Setup and Operation of Sparky

Authors: Josh Patterson, Galen Riley, and Andy Hill

Summary

- Operatation and setup document for Sparky REST server
- Details component's location and ports
- Illustrates components inter-dependencies and mechanics
- Sparky
  - provides REST access for logical data pulls
  - Provides a web ui for Boolean queries and data searches
  - Example REST call: http://sparkyserver:8084/historian/timeseriesdata/read/historic/2766/1258477319733/1258477640833/xml

Component Breakdown

- HDFS (*archive store*)
- HBase (*index store*)
- Sparky HTTP Server

HDFS

- Purpose: stores all pmu archive files

HBase

- Purpose: stores index of archive file data blocks in HDFS
- Also stores index of data mining operations, such as anomalies and specific patterns

Sparky Installation Steps

Download and Untar Sparky Project

Retrieve the project archive from codeplex at http://openpdc.codeplex.com

Configuration of Sparky

Modify bash scripts

Edit conf\sparky-site.xml to modify configuration settings.

Property	Description
hadoop.hdfs.uri	file system access point
sparky.http.port	the port on which the sparky web service should run
hbase.uri	the URI which should resolve to hbase
hbase.stargate.REST.uri	the URI which should resolve to the hbase REST service
hbase.stargate.REST.port	the port on which the hbase REST service should run

Loading Sample Data For Indexing

In order to begin indexing operations you'll need a sample openPDC archive. There is a sample archive available in the data subdirectory of the main openPDC project named "openpdc_archive.d".

Running the Indexer

To run the single file indexer from the command line type:

Bin/sparky indexer –indexFile {hdfs_path}

Once you have run the indexer on the file you should be able to see the file's contents show up in the search interface and in the REST streaming interface. For reference, simply typing

bin/sparky indexer

will give you a list of the other options offered by the indexer.

How To Start Sparky

The following steps need to happen in order (it's a rolling dependency chain).

1. Start HDFS
    a. Start the archive storage cluster, if not started
    b. Start the HBase HDFS store, if not started
2. Start Zookeeper for HBase
3. Start HBase
4. Start the Stargate REST bridge for HBase
5. Start Sparky

Start HDFS

Refer to standard Hadoop documentation on how to start your HDFS repository.

Start HBase

Refer to standard HBase documentation on how to start your HBase server. In addition to starting Zookeeper, you'll also need to setup the Stargate REST bridge to HBase for this particular version of Sparky. For this version we used different versions of Hadoop for the data repository than for HBase so it created a conflict between the jar dependencies. We got around this by using the HBase REST interface. While not ideal, this allowed us to continue developing our prototype under certain artificial constraints. Future versions of Sparky would ideally have matching versions of Hadoop and not run into this limitation. Our time series repository version was Hadoop 0.19.0 while we went with Hadoop 0.20.0 in order to get the (at the time) latest version of HBase.

Running the Sparky Server

Once your HDFS data repository and HBase servers are online you can start Sparky. The server component of Sparky is driven by bash scripts very in a similar fashion to Hadoop and HBase's hash script system. To start the web/REST server type:

Bin/sparky server start

And to stop the web/REST server type:

Bin/sparky server stop

Once the server has started, you should be able to see the web interface at:

http://sparkyserver:8084/

Running Queries via REST

- /historian/timeseriesdata/read/current/<one or more ID delimited by comma>/[xml|json]
Returns the latest time-series data for the measurement IDs specified in the comma-delimited list.

- /historian/timeseriesdata/read/current/<starting ID in the range>-<ending ID in the range>/[xml|json]
Returns the latest time-series data for the measurement IDs specified in the range.

- /historian/timeseriesdata/read/historic/<one or more ID delimited by comma>/<start time>/<end time>/[xml|json]
Returns historic time series data for the measurement IDs specified in the comma-delimited list for the specified GMT time span. The time can be absolute time (Example: 09-21-09 23:00:01 for Sep 21, 09 11:00:01 pm) or relative to the current time (Example: * for now or *-1m for 1 minute ago where s = seconds, m = minutes, h = hours and d = days).

- /historian/timeseriesdata/read/historic/<starting ID in the range>-<ending ID in the range>/<start time>/<end time>/[xml|json]
Returns historic time series data for the measurement IDs specified in the range for the specified GMT time span (time format is same as above).

Running Queries via Search Interface

The search interface has a set of parameters that can be passed. These parameters are point_id, time_range, and debug_index_bucket in the following formats.
point_id: <ID>
time_range: <start time> - <end time>
debug_index_bucket: <ID>

Where point_id and debug_index_id are integers and a time_range example would be 09-21-09 23:00:01 for Sep 21, 09 11:00:01 pm. Currently, debug overrides all other parameters.

List of Sparky JAR dependencies

These jars should be in your ./lib directory:

ant-1.7.1.jar
ant-launcher-1.7.1.jar
asm-3.2.jar
commons-codec-1.3.jar
commons-httpclient-3.0.1.jar
commons-logging-1.0.4.jar
hadoop-0.19.0-ant.jar
hadoop-0.19.0-core.jar
hadoop-0.19.0-tools.jar
jersey-core-1.1.4.1.jar
jersey-server-1.1.4.1.jar
jetty-6.1.22.jar
jetty-util-6.1.22.jar
jsp-2.1-6.0.2.jar
jsp-api-2.1-6.0.2.jar
jsr311-api-1.1.1.jar
log4j-1.2.15.jar
openPDC_Historian.jar
servlet-api-2.5-20081211.jar