Entity Priority

We can't discuss how to admit entity states until we agree on the primitive measuring an entity's priority in the cache. We don't want to admit the latest entity for correctness by throwing out a hot entity. We need a time-decayed count sensitive to the number of hits, length of time, and sampling interval. For example, an entity from a 5-minute interval detector that is hit 5 times in the past 25 minutes should have an equal chance of using the cache along with an entity from a 1-minute interval detector that is hit 5 times in the past 5 minutes. It might be the case that the frequency of entities changes dynamically during run-time. For example, entity A showed up for the first 500 times, but entity B showed up for the next 500 times. The formula should give entity B higher priority than entity A as time goes by. We next describe our priority measurement.

Each detector has its own landmark L: period 0 when the detector is enabled. Consider one entity's count increments at period $i > L$. Given a positive monotone non-decreasing function g, the decayed priority of the increment for the entity period $p \geq i$ is:

$$w(i, p) = \frac{g(i-L)}{g(p-L)}$$

For each cache hit, we maintain the sum of $g(i - L)$. Consider the stream of an entity's increment period i

$$S = \{3, 4, 5, 7, 8\}$$

Recall that the landmark period $L = 0$, and set $g(n) = e^{0.125n}$. We use the decay constant 0.125. The half life is 8* ln(2). This means the old value falls to one half with roughly 5.6 intervals We chose 0.125 because multiplying 0.125 can be implemented efficiently using right shift and the half life is not too fast or slow . Evaluated at $p = 10$, the decayed priorities are respectively

$$\{0.50, 0.55, 0.61, 0.74, 0.82\}$$

The sums of $g(i - L)$ during the process of cache hits are

$$\{0.50, 1.05, 1.66, 2.40, 3.82\}$$

Each detector maintains an ordered map, filled by entities's accumulated sum of $g(i - L)$. Since $g(p - L)$ is changing and they are the same for all entities of the same detector, we can compare entities' priorities $w(i, p)$ by considering the accumulated sum of $g(i - L)$.

When needing to replace an entity, we query the minimum from each ordered map and compute $w(i, p)$ for each minimum entity by scaling the sum by $g(p - L)$. Notice $g(p - L)$ can be different if detectors start at different timestamps. The minimum of the minimum is selected to be replaced. The number of multi-entity detectors is limited (we consider to support ten currently), so the computation is cheap.

Theorem: The sum of exponential functions can grow large and potentially exceed the capacity of common floating point types. We can express all of the values in log space to avoid overflow.

Proof (to Joyce's credit):
Instead of growing exponentially, new terms grow linearly. The relationship between entities (greater than, equal to, less than) will be the same. Previously, the decayed count C is the sum of decayed priorities of stream items

$$C = \sum_{i=1}^{n} (g(t_i - L)/g(t - L))$$

In log space, the decayed count becomes:

$$\log(C) = \log(\sum_{i=1}^{n}(g(t_i - L)/g(t - L))) = \log(\sum_{i=1}^{n}(g(t_i - L)) - \log(g(t - L))$$

To add a new term each time the entity gets a hit after the first hit, we use the log summation relationship:

$$log(a + b) = log(a) + log(1 + \frac{b}{a}) = log(a) + log(1 + e^{log(b) - log(a)}) \quad \text{(i)}$$

So the update step would be

$$\log(\sum_{i=1}^{k}(g(t_i - L)) = \log(\sum_{i=1}^{k-1}(g(t_i - L)) + \log(1 + e^{\log(g(t_k - L)) - \log(\sum_{i=1}^{k-1}(g(t_i - L)))})$$

Let $old\ count = \log(\sum_{i=1}^{k-1}(g(t_i - L))$. The update step is given by

$$\log(\sum_{i=1}^{k}(g(t_i - L)) = old\ count + log(1 + e^{\log(g(t_k - L)) - old\ count}) \quad \text{(ii)}$$

However, in this update step, we need to compute $e^{\log(g(t_k - L)) - old\ count}$, which can still result in overflow if $e^{\log(g(t_k - L)) - old\ count}$ is large. This can happen if it has been a long time since the entity's last hit (e.g. an entity with 1-minute interval that receives a hit after more than 88+ minutes without a hit. 88 minutes are derived by solving the equation $e^x > max\ float$).

However in the cases where $\log(g(t_k - L))$ is so much larger than $old\ count$ that $e^{\log(g(t_k - L) - old\ count}$ would overflow, then $\log(g(t_k - L))$ is an extremely close approximation to the true count at t_k. To show this is true, according to (i) and (ii):

$$\log(\sum_{i=1}^{k}(g(t_i - L)) = old\ count + (\log(g(t_k - L)) - old\ count) + \log(1 + e^{-(\log(g(t_k - L) - old\ count))})$$

If $e^{\log(g(t_k - L)) - old\ count}$ is so large it overflows, then its inverse will be so small it underflows to 0, so that last term is approximately $\log(0 + 1) = 0$. Then:
$\log(\sum_{i=1}^{k}(g(t_i - L)) = \log(g(t_k - L))$. In fact, this approximate $\log(\sum_{i=1}^{k}(g(t_i - L))$ is so close to the true $\log(\sum_{i=1}^{k}(g(t_i - L)))$ that a float doesn't even have enough precision to distinguish the difference.

IMPORTANT NOTE: this approximation is only close if $e^{\log(g(t_k - L) - old\ count}$ is large. If it isn't large enough to cause overflow, just use the exact calculation.