June 26, 2013 (4/2/14 rev)
PARC

Rank Statistics for High Speed Content Stores

Marc Mosko[1]*

Abstract
In a content-centric network, routers may have a Content Store to serve as an intermediate cache of objects. If backed by disk, these content stores could be very large, comprising millions of objects. An object replacement policy needs to know object rank statistics such that popular objects remain in the cache and less popular become eligible for eviction. We present an on-line high-speed algorithm to determine if a given object is in the top-nth percentile.

Keywords
Content Centric Networks – Named Data Networks

[1] *Palo Alto Research Center*
***Corresponding author**: marc.mosko@parc.com

Contents

Introduction

We use a self-sampling paradigm and exponentially weighted moving averages to determine the top Nth percentile of popular content. By self-sampling, we mean that we only check content popularity when it is accessed. Popular content will be accessed many times, and thus sampled many times, while unpopular content will only be sampled a few times. When content is accessed, we calculate the mean service rate of that content per sampling period τ. This is then combined with a system-level mean service rate. If the content's service rate is within the top Nth percentile of the system service rate, the content is deemed "popular".

1. Algorithm

An Exponentially Weighted Moving Average (EWMA) is defined as in Eq. 1. A sample X_i is added to a moving average with weight α.

$$Y_i = \alpha * X_i + (1-\alpha) * Y_{i-1} \tag{1}$$

The coefficient α comes from the filter time constant τ and the time since the last sample T:

$$\alpha = 1 - e^{-T/\tau} \tag{2}$$

If multiple samples are added to the filter in the same time step (same T), then the subsequent samples are added as:

$$Y_i = \alpha * X_i + Y_i \tag{3}$$

We also define an Exponentially Weighted Moving Variance (EWMV) similarly to Eq. 1, except we are adding in the sample error from the mean, squared:

$$V_i = \alpha * (X_i - Y_{i-i})^2 + (1-\alpha) * V_{i-1} \tag{4}$$

To calculate rank statistics, whenever a content object satisfies a request, we add in a "1" value as X_i. Thus, for each content object, we have an EWMA of the number of service responses per time constant τ. Call this value Y_{oid}, where *oid* is the object id.

In addition, we maintain a system-wide service response rate calculated the same way, but for all service requests that are satisfied. Call this Y_{sys} for the system service response rate.

We also maintain an EWMA of the per-object service response rates. After calculating Y_{oid} for a given object, we add it in to A_{sys}. This gives a system-wide total response rate.

The system-wide normalized average response rate is A_{sys}/Y_{sys}. If one wishes to avoid division, one could think of the scaled per-object rate as $Y_{oid} * Y_{sys}$.

We also compute a system-wide EWMV as V_{sys}, by adding in the value $(Y_{oid} * Y_{sys} - A_{sys})^2$.

The computation of V_{sys}, A_{sys}, Y_{sys}, and Y_{oid} is done for each object access. Because the value of X_i is "1" for Y_{oid} (i.e. we execute eq. 1 for each service request), this process requires 6 multiplies and 5 additions per sample added in the same T. The first sample takes an additional multiple and addition. The values for α may come from a table lookup, and do not require calculating an exponential.

To find popular content in the top-Nth percentile, we need to find the inverse error function of N, which gives us the distance from a (0,1)-Normal mean that covers the top-Nth percentile. This process is done by a general purpose CPU and stored in the network processor whenever N changes. We use an upper threshold t_u and lower threshold t_l to provide hysteresis.

A piece of content is deemed popular if, after calculating the above values:

$$Y_{oid} * Y_{sys} >= A_{sys} + t_u * V_{sys} * Y_{sys} \tag{5}$$

It is deemed unpopular if

$$Y_{oid} * Y_{sys} < A_{sys} + t_l * V_{sys} * Y_{sys} \tag{6}$$

Thus, a piece of content's rank is calculated with a small number of multiply and add operations per content access.

As time progresses, items that are marked as popular, because their recent service rate is above t_u, may become unpopular. When their service rate falls below t_l, they are marked as unpopular and, for example, their storage is moved to an "available" list. Some items, however, may become unpopular very quickly and their request rate go to identically 0. In this case, they will not be sampled again and will remain in the "popular" category taking up that premium storage space. Therefore, a background task to slowly search for such instant deaths may be needed.

2. Pseudocode

Algorithmically, we use the following pseudo-code to compute an EWMA for Eq. 1 . In the initialization, if it is the very first value, we use $v/8$ to start a slow ramp up to v, rather than jumping immediately to the value. Other proportionality constants could be used, though for performance using powers for two for right shifts is efficient.

This formulation also works for EWMV (Eq. 4, where v is $(X-Y)^2$, where Y is the current EWMA.

```
Add_sample(time t, value v)
   if this is the first value
      last_sample_time = t
      before_last_time = 0
      current_value = v / 8
      return

   if t equals last_sample_time
      delta = t { before_last_time
      alpha = lookup(delta)
      current_value = alpha * v + current_value
      return
   else
      ; in this case, it's a new sample period
      delta = t { before_last_time
      alpha = lookup(delta)
      current_value = alpha * v +
                  (1 - alpha) * current_value
      before_last_time = last_sample_time
      last_sample_time = t
      return
```

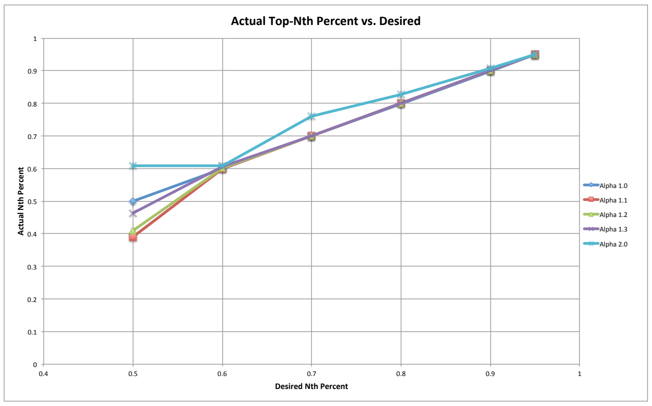

Figure 1. Actual vs. Desired Nth percentile ranking

3. Optimization

To execute an EWMA filter in a network processor with limited 32-bit ALU, we create a pre-populated table of α values, such as computed by an attached general purpose CPU. If we are working in milli-second timesteps, then a table of 2048 entries covers over 2 seconds. We also use fixed-point math for the calculation of Y, such as 5 decimal digits to capture the resolution needed for the α values. Calculations of Y could be done using 64-bits, then rounded down to the top 32-bits.

In a system with a time constant of, for example, 1 minute and a time system in 100 msec resolution, we have $\tau = 600$.

A system could also take a "controller" approach to calculate t_u and t_l such that the number of items in the cache is around some number. For example, a PID-style controller could adjust the thresholds such that the number of items marked "popular" is about some number k.

4. Experimental Results

We use a custom event simulator to model a large, 10-million item content store. We use a time clock resolution of 10 msec and allow up to 100 object request arrivals per interval, i.e. up to 10,000 requests per second. Content requests arrive in a Zipf distribution with Zipf α of 1.0, 1.1, 1.2, 1.3, or 2.0. The algorithm is tested for top-Nth percentiles of 50%, 60%, 70%, 80%, 90%, and 95%. The filter time constant was 10 seconds (τ) and we ran the experiment for 100 seconds of simulated time, with three trials per parameter combination (Zipf α and Nth percentile).

We calculated the actual Nth percentile ranked as "popular" compared to the desired percentile rank. The mean squared error averaged 0.0013, with most trials being exact out to 3 decimal places. The worst MSE was 0.0116, which corresponded to ranking the 50th percentile for a Zipf alpha of 2.0. This is because it only takes 1 or 2 content objects to cover the lower 50th percentile at that alpha. Our algorithm actually chose 60.8% of the content (2 objects) as popular, instead of 50%.

5. Conclusion

We have presented a system that calculates which items in a very large population fall in to the Nth percentile using a self-sampling paradigm. The method uses only a small number of multiplies and additions per sample from the population. Experiments on items distributed with a Zipf distribution show that the system accurately captures the top Nth percentile, except for low N, such as around 50%, where the sample variance becomes high due to the extremely small number of objects.