Doug Meil
August 1, 2011
HBase-4147 – StoreFile Read Report, 1st pass Design

General Statement of Need

Currently, HBase RegionServer metrics high-level stats that focus on RPCs (e.g., requests = number of Get or Put RPC requests), and HBase doesn't provide much in the way to understand which tables/CFs/StoreFiles are being accessed, and for how long.

The proposal is a structure that would summarize StoreFile read statistics on a period interval:

- StoreFile
  - The full-path is required, because we also want to know /CF/region/Table.
- # of times the StoreFile was accessed in the reporting period
- Total time spent reading the StoreFile in the reporting period
- Type of read
  - "system" (e.g., compaction) vs. "user" (e.g., application)

This document is stating the general requirements and is presenting a 1st pass design. Not everything is figured out – comments welcome.

Use Cases for Output

There are several use cases for obtaining this information.

1. **UI**
   a. Having this structure available in the RegionServer's admin web pages would be useful.
2. **RegionServer API**
   a. Minimally, in order to satisfy the web-page use-case, a change to the RegionServer API would be required.
3. **Configurable Output and 3rd Party Operational Tools**
   a. Ganglia, etc.
      i. 3rd party tools like Ganglia would conceivably be interested in this information.

ii. As an aside, though, I'm not entirely sure how it would handle it because the resulting information is not a single datapoint, but a list of complex structures.

b. JMX

i. I'm assuming that folks would be interested in seeing this information exposed via JMX, but I have the same question about use (i.e., not a single "metric" but returning a list of complex structures).

ii. For what it's worth, at Explorys we found that using Jconsole to connect to the cluster made HBase unstable, so we don't do it.

c. Custom Output

i. The idea that Todd had raised on the dist-list was a configurable sink for the summary structure such that it could either be written to custom logs, or emailed, etc.

ii. HBase Logs

1. It is the opinion of the author that HBase should be "self sufficient" to a great extent in terms of diagnostic capabilities. For example, an HBase user should not be *required* to install additional tools to diagnose HBase.

2. Can additional tools be helpful? Absolutely. But it should not be a requirement for every user to write a custom script or install a 3rd party framework to capture this information (capture is the important point here, not processing).

3. Thus, the capability to periodically spool a StoreFile Read summary report to the HBase logs (e.g., write output every XXXX), with a default of 5 minutes or something.

4. If a user wanted to make a cursory manual inspection of what is changing over time, the logs will show the contents with no additional capture responsibilities.

d. Else?

Proposed Changes

1. **StoreFileScanner**

    a. Member variable indicating "read type" (i.e., system vs. user).

        i. This would default to user.
        ii. A new overloaded constructor could be created that would take StoreFile.Reader, HFileScanner, and an enum for ReadType.)

    b. Member variable with the total number of milliseconds of time spent reading.

    c. Instrument the following methods to increment read-time…

        i. `next()`
        ii. `seek()`

        iii. Question: what about the static methods `seekAtOrAfter`, and reseekOrAfter?

    d. Send the individual read statistics to a collection service on `close()`.

        i. The intent of the send being done on close is to minimize the chattiness of this collection process.

        ii. Question: are *StoreFileScanners* typically closed? If not, that's a problem for this approach.

            1. Based on what I understand of the code, *StoreScanner's* `close()` method would need to be altered to also close the *StoreFileScanners*.

2. **Store**

    a. It would be extremely helpful to differentiate the reads that are initiated by system activity and user (i.e., application) reads.

    b. For example, in *Store's* `compactStore` method, if there was a way to pass ReadType.System to the *StoreFileScanners* it would achieve this goal.

3. **StoreFileRead (new)**

    a. This class represents and individual StoreFile read (i.e., not summary) and would have the following structure:

        i. ReadType
        ii. StoreFile path
            1. Preferably, the act of recording a read statistic would not do any parsing.
            2. It is assumed that the StoreFile path is the full-path (e.g., table/region/CF/StoreFile)
        iii. Milliseconds spent reading the StoreFile

    b. StoreFileScanner would post an instance of this class to a "Read Stats Collection Service."

4. **Read Stats Collection Service (new)**

    a. Hand-waving here.

    b. A design priority is not to do any summarization in the thread that is doing the reading.

    c. StatisticsCollector (Straw Man)

        i. *StoreFileScanner's* `close()` method could do the following…

        ii. `StoreFileRead sfr = new StoreFileRead( storeFile, millis);`
        iii. `StatisticsCollector.record( sfr );`

        iv. *StatisticsCollector's* `record` method would drop this *StoreFileRead* instance onto a *ConcurrentLinkedQueue*, and be done with it.

5. **Read Stats Summarization Service (new)**

    a. *StatisticsCollector* would require a background thread to drain and summarize the contents of the individual *StoreFileRead* instances.

    b. Because the intent of the summarized report is to be period-based (e.g., every minute? Every 5 minutes?), the summarization thread needs to be aware of the intended period.

        i. This would need to be configured for the RegionServer.

c. Once calculated, it could make the current summary-view available via `List<StoreFileReadSummary> = StatisticsCollector.getStoreFileReadSummary();`

    i. Where *StoreFileReadSummary* would be defined as

        1. ReadType
        2. Table
        3. Region
        4. ColumnFamily
        5. StoreFile
        6. # of times the StoreFile was accessed in the period
        7. # of milliseconds processing the StoreFile in the period.
        8. periodLength in millseconds
            a. arguably, this could be in a containing object.
        9. Time summary generated
            a. Ditto.

6. RegionServer API

a. Add `List<StoreFileReadSummary> getStoreFileReadSummary` to *HRegionInterface*

b. Add implementation of `getStoreFileReadSummary` in *HRegionServer*

c. Question: RegionServer API effect on *StoreFileReadSummary* class.

    i. Everything that is returned by the RegionServer API implements *Writable*.

7. Web UI

a. This could either be jammed onto the existing RegionServer detail page (preferably at the top), or a new web-page linked from the RegionServer detail page could be created.

b. Because this is summary information (reminder: at table/CF level), it might still fit on the same page.

8. Configurable Output

a. More hand-waving starts.

b. JMX

i. RegionServer statistics are exposed via *RegionServerStatistics* (which extends *MetricsMBeanBase*)

ii. Question: does this information go in *RegionServerStatistics*, or a new *StoreFileReadSummaryStatistics* class?

1. It feels like the latter, but I'd like others to comment on this.

c. Ganglia

i. At the moment I'm not entirely sure how Ganglia integrates with HBase (e.g., JMX vs. direct RS API calls).

d. Custom Output

i. Serious hand-waving here.

ii. What if I wanted this information logged to a file whenever a new summary is calculated?

iii. I think a more general pattern for metrics/statistics output is needed.

iv. Straw-Man Proposal

1. *StatisticsCollector* has configurable output formatters.

2. And also has pluggable output formatters.
   a. E.g.,
   b. MyStoreFileReadSummaryFormatter extends StoreFileReadSummaryFormatter
      i. formatOutput(StoreFileReadSummary summary)
      ii. // your code goes here.

3. But some formatters are shipped out of the box, such as the log-file one.