Streaming Percentile Estimatation

Background

I'm looking into better ways of doing estimation of high-percentile latency for HBase metrics (which also applies to HDFS). There are a lot of different algorithms out there, and the right one depends on our requirements.

Right now, HBase uses forward-biased reservoir sampling, which is an efficient way of randomly sampling some fixed number k "recent" values from a stream. This is fine for calculating gross statistics (avg, stddev, min, max), and gives a sense for how the stream is changing over time. Best, bounded memory usage.

However, it doesn't give good (any?) error bounds on calculating high percentiles, since random sampling does not capture the tail well. It also doesn't translate easily into time windows, since the eviction rate of the reservoir is based on the exponential decay function chosen.

I think a better metrics framework would allow for capturing latency over multiple different time windows (e.g. 10s, 1m, 5m, 15m), and doing it with bounded error (e.g. 99th percentile with 0.1% error). With these as design criteria, I'm listing what I think are the two best candidates. I also do some comparison to simpler approaches that do not have bounded error.

Candidates

- "Effective Computation of Biased Quantiles over Data Streams" by Cormode, Korn, Muthukrishnan, Srivastava.
    - This algorithm (CKMS) can do efficient calculation of high percentiles, but can't do it for a sliding time window. You'd have to roll it over every interval and start a new one.
- "Approximate Counts and Quantiles over Sliding Windows" by Arasu and Manku.
    - This algorithm can do a sliding window, but requires storing significantly more state than CKMS as a result.

CKMS

You want less error the higher the percentile you ask for. That is, 90th percentile latency with +- 1% error, but 99th with +- 0.1% error. Getting more accuracy requires more memory, but there are nice ways of doing this for high percentiles. This is exactly what CKMS does.

I think reasonable error defaults can be chosen here. Todd brought up that operators like to

think about it in the reverse though, instead choosing some memory bound (order 10s of MBs) and taking whatever accuracy results from that.

Based on some small tests, I was getting really good estimates for 50,000 requests with ~1000 sampled items, so O(kilobytes) of memory. Scaling is log factor with the # of requests in the interval, so I think we're looking at O(megabytes) total per regionserver.

I think that's good enough not to worry about it, but I can also probably crunch the math to enable setting a desired memory bound.

CKMS can't do sliding windows. Instead, you rollover every interval by taking a snapshot of the percentiles you're interested in and then starting a new CKMS run.

Arasu and Manku

The main draw here is that it can do arbitrary percentiles for sliding time windows. However, this basically requires keeping around a lot of fine-grained snapshots of CKMS runs, and requires something like O(log(N)*log(1/error)) more state.

To give a small idea, if you want 0.1% error, you have to save 1000 items per CKMS snapshot, thus ~8KB for longs. Assuming 10k req/s and a 5 min window (3mil reqs), according to the paper you'd need 1500 of these snapshots => 12MB. This is in addition to the memory for the instances of CKMS you're currently running.

So, significant but doable if sliding windows are advantageous. The algorithm is also somewhat more complicated, which is why generally I'd like to just rollover CKMS periodically.

Simpler things

Pre-binned histogram

If all people want is a rough feeling for the latency distribution, you can chunk up your expected range of latencies into a number of bins, and then just maintain a counter for each bin. The bins could even be log-scale for efficiency. This is O(b) memory for b bins.

This has the same caveats on accuracy vs. memory and rollover vs. sliding window, since the algorithms are very similar. Sliding windows would probably require a different algorithm, and no longer be O(b).

Deciding the right bins would be something of a tuning parameter too, since it depends on your machine configuration and workload.

SLA compliance counter

Perhaps the base-case of the pre-binned histogram, you basically just have two bins: one bin for requests meeting the SLA, one for requests that don't. Then you have an easy boolean indicator for if the SLA is being met. Drilling down further would be left to other means. Same caveat for rollover vs. sliding window.

Exponentially-weighted stochastic approximation

See “Incremental Quantile Estimation for Massive Tracking” by Chen, Lambert, Pinheiro.

Kind of an old paper. It’s really efficient in that it just incrementally adjusts an estimate every time it sees a new item, but I don’t think there is a theoretical bound on error, and the empirical evaluation shows kind of large error. It doesn’t slide or take recency into account, but it could be rolled over.

As a bonus though, it’s already implemented in Mahout as OnlineSummarizer.

Detailed Paper summaries

Summaries for the two algorithms I liked:

“Effective Computation of Biased Quantiles over Data Streams” by Cormode, Korn, Muthukrishnan, Srivastava.

This one is for distinct time windows, it doesn’t have a time-based sliding mechanism. It allows you to specify targeted (percentile, error) pairs that you’re interested in, and then it adjusts its sampling to use the minimum required error to satisfy the (percentile, error) constraints. For instance, if you ask for (0.9, 0.01) and (0.99, 0.001), it can sample at 1% error around the 90th percentile, and at 0.1% error around the 99th percentile. This is clearly more efficient than sampling at 0.1% error across the entire interval.

They also provide some implementation details and benchmark numbers. They managed to do this for gigabit ethernet at line rate with 26% of a 2005-era core, but that was in C++.

“Approximate Counts and Quantiles over Sliding Windows” by Arasu and Manku.

This one is for sliding windows. It essentially uses a less general form of CKMS called GK to store “sketches” of various window lengths (where length is some # of measurements), which characterize the distribution for that window. A sketch with GK is done by deciding on what the minimum error for the sketch has to be, and then saving each (percentile % error == 0) percentile. This has to be uniform sampling, not biased as in CKMS.

It stores a hierarchy of these sketches, where each sketch up in the hierarchy covers double the # of measurements. Then, you have O(log(N)) levels in this hierarchy, where N is the # of measurements in your biggest desired sliding window. The levels of the hierarchy overlap, so all of your N measurements are covered by a sketch in each of the log(N) levels. There’s a nice diagram in the paper if this didn’t make sense.

You can then compute a sliding window over the last n measurements. This is done by merging the sketches for the window, trying to first use the biggest sketches since they’re the most accurate. I think you can translate this count-based window into a time window by storing a timestamp with each sketch, and expiring them as they get too old, and you can maybe make the N dynamic too.

Since you have to use uniform error for the sketches, and keep around this hierarchy of

sketches, it requires a lot more memory than rolling over CKMS for distinct time intervals. Assuming you want 0.1% error, you'd have to save 1000 items in each sketch, times O(log(N)) sketches, in addition to the sketches you're building currently (another O(log(N)) * space requirement for GK).