Chukwa: Architecture and Design

Table of contents

1. Introduction

Log processing was one of the original purposes of MapReduce. Unfortunately, using Hadoop for MapReduce processing of logs is somewhat troublesome. Logs are generated incrementally across many machines, but Hadoop MapReduce works best on a small number of large files. And HDFS doesn't currently support appends, making it difficult to keep the distributed copy fresh.

Chukwa aims to provide a flexible and powerful platform for distributed data collection and rapid data processing. Our goal is to produce a system that's usable today, but that can be modified to take advantage of newer storage technologies (HDFS appends, HBase, etc) as they mature. In order to maintain this flexibility, Chukwa is structured as a pipeline of collection and processing stages, with clean and narrow interfaces between stages. This will facilitate future innovation without breaking existing code.

Chukwa has four primary components:

1. **Agents** that run on each machine and emit data.
2. **Collectors** that receive data from the agent and write it to stable storage.
3. **MapReduce jobs** for parsing and archiving the data.
4. **HICC**, the Hadoop Infrastructure Care Center; a web-portal style interface for displaying data.

Below is a figure showing the Chukwa data pipeline, annotated with data dwell times at each stage. A more detailed figure is available at the end of this document.

2. Agents and Adaptors

Chukwa agents do not collect some particular fixed set of data. Rather, they support dynamically starting and stopping *Adaptors*, which small dynamically-controllable modules that run inside the Agent process and are responsible for the actual collection of data.

These dynamically controllable data sources are called adaptors, since they generally are wrapping some other data source, such as a file or a Unix command-line tool. The Chukwa agent guide includes an up-to-date list of available Adaptors.

Data sources need to be dynamically controllable because the particular data being collected from a machine changes over time, and varies from machine to machine. For example, as Hadoop tasks start and stop, different log files must be monitored. We might want to increase our collection rate if we detect anomalies. And of course, it makes no sense to collect Hadoop metrics on an NFS server.

3. Data Model

Chukwa Adaptors emit data in *Chunks*. A Chunk is a sequence of bytes, with some metadata. Several of these are set automatically by the Agent or Adaptors. Two of them require user intervention: `cluster name` and `datatype`. Cluster name is specified in `conf/chukwa-env.sh`, and is global to each Agent process. Datatype describes the expected format of the data collected by an Adaptor instance, and it is specified when that instance is started.

The following table lists the Chunk metadata fields.

Field	Meaning	Source
Source	Hostname where Chunk was generated	Automatic
Cluster	Cluster host is associated with	Specified by user in agent config
Datatype	Format of output	Specified by user when Adaptor started
Sequence ID	Offset of Chunk in stream	Automatic, initial offset specified when Adaptor started
Name	Name of data source	Automatic, chosen by Adaptor

Conceptually, each Adaptor emits a semi-infinite stream of bytes, numbered starting from zero. The sequence ID specifies how many bytes each Adaptor has sent, including the current chunk. So if an adaptor emits a chunk containing the first 100 bytes from a file, the sequenceID of that Chunk will be 100. And the second hundred bytes will have sequence ID 200. This may seem a little peculiar, but it's actually the same way that TCP sequence numbers work.

Adaptors need to take sequence ID as a parameter so that they can resume correctly after a crash, and not send redundant data. When starting adaptors, it's usually save to specify 0 as an ID, but it's sometimes useful to specify something else. For instance, it lets you do things like only tail the second half of a file.

4. Collectors

Rather than have each adaptor write directly to HDFS, data is sent across the network to a *collector* process, that does the HDFS writes. Each collector receives data from up to several hundred hosts, and writes all this data to a single *sink file*, which is a Hadoop sequence file of

serialized Chunks. Periodically, collectors close their sink files, rename them to mark them available for processing, and resume writing a new file. Data is sent to collectors over HTTP.

Collectors thus drastically reduce the number of HDFS files generated by Chukwa, from one per machine or adaptor per unit time, to a handful per cluster. The decision to put collectors between data sources and the data store has other benefits. Collectors hide the details of the HDFS file system in use, such as its Hadoop version, from the adaptors. This is a significant aid to configuration. It is especially helpful when using Chukwa to monitor a development cluster running a different version of Hadoop or when using Chukwa to monitor a non-Hadoop cluster.

For more information on configuring collectors, see the Collector documentation.

5. MapReduce processing

Collectors write data in sequence files. This is convenient for rapidly getting data committed to stable storage. But it's less convenient for analysis or finding particular data items. As a result, Chukwa has a toolbox of MapReduce jobs for organizing and processing incoming data.

These jobs come in two kinds: *Archiving* and *Demux*. The archiving jobs simply take Chunks from their input, and output new sequence files of Chunks, ordered and grouped. They do no parsing or modification of the contents. (There are several different archiving jobs, that differ in precisely how they group the data.)

The Demux job, in contrast, take Chunks as input and parse them to produce ChukwaRecords, which are sets of key-value pairs.

For details on controlling this part of the pipeline, see the Administration guide. For details about the file formats, and how to use the collected data, see the Programming guide.

6. HICC

HICC, the Hadoop Infrastructure Care Center is a web-portal style interface for displaying data. Data is fetched from a MySQL database, which in turn is populated by a mapreduce job that runs on the collected data, after Demux. The Administration guide has details on setting up HICC.

And now, the full-size picture of Chukwa: