Table of Contents

HBase

Open-source BigTable implementation on top of HDFS or other distributed FS in the Hadoop ecosystem. Column-based NOSQL database. A stored value is addressed by rowkey, column family, column qualifier, timestamp. From the point of view of the DB rowkey, family, qualifier and value are untyped byte arrays.

Scan
: A query to the HBase database returning multiple rows. Can be limited by a range of rowkeys, by columns and by timestamp. Can use filters for more control.

Queries

Interfaces

HBase Shell

The easiest tool to test HBase, query a table or debug a custom filter or coprocessor.

Java client

The fastest way to query HBase. See test results.

REST interface

HBase provides a REST server that can be run on any node in the cluster (or outside the cluster if desired). It acts as an HBase client and a REST server to provide access to HBase data for non-java clients.

See appendix for more. Advantages:

- Cross-platform and universal

Main problems:

- Is supposedly slower than Thrift, not tested
- Filters and scans can be used with REST, but not custom filters and not Endpoints.

Thrift interface

HBase also provides a Thrift server that can be run on any node in the cluster (or outside the cluster if desired). It acts as an HBase client and a Thrift server to provide access to HBase data for non-java clients.

Thrift is an interface definition language and automatically generates language-specific bindings for the interface.

Advantages:

- Cross-platform
- Reported as faster than REST
- Custom filters can be used if configured on the Thrift server

Disadvantages:

- Is slower than a Java client for large scans

- Language-specific bindings need to be built
- Endpoints (stored procedure Coprocessors) can't be run with Thrift

Phoenix

Phoenix is a tool to provide an interface very close to SQL to query HBase tables. Not for batch processing, not based on MapReduce.

Readme

Query language reference

Things we can learn from Phoenix:

- Key salt
- Using filters
- Using coprocessors

Reasons for us not to use Phoenix:

- JDBC interface, which means Java client. Our existing code is Python.
  - Ways to use JDBC in Python are sketchy, but may warrant another look.
- Needs to be installed on the cluster.
  - But if we use custom filters and coprocessors ourselves, they would need to be installed too.

Filters

Fliters limit the amount of data retrieved by an HBase scan based on some filtering criteria. Some useful examples: PageFilter (N) is great for debugging, limits the amount of rows returned from a scan to N.

Custom filter classes can be created. More info

Coprocessors

Introduction

Coprocessor types:

- Observers ~ triggers
- Endpoints ~ stored procedures

Endpoints can't be called through Thrift out of the box: https://issues.apache.org/jira/browse/HBASE-5600

Phoenix observers

Observers are used by Phoenix for many things, including aggregation.

We may possibly use their coprocessors directly for custom scans instead of going through jdbc. This approach suffers from the fact, that Phoenix coprocessors are very tightly coupled with Phoenix metadata and utilities. So setting the correct Scan custom attributes without Java and Phoenix classes is almost impossible.

We may also just mimic the way Phoenix uses observers to do aggregation. They were not really designed for that so it would not be trivial at all, but it is possible (Phoenix works after all). Using observers instead of endpoints wold mean aggregation is possible from any client that support custom Scan attributes, including

Thift.

Processing

HBase from MapReduce jobs

Examples

Pig

UDF (user-defined functions)

UDF are used to do any data processing within PIG more complex that restructuring bags and straightforward aggregation.

Python UDFs. Actually Jython, take care. For example, Java date libraries need to be used for any date manipulation.

Not convenient to construct rowkeys because of Unicode issues. Pig byte arrays are strings in the jython UDF, but jython only has unicode strings. As a result creating a rowkey byte array with binary values results in some bytes being encoded into pairs of bytes.

HBase from Pig

Works:

```
STORE data INTO 'ssbtest' USING org.apache.pig.backend.hadoop.hbase.HBaseStorage('cf:etime, cf:va
```

First value in the tuple used as the key. To make a compound key either use several calls to CONCAT or write a UDF

Phoenix from Pig

Phoenix provide a Pig storage class to store to a Phoenix table from Pig. This should work but fails with an error:

```
STORE data INTO 'hbase://ssbtest' USING com.salesforce.phoenix.pig.PhoenixHBaseStorage('dashb-ai-
```

Source of the class in question.

One can always use HBase storage for Pig instead. Phoenix tables are just HBase tables with some parameters and coprocessors set. The main disadvantage is concatenating the rowkey value needs to be done manually with HBase storage.

Appendix

OpenTSDB - an interesting monitoring project to (possibly) take some schema ideas from.

Benchmark scan Java vs Thrift

data	1 month aggregated transfer data, 1.7M rows
key format	bTtttVvvvVvvvSsssSsssSsssSsssDdddDdddDdddDddd
	salt = hash mod 16, 10 min time bin, vo, src, dst
scan parameters	caching = 1000 batch = 100

java client on cluster (dashb-ai-410):

time range	vo src dst	row count	prep	scan	total
1072637400 1672810200	"" "" ""	1770598			22659 ms
1372637400 1372810200	"" "" ""	105812	6 ms	831 ms	837 ms
1372637400 1372810200	"atlas" "" ""	68970	5 ms	624 ms	629 ms
1372637400 1372810200	"atlas1" "" ""	0	5 ms	103 ms	108 ms
1372637400 1372810200	"atlas" "GRIF" ""	11	5 ms	134 ms	139 ms
1372637400 1372810200	"atlas" "GRIF" ""	11	5 ms	134 ms	139 ms
1372637400 1372810200	"atlas" "" "GRIF"	1418	5 ms	270 ms	275 ms
1072637400 1672810200	"" "" "GRIF"	26456	8 ms	4743 ms	4751 ms
1072637400 1672810200	"" "GRIF" ""	1400	10 ms	1611 ms	1621 ms
1072637400 1672810200	"atlas" "GRIF" ""	417	9 ms	1195 ms	1204 ms
1372637400 1372810200	"atlas" "GRIF" "GRIF"	2	10 ms	142 ms	152 ms

WARNING: some rows are for some reason left out when src and/or dst is set. This bug needs to be found. The python+thrift code works more correctly

java client on desktop

not tested	exception when creating scan

thrift client on cluster (dashb-ai-410)

time range	vo src dst	row count	walltime
1372637400 1372810200	"atlas" "" ""	68970	11.0466649532

More info on the REST interface

In python requests is a good library for working with REST. Happybase is supposed to be a more high-level HBase-specific library, I have not had time to try it.

Rest interface reference

Filters and scans can be used with REST, but not custom filters and not Endpoints.

Filter parameters reference

Example:

```
<Scanner batch="5"> <filter>{
  "type": "FamilyFilter",
  "op": "EQUAL",
```

```
  "comparator": {
    "type": "BinaryComparator",
    "value": "dGVzdHJvdw\u003d\u003d"
  }
}</filter> </Scanner>
```

Yes, json-like inside xml. Rest also supports application/protobuf instead of XML, but filter definition probably still uses the same format. Note that values are b64 encoded.

Details about filters

Custom filters

Any filter can be used from HBase shell, provided it's in the classpath. hbase> scan 't1', {FILTER => org.apache.hadoop.hbase.filter.ColumnPaginationFilter.new(1, 0)}

To be able to use filter through Thrift or REST it needs to provide

```
createFilterFromArguments(ArrayList<byte[]> filterArguments)
```

This provides a way to create a filter from string parameters, which is exactly what Thrift/Rest provide. To debug this method for your custom filter from HBase shell, register the method (so that hbase can look up method class name from a short method name) and try a scan like this:

```
hbase> org.apache.hadoop.hbase.filter.ParseFilter.registerFilter("FuzzyRowFilterFromText", "org.a
scan 'SSB3', {FILTER => "(FuzzyRowFilterFromText ( '\x00\x00\x00\x00\x00\x00ssb_metric_18\x00\x00
```

To use the custom filter with other interfaces, it needs to be registered too:

Adding custom filters to Thrift server configuration.

In CDH4 Cloudera manager:

Services->Hbase1->Configuration->View and edit->Default Category->HBaseThrift Server (Default)->Advanced->HBase Thrift Server Configuration Safety Valve for hbase-site.xml:

```
<property>
  <name>hbase.thrift.filters</name><value>FuzzyRowFilterFromText:org.apache.hadoop.hbase.filter.F
</property>
```

This is not implemented for REST yet.

Filters with skip hints

Easy to imagine, harder to understand and implement. Faster than simple filters that just check one row.
http://hadoop-hbase.blogspot.ch/2012/01/hbase-intra-row-scanning.html
http://hadoop-hbase.blogspot.ch/2012/01/filters-in-hbase-or-intra-row-scanning.html

FuzzyRowFilter

FuzzyRowFilter is a rowkey filter allowing filter by subsets of the rowkey bytes. Advantages:

- Fast: provides skip hints to the scanner
- Structurally simple: just provide keyvalues and byte masks

- Part of HBase

Disadvantages:

- Not very friendly: construct keyvalues and byte masks yourself
- Does not support ranges for rowkey parts, only discrete values.
  - It's OK in our use-case, we only have one part of the key that needs range filtering: timestamp. We want it to be at the front of the key anyway, so we can use scan's startRow and stopRow to filter by timestamp. Then this filter would only be used to filter by resource and by metric, which don't need range filtering.
- Does not allow for variable-length rowkey parts.
  - Either we use fixed length of resource and metric strings
  - Or we use binary resource id and metric id to save space
- Does not implement createFilterFromArguments
  - This is pretty easy to fix with a subclass that implements this method
- Consequently can't be used with REST or Thrift out of the box (they need createFilterFromArguments)

SkipScanFilter

SkipScanFilter is a rowkey filter allowing to filter by key columns in Phoenix tables. Used iternally by Phoenix queries. Advantages:

- Fast: provides skip hints to the scanner
- Supports ranges for key column values.
- Supports variable-length and empty key columns.
- Easier to use: for each key column provide a set of value ranges
  - But key columns are defined in Phoenix

Disadvantages:

- Part of Phoenix
- Needs Phoenix metadata (key column parameters) to construct the filter
- Does not implement createFilterFromArguments
  - Which is harder to implement ourselves because of depending on Phoenix metadata

Cloudera CDH4

Ports used:

Used by the manager stack

Used by actual servers

Thrift moved from the default 9090 to 7182 (manager port) because we have a lot of ports open as it is. Thrift is running on host 414, manager is on host 410 so there is no conflict as of now.

in Thrift howto:

This filter does not provide the necessary method so we create our own. The class definition is here

To use this filter, build this class into a jar. Eclipse can be used if HBase and Hadoop jars are added to the build path. Otherwise one can build it on any cluster node.

Then this jar must be added to Thrift server classpath, the easiest way is to put it in /usr/lib/hbase/lib with the other HBase jars.

Then the filter must be registered through Thrift server configuration. See #ClouderaRegisterFilter for details on how to do that in Cloudera CDH4

After restarting the thrift server the filter should be available through Thrift.

To test the FuzzyRowFilterFromText wrapper see #ShellCustomFilter

SSB howto:

Put data into HDFS: If "ssb" is a local dir with your data and hdfs://user/root/ is a hdfs dir where you have write permissions

```
hadoop fs -put ssb /user/root
```

Put this data into HBase:

Example MapReduce job

Not necessary to use a MapReduce job here. This one does not even have a reducer. If necessary it should not be hard to rewrite as a standalone utility. When it's not a MapReduce job there would be no need to copy input data to HDFS first ether.

To build and run this MapReduce job on a node in the cluster:

```
cd /root/ssbMapReduce/src/
javac -cp $HADOOP_CLASSPATH:`hbase classpath` SsbStore.java &&
jar cf mr.jar *.class &&
HADOOP_CLASSPATH=$HADOOP_CLASSPATH:`hbase classpath` hadoop jar mr.jar SsbStore /user/root/ssb SS
```

To request SSB data from HBase

REST interface scan

REST interface scan with FuzzyRowFilter (does not work, REST does not support this filter)

Thrift interface scan with FuzzyRowFilter

Transfers howto:

Put data into HDFS just like SSB

Put this data into HBase: Example MapReduce job

```
cd transfersMapReduce/src/org/cern/dashboard/hadoop/ &&
mkdir -p _classes &&
javac -cp `hbase classpath` -d _classes *.java &&
jar -cvf transfers_mr.jar -C _classes . &&
HADOOP_CLASSPATH=$HADOOP_CLASSPATH:`hbase classpath` hadoop jar transfers_mr.jar org.cern.dashboa
```

Thrift client scan

Java client scan

Running java client:

```
cd transfersMapReduce/src/org/cern/dashboard/hadoop/ &&
mkdir -p _classes &&
javac -cp `hbase classpath` -d _classes *.java &&
jar -cvf transfers.jar -C _classes . &&
java -cp `hbase classpath`:transfers.jar org.cern.dashboard.hadoop.TransfersScan \
16 1372637400 1372810200 \
"atlas" "" ""
```

New topic ElasticsearchKibanaAuthenticationNotes

This topic: LCG > NoSQLStorageResearch
Topic revision: r12 - 2014-04-23 - IvanKadochnikov