more regions for HBase tables. In the event of a Region Server failure, other Region Servers will eventually take over serving the regions originally managed by the "failed" Region.

3.2.9 HBase Schema Design

For Phase 0, the HBase schema is simple, consisting of two tables i:e LOGS_META and LOGS_CURRENT. The following two tables describe the schema for each of the HBase tables.

LOGS_META

RowID	< LOGS_ID>
Column Family Name	SYSTEM
Key Column Family Initial Settings	BLOOMFILTER => 'ROW' REPLICATION_SCOPE => '0' COMPRESSION => 'LZO' VERSIONS => '10' TTL => '2147483647' BLOCKSIZE => '65536' IN_MEMORY => 'false' BLOCKCACHE => 'true'
Column Qualifiers and Types	business_key : String cluster_id : String cluster_member_id: String system_id : String serial_number : String system_version : String system_model : String partner_system_id : String partner_serial_no : String logs_type logs_gen_date logs_received_date time_zone hostname partner_hostname

RowID	< LOGS_ID>
Column Family Name	META
Key Column Family Initial Settings	BLOOMFILTER => 'ROW' REPLICATION_SCOPE => '0' COMPRESSION => 'LZO' VERSIONS => '10' TTL => '2147483647' BLOCKSIZE => '65536' IN_MEMORY => 'false' BLOCKCACHE => 'true'
Column Qualifiers and Types	<section_name> : String (e.g: {"format" : "<GZIP|BASE64|TEXT>", "numberOfParts": 10, "parts" : [{ 1 : { "filename":"/path/to/file/name.seq" , "position" : 12345} , 2 : { "filename":"/path/to/file/name2.seq", "position" : 17890}] }) – Note this format support the eventuality that sections themselves may be split during ingestion.

LOGS_CURRENT

RowID	< LOGS_BIZ_KEY> (e.g. <cluster_id>_ <system_id>_ <serial_number>
Column Family Name	SYSTEM
Key Column Family Initial Settings	BLOOMFILTER => 'ROW' REPLICATION_SCOPE => '0' COMPRESSION => 'LZO' VERSIONS => '10' TTL => '2147483647' BLOCKSIZE => '65536' IN_MEMORY => 'false' BLOCKCACHE => 'true'
Column Qualifiers and Types	last_ logs_id : String

N**ote**: The current plan for more detailed HBase schema design in future phases is to leverage AVRO as the storage format instead of JSON for columns that will contain collections of data. One driving factor for this is that the AVRO structures take about half as much storage as the JSON formats (determined during proof of concept testing). Also AVRO is emerging as a Hadoop friendly data serialization format. Additional future phase design activities around AVRO support include:

- Design of User Defined Functions for reading HBase AVRO data from within Hive and Pig.
- Additional small proofs of concept to ensure the ability to serialize/deserialize AVRO data meets all of the functional requirements for LOGS.
- Follow-up discussions to assess the AVRO roadmap for Cloudera and Apache Hadoop

3.2.10 Compression

We will be using LZO Compression throughout the Hadoop Ecosystem. LZO allows fast compression and decompression and significantly reduced the amount of disk storage required for storing log's in the cluster.

3.3 Data Loading process to DW/DM

The above process defines the data being extracted and stored into HDFS. Below section will define the loading process from HDFS/HBASE to Oracle DSS system. This is purely an ETL process, which will make use of ETL cluster. The ETL tool will be Pentaho Data Integrator (PDI).

With the time constraint and high volume, the need of the design is to finish the complete load in time without missing the SLA. In big data volume and time constraint scenario, following four things play a big role.

1. Restartability
2. Delta capture
3. Multi-threading
4. Clustering

3.3.1 Restartability-

Restartability is defined as starting the process again from the point of failure. This can also be extended to incremental load that is, start of the process from the point of success. Restartability is achieved thru by implementing check points at different stages of data flow from the source to the target.

3.3.2 Delta capture (CDC) –

This is basically finding the changes that have taken places in the source side and to find those changes and load to the target. To find these changes in the source side, the records that have gone thru changes, need to be identified in the source side and marked as change records thru a flag or thru a date time stamp i:e the last updated timestamp.

3.3.3 Multi-Threading –

This approach is followed to maximize the utilization of the available resources. This is achieved by increasing the number of processing threads i:e parallelism (component, pipe and data).

3.3.4 Clustering –

In clustering we make use of commodity hardware, desktops, to distribute the load to have a better performance and load balancing. We can keep adding nodes to the existing cluster in case of high volume without putting much time on development.

Incremental load / restartability –

This will take care of the restartability as well as incremental load.

The HBASE/HDFS read component of the Pentaho ETL tool provides the mechanism to start the reading of the source (scan) from a particular row key and end at a particular row key to avoid the full scan of the source.

The design is based on the utilization of start key and end key. From the point of view of restartability or incremental load these two fields have to be populated before the load. These two fields will be provided from a parameter file.

Once the key value is numeric value or date-time stored as a numeric value as key, then it becomes easier to start from the point of last load whether a successful load or an unsuccessful load.

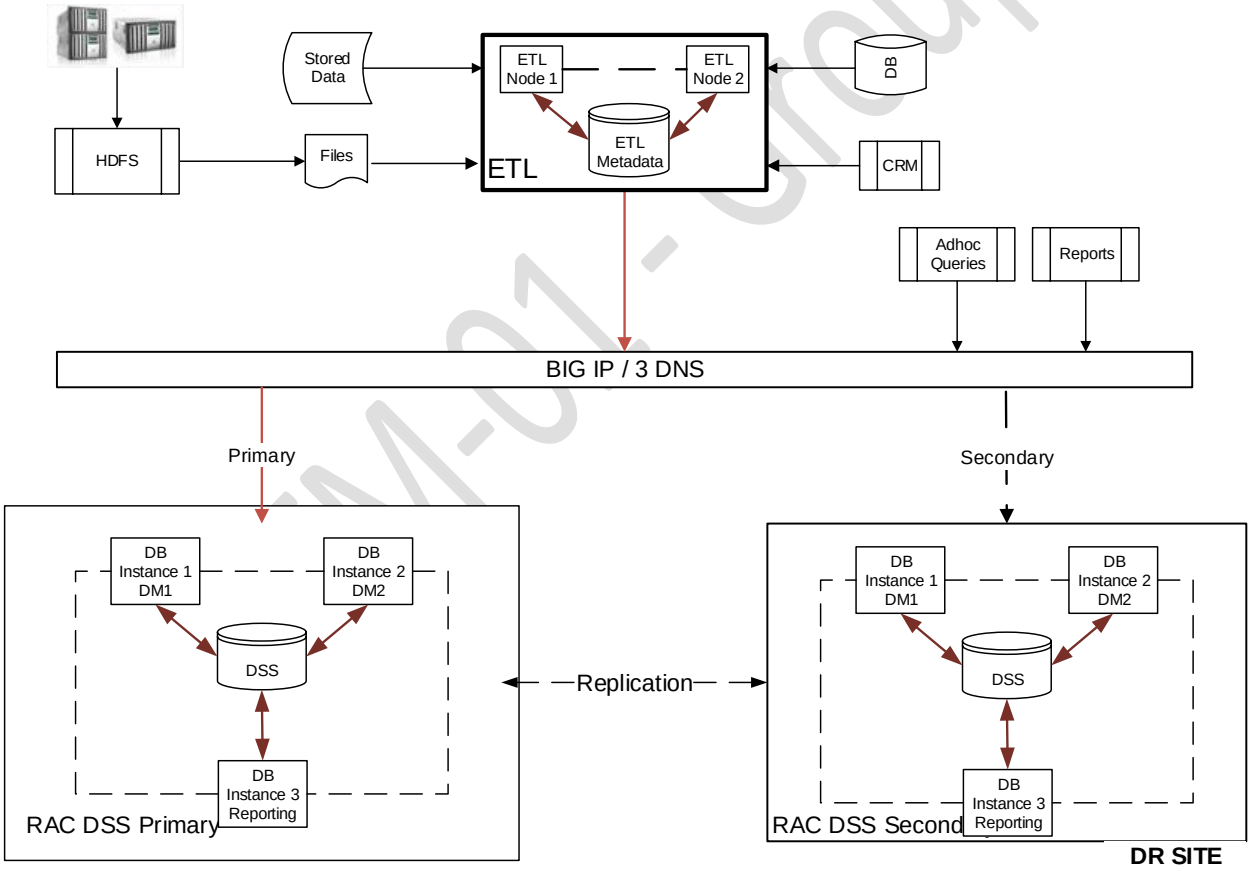
The idea is to capture the last key value of the last load in a parameter file and utilize the same to determine the start key value for the next load.

Parameter file/Table

The parameter file / table will contain the following information

1. Name of the object
2. Last transferred key to the target
3. Number of lines/records to fetch – This will help in defining the actual load
4. Date and time when this was updated

Here is the overall ETL Architecture for data load from different sources including Hadoop.

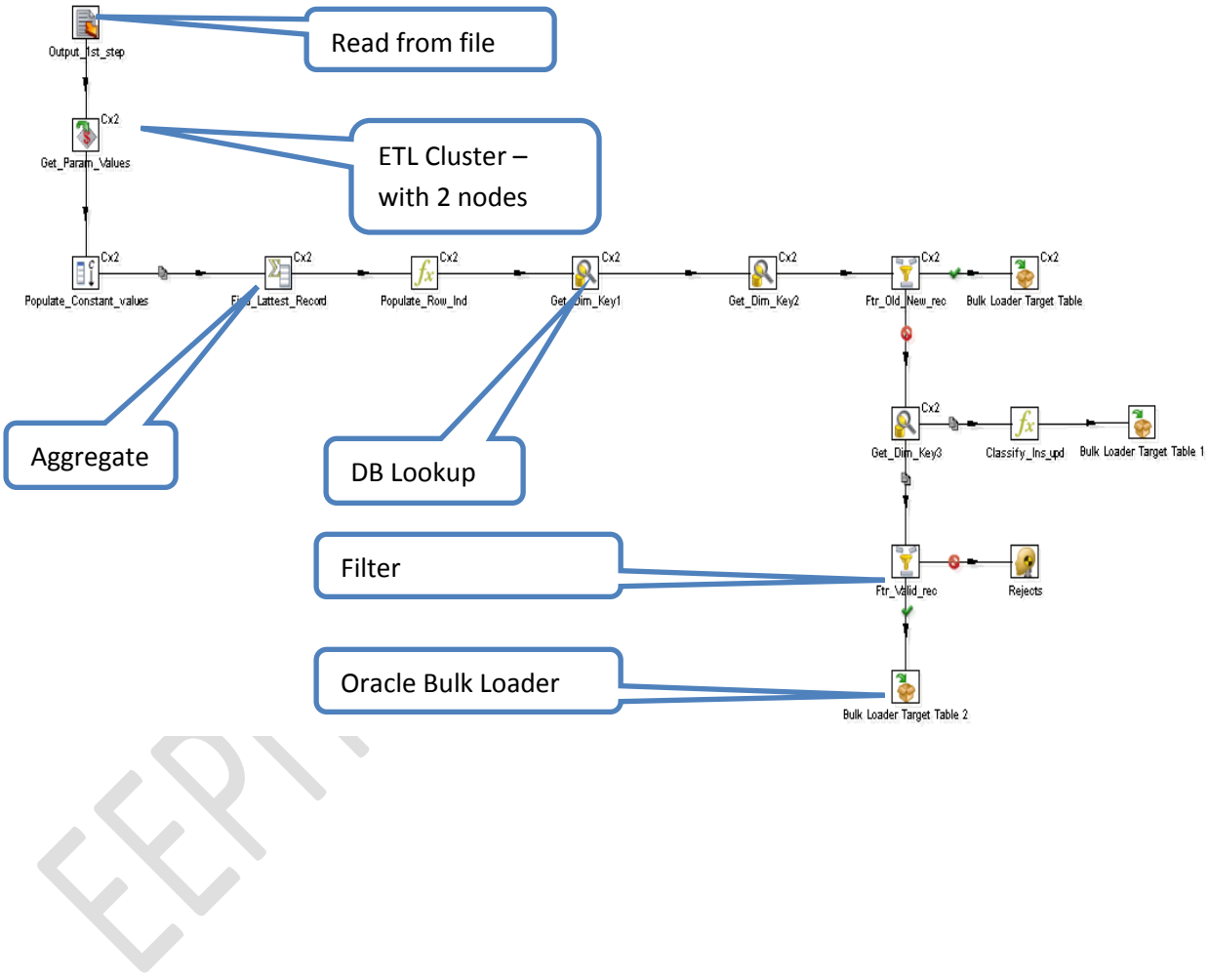


Availability / Scalability / Load Balancing

The hardware components illustrated in the diagram above are described in the tables below for ETL server cluster:

Master Nodes	Slave Node 1	Slave Node 2
Class A machines (RX200)	Class B machines (RX300)	Class B machines (RX300)
2 Sockets x 8 core CPU	2 Sockets x 8 core CPU	2 Sockets x 8 core CPU
96 GB RAM	96 GB RAM	96 GB RAM
OS: RHEL5.6	OS: RHEL5.6	OS: RHEL5.6

Here is an example of data flow with the use of ETL cluster, where the source is file and target is RDBMS…

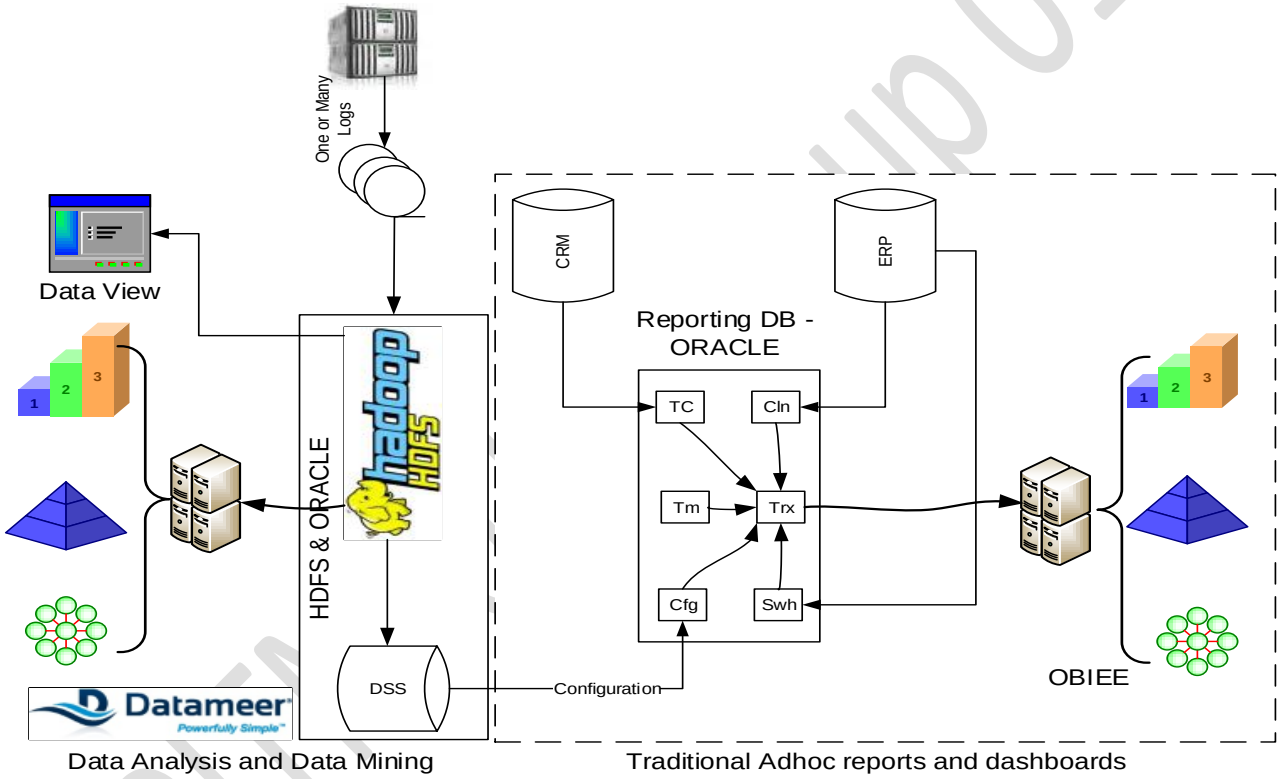


4 OLAP & Data Mining Capabilities

4.1 Next step – Over all data consumption – Reporting and Analysis

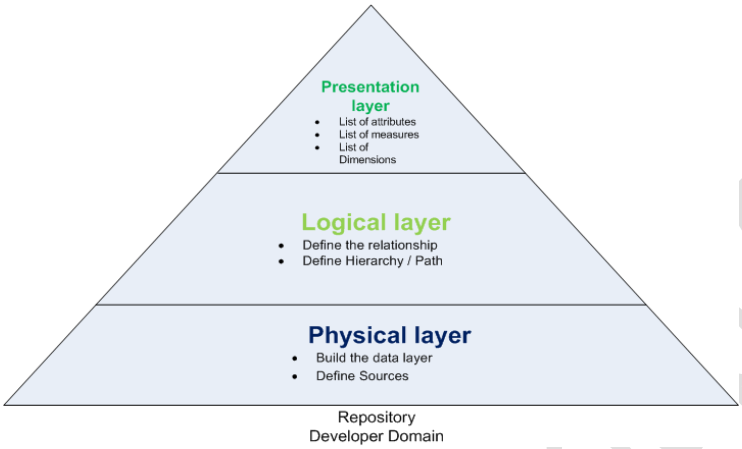
Now the data is available at HDFS as well as different Datamarts and DSS for consumption. The traditional reporting relating to structured data will be executed by OBIEE and the ad-hoc reporting, analysis and mining happening on the big volume of data and those are time consuming can happen on the Hadoop cluster by marrying the semi-structured data with structured data with the help Datameer.

Below is the architectural diagram for the reporting side….



4.1.1 OBIEE – Oracle Business Intelligence Enterprise Edition

In OBIEE, there are 3 different layers that need to be build for the user to use the tool and the definition of those layers defines the repository which the user will use to do the slicing and dicing.



Physical layer – In this layer, mapping to source is defined

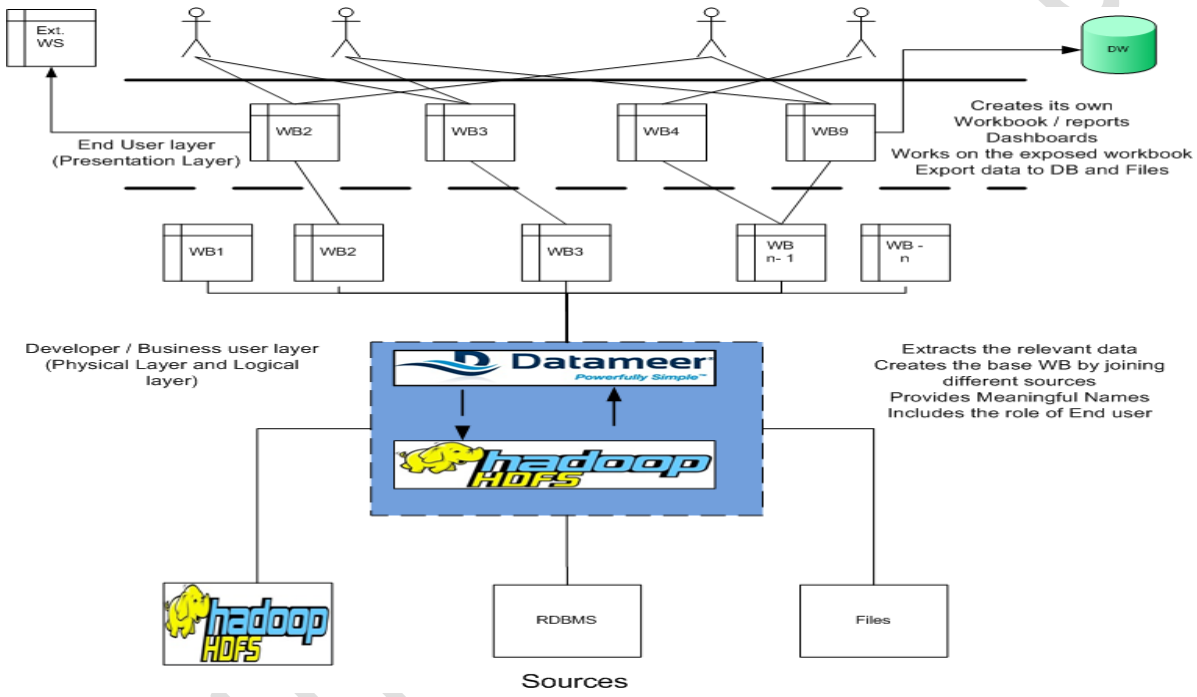
Logical layer – In this layer, defining the relationships, hierarchy which will help in drill down, drill across type of reporting

Presentation layer – In this layer the user is exposed to dimensions, attributes, facts, measures. Repository is developed by the developer. This layer hides the complex data structures from the end user as the end user sees the elements that are exposed in the presentation layer to be used in reporting. All the drill down and drill across and reporting is limited to what is exposed in presentation layer.

4.1.2 DATAMEER

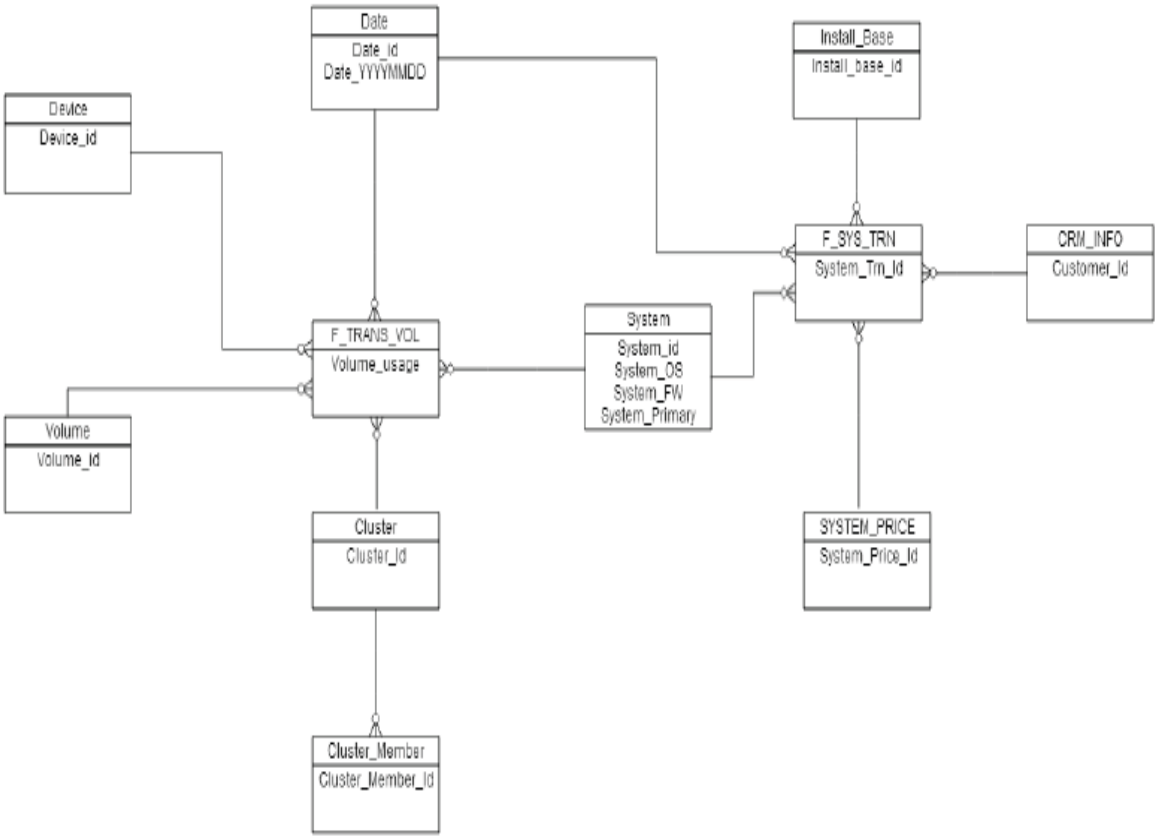
Datameer tool will be used extracting data from different sources to HADOOP cluster, where a developer can develop the data extraction, creates the base workbook like MS-XLS, basically creates a security and abstraction layer for the end user. Based on the requirement, developer publishes a list of workbook for the users, there by hiding the complexity of data structures from the end users. Business user can directly work on the dataset available in HDFS. End users when log in, are directed to the workbook , that is assigned based on the security and roles defined in the tool, based on the definition of R&R, the user can export data to XLS WS and to database like Oracle

Here is an example where multiple users can do data analysis and data mining using Datameer by combining structured and semi-structured data.



4.2 Data Model/Reporting Model

Here is a snapshot of a STAR/SNOWFLAKE schema that will be build in different layers of OBIEE for the reporting to be done in the tool at the same time this will help to understand the different relationship to do the analysis and mining in Datameer. Here there are 2 facts table and multiple dimension tables and these tables are used to calculate the volume usage by the client and at the same time, will provide what type of systems have been installed with what firm-ware, OS at different clients that is providing the maximum benefit. It will also provide how many upgrades as system for a client have gone thru over a period of time.



The tools OBIEE/Datameer provide functionality to do the customer segmentation based on the business need and for us; it is needed to segment based on the volume usage and work load.

What system is doing well with what combination of OS , Firmware and how many customer do have that type of system can very well be found from the logs along with other data sets taken from CRM and putting them in HDFS thru Datameer to do the data mining as well as analysis.

4.3 Decision Making-Data Mining

The new architecture provided by the Project team helps in instant decision making and continuous analysis from the data collected in the run time.

This has subsequent advantages over the traditional BI based decisional making systems.

The average lag between the data creation and the report generation is around 24 hours.

For businesses which depend on the BI for various operations like customer retention, offer generation to name a few, the freshness of the data in the report plays a crucial role.

We will be highlighting in our model on how the new architecture proposed can help in offer generation on the run.

Attached below is the statistics collected on run time in an Active Data ware house:

Software Configuration	Configuration Frequency	Counts from logs	Customer Base of OS
Windows 98 OS + 1GB RAM	weekly	10M	100M
Windows 98 OS + 4 GB RAM	Fortnight	15M	80M
XP + 1 GB RAM	Monthly	30M	500M
XP + 4 GB RAM	Fortnight	30M	300M
Windows 7 + 4GB RAM	Half Yearly	10M	150M
Linux + 4 GB RAM	Yearly	5M	50M

Attached below is some of the offers which can be thought off in order to increase the customer base, customer retention and provide lucrative offers to the new customer segment:

Offers generated

New users:

1) Tie up with Windows 7 OS user to avail 20% off on using our product X Antivirus

2) Linux announces 30% off on users of X-antivirus

Existing users:

1)Windows 98 users to get 40% off on the X antivirus

2) Window XP -1 GB users to get 50% off

5 Conclusion

The architect team is confident based on the overall requirement related to non-functional as well as functional requirement; the combination of Hadoop, Pentaho, Oracle, OBIEE, Datameer will certainly be able to solve the current problem to confidence level of 90%. The architect team also suggests looking at TIBCO Spotfire and SAP HANA for reporting as well as analysis.

6 References

CDH3 –

http://www.cloudera.com/content/support/en/documentation/cdh3-documentation/cdh3-documentation-v3-latest.html

Pentaho Data Integrator –

http://www.pentaho.com/explore/pentaho-data-integration/

OBIEE –

http://en.wikipedia.org/wiki/Oracle_Business_Intelligence_Suite_Enterprise_Edition

Datameer

http://www.datameer.com/blog/uncategorized/introducing-datameer-2-0-data-analytics-democratized.html