Generate Data for Memcached Experiment

Zhenghao Qian ....

February 3, 2013

What Model We Choose

"Workload Analysis of a Large-Scale Key-Value Store" analyzes 5 kinds of data on Facebook's production memcached servers: user-account status infomation(USR), object metadata of one applicatioin(APP), general-purpose data(ETC), server-side browser information(VAR) and system data(SYS). Among them, the paper does a case study on ETC, "the most representative of large-scale, general-purpose KV stores" and analyzes the statistical models for key sizes, value sizes and inter-arrival gaps.
To do generate universal traffic flow, we choose to model ETC-like data in our experiment. The data is generated according to the models given in the paper. The generator picks a random floating number p between 0 and 1 (with the current time stamp as the seed) and plugs p into the inverse of cumulative distribution function to get the random variable we need. The unit for key size and value size is byte and the unit for inter-arrival gap is microsecond.

How We Generate Random Data

1. The model for key sizes in bytes is Generalized Extreme Value distribution with parameters $\mu = 30.7984$, $\sigma = 8.20449$, k = 0.078688. Note that k is equivalent to ξ, the shape.

   In our case, $k \neq 0$, so the CDF of Generalized Extreme Value Distribution is:
   $$CDF = e^{-t(x)} \tag{1}$$
   $$t(x) = (1 + \xi z)^{-1/\xi} \tag{2}$$
   $$z = \frac{x - \mu}{\sigma} \tag{3}$$
   We calculate the inverse of CDF:
   $x = \sigma \frac{(\frac{1}{-ln(p)})^k - 1}{k} + \mu$
   p is a random variable between 0 and 1 and x is the random value we need.
   Most of the generated data falls between 0 and 100, which correlates with the results from the paper.

2. The model for value sizes in bytes is Generalized Pareto Distribution with parameters $\mu = 0$, $\sigma = 214.476$, k = 0.348238. Note that k is equivalent to ξ and μ is equivalent to θ.

$$CDF = 1 - (1 + kz)^{-1/k} \tag{4}$$

$$z = \frac{x - \mu}{\sigma} \tag{5}$$

   We calculate the inverse of CDF:
   $x = \sigma \frac{(\frac{1}{1-p})^{k} - 1}{k} + \mu$
   Similarly, plug in p and get the desired value size x.
   Most of the generated data falls between 100 and 1000, which corresponds to the observation on the paper.

3. The model for inter-arrival gates is Generalized Pareto with parameters $\mu = 0$, $\sigma = 16.0292$, k = 0.154971. CDF is the same as above. Almost all of the inter-arrival gap variables are under 100, which means most of them are under 0.0001 second. Because of the latency of the simulator, we could omit this inter-arrival gap for now.

In the end, we fill keys and values of random sizes with upper-case letters.

Client Programs

In general, we have four client programs:

1. traffic generator generates the data offline: it stores the exact key sizes, value sizes and inter arrival gaps in three separate files. We put those three files in the disk image of each node. The keys are all encoded under certain format: key size times index modulo 26. So every key is deterministic, if the key size is known. Therefore, auto memcached can read the same keys from the servers.
   Usage: [how many random numbers generated] [k/v/i for key, value or inter-arrival gap]
   Square brackets means required attributes, () means optional attributes.

2. data init reads the exact keys and value sizes, fills values with random upper-case characters and stores the key-value pairs to memcached servers. The last two programs won't start until data init finishes storing all the data. This program runs on client side and talks to servers nodes.
   Usage: [core id] [number keys to store] (base addr) (port)

3. ping memcached pings the memcached servers periodically for servers' throughput(bytes read and bytes written), the free memory and cpu workload. This program

runs on server side after memcached servers start in daemon mode.
Usage: [ping interval] [core id starting from 0] [times]

4. auto memcached reads the exact same keys as those stored by data init and query the values from memcached server nodes. For every value, it does a simple check sum to ensure the integrity of values. It also models cache misses(hit disks) by adding a fixed process latency.
   Usage: [how many servers] [number keys to read] [whether it checks the returned value](ip addr) [miss penalty] (port)

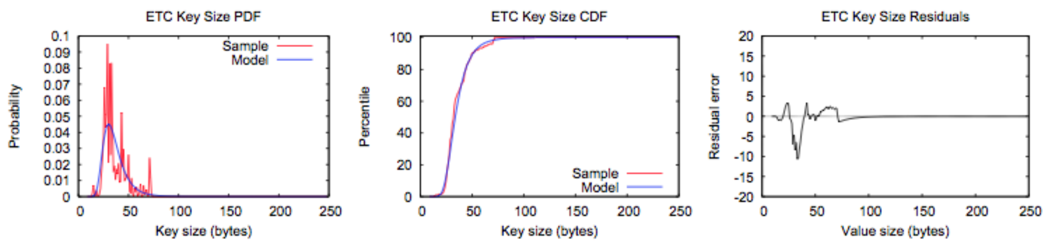

Figure 1: Excerpted from the paper. Distribution model for key sizes.

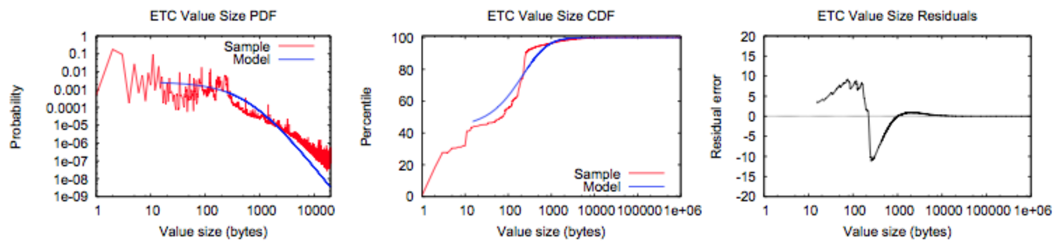

Figure 2: Excerpted from the paper. Distribution model for value sizes.

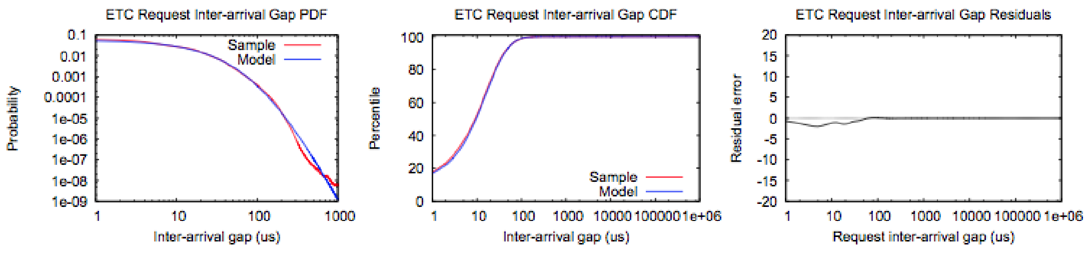

Figure 3: Excerpted from the paper. Distribution model for key sizes.