Assessment of bucketing used for backend tests which report to CirrusSearchUserTesting log

Mikhail Popov

2 December 2015

Summary

After the experimental analysis of our previous A/B test (Language Switch test), we became concerned about our procedure of selecting users for testing and assigning them to experimental/control groups. While the web queries were evenly bucketed, API queries were not as evenly bucketed. In this assessment we validated the technique and showed that the users were evenly bucketed from both sources, but that heavy users of the API skewed the bucketing proportions as hypothesized.

Background (by Erik Bernhardson)

The last analysis Oliver ran showed that the balance of requests for web were fairly even, but that significantly more requests ended up in bucket a for the API (Source).

This is most likely because we are using a consistent bucketing scheme. This scheme is:

1. Take the IP address + x-forwarded-for + user-agent, md5 them together using ':' separator (see https://github.com/wikimedia/mediawiki-extensions-CirrusSearch/blob/master/includes/ElasticsearchIntermediary.php#L593)
2. Convert that 128 bit number to a floating point probability between 0 and 1 (see https://github.com/wikimedia/mediawiki-extensions-CirrusSearch/blob/master/includes/UserTesting.php#L182)
3. Accept all users that meet `1/$sampleRate >= $probability` (see https://github.com/wikimedia/mediawiki-extensions-CirrusSearch/blob/master/includes/UserTesting.php#L207)

Most likely the reason for the misbalance is that some users send 1 or 2 requests and some users send 100k requests. Those heavy users will bias whatever bucket they end up in.

Data Collection

From the same report, we saw that 8 November 2015 was the busiest day, so we have decided to study the Cirrus search request logs from that particular date. We fetched a little over 41 million records via Hive. We kept a random sample of 1 million queries.

```
queries <- paste("USE EBernhardson;
SELECT ts, identity, source, day
FROM CirrusSearchRequestSet
WHERE year = 2015 AND month = 11 AND day =", sprintf("%02.0f", 8),
"AND INSTR(requests.querytype[SIZE(requests.querytype)-1], 'full_text') > 0")
hashes <- lapply(queries, function(query) {
  cat("Running hive query:\n", query, "\n\n")
  query_dump <- tempfile()
  cat(query, file = query_dump)
  results_dump <- tempfile()
  try({
```

```
    system(paste0("export HADOOP_HEAPSIZE=1024 && hive -S -f ", query_dump, " > ", results_dump))
    results <- read.delim(results_dump, sep = "\t", quote = "", as.is = TRUE, header = TRUE)
  })
  file.remove(query_dump, results_dump)
  return(results)
})[[1]]
hashes$day <- NULL
hashes$timestamp <- as.POSIXct(as.numeric(as.character(hashes$ts)), origin = '1970-01-01', tz = 'GMT')
hashes <- hashes[sample.int(nrow(hashes), 1e6), ]
```

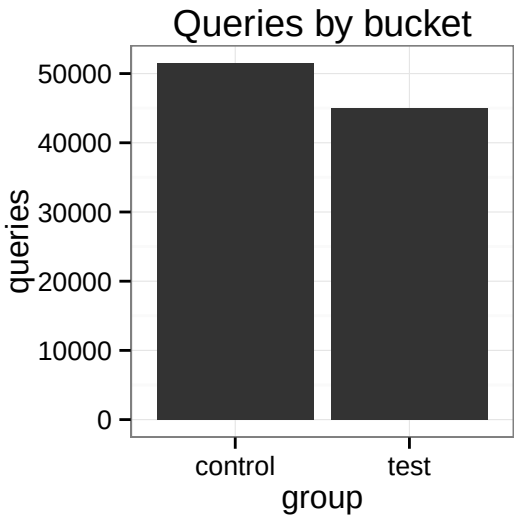
Now that we have the identities hashes – MD5(IP+XFF+UA) – we need to replicate the hex-to-probability procedure. (We have verified the procedure to make sure it yields results identical to the PHP code linked to above.)

```
library(bitops) # install.packages('bitops')
hex_to_prob <- function(hash) {
  hash_sum <- 0
  for ( i in seq(1, nchar(hash), 4) ) {
    dec <- strtoi(substr(hash, i, i + 3), base = 16)
    # // xor will retain the uniform distribution
    hash_sum <- bitXor(hash_sum, dec)
  }
  return(hash_sum / (bitShiftL(1, 16) - 1))
}
hashes$probability <- sapply(hashes$identity, hex_to_prob)
```

Next we replicate the procedure for selecting a query to be included in the test, and, if selected, which bucket to place it in. This is a deterministic process dependent on the probability derived above.

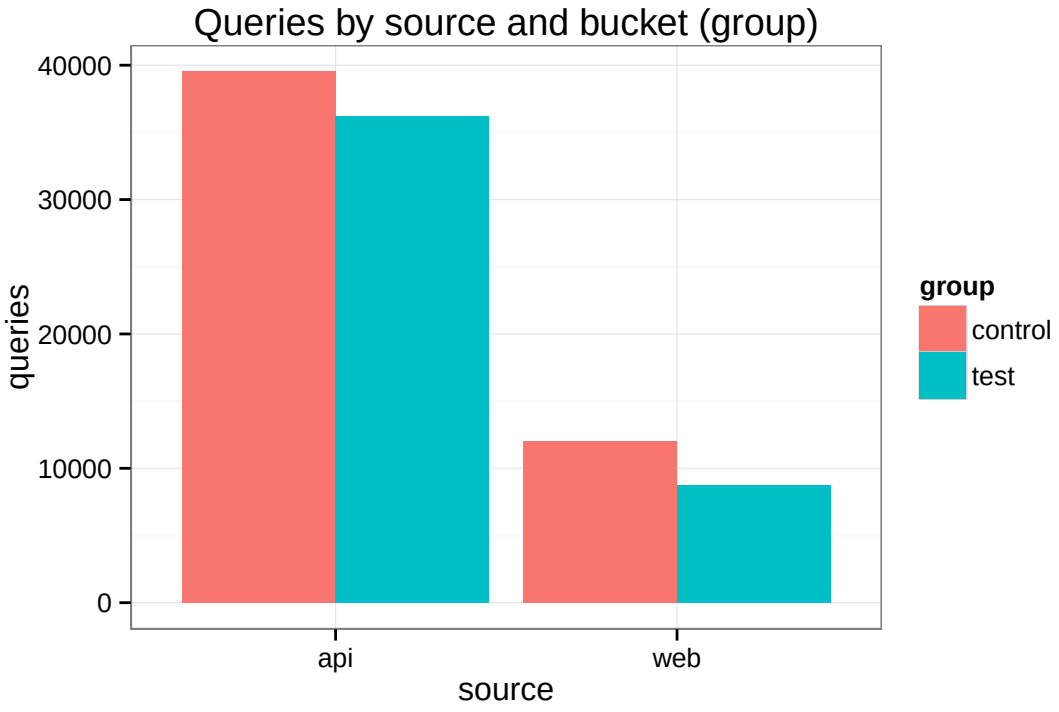
```
one_in <- function(probs, sample_rate) {
  rate_threshold <- 1/sample_rate
  temp <- numeric(length(probs))
  temp[rate_threshold >= probs] <- probs[probs < rate_threshold] / rate_threshold
  return(temp)
}
hashes$group <- 'not in test'
hashes$bucket_probability <- one_in(hashes$probability, 10)
hashes$group[1/10 >= hashes$probability] <- 'in test'
hashes$group[hashes$group == 'in test' & hashes$bucket_probability < 0.5] <- 'control'
hashes$group[hashes$group == 'in test' & hashes$bucket_probability >= 0.5] <- 'test'
```

Analysis



This is consistent with what we've been seeing in our previous tests. The hash-to-probability-to-bucketing pipeline does not yield evenly sized buckets *with respect to the volume of queries*.

Queries by Source and Group



From this figure it is not clear whether the bucket splits are different betweeen the two sources.

Null Hypothesis (H0): Bucketing proportions are independent of source.

```
##       group
## source control  test
##    api   0.522 0.478
##    web   0.578 0.422
```

```
## $`Bayes Factor`
## [1] 2.8e+42
##
## $Interpretation
## [1] "Very strong evidence against H0."
```

That is, there is strong evidence of association.

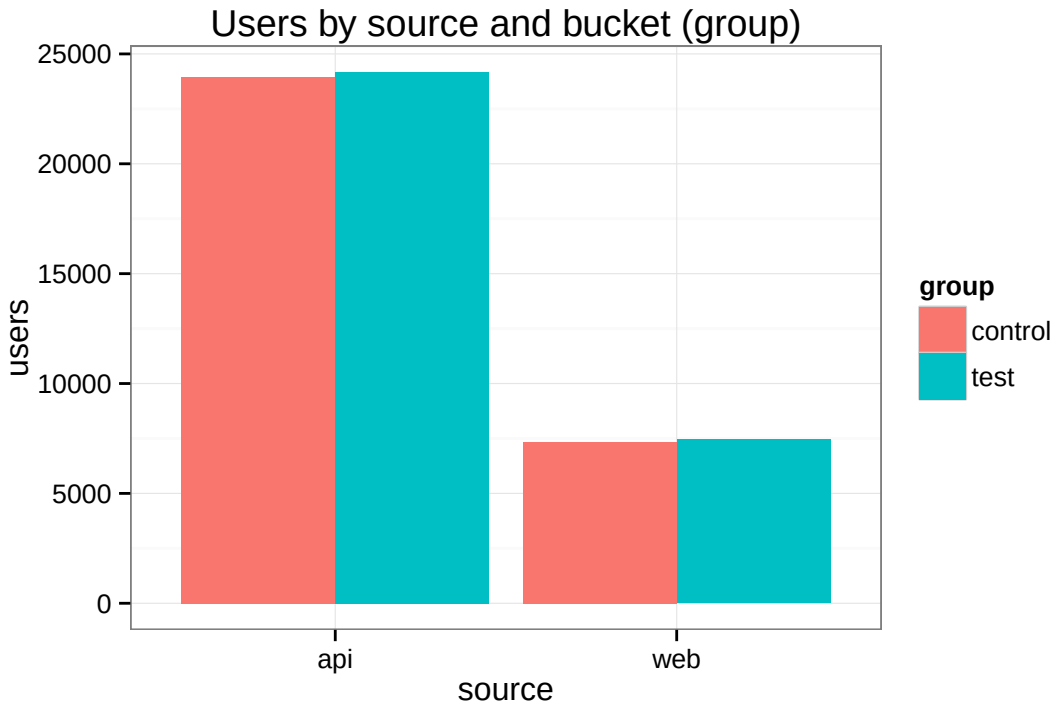
```
##       [,1]    [,2]
## PD -0.0634 -0.0482
## RR  0.8913  0.9157
```

The proportion of API queries in the control bucket is 4.82-6.34% lesser than the proporton of web queries. API queries are 0.89-0.91 times less likely to end up in the 'test group' bucket than web queries. This is *NOT* to say that queries are more or less likely to end up in one bucket or the other, but that queries from one source are more/less likely to end up in a particular bucket than queries from another source.

Users by Source and Group

One hypothesis is that API users send a lot more queries than web users do, so if a few very active API users are selected to be in the test, then the bucket they are (pseudo-randomly but essentially deterministically) assigned to will impact how many queries end up in the bucket.

Let us examine how source and bucketing look like when we focus on users rather than queries.



From this figure it's pretty clear that user bucketing is even across the two sources. We will again use the Bayes Factor to perform a Bayesian test of indendence.

```
##       group
## source control  test
##    api   0.498 0.502
##    web   0.496 0.504
```

```
## $`Bayes Factor`
## [1] 0.0135
##
## $Interpretation
## [1] "Not worth more than a bare mention."
```

```
##        [,1]   [,2]
## PD -0.00759 0.0108
## RR  0.98491 1.0222
```

There we have it – the test of independence says we don't have evidence of association, the proportion difference credible interval includes 0, and neither source is more or less likely than the other to have more in any one particular bucket. Clearly, the disparity/bias is coming from heavy users. Let's take a look at the volume of queries per user (Q.P.U).

source	group	Users	Total Queries	Maximum Q.P.U.	Median / Average	99th Percentile Q.P.U.
api	control	23970	39524	1188	1.000 / 1.649	5
api	test	24183	36223	819	1.000 / 1.498	5
web	control	7348	11992	3995	1.000 / 1.632	3
web	test	7458	8770	230	1.000 / 1.176	3

The buckets are actually not to so disparate once we remove the web user in the control group that's an outlier with 3,995 queries: 11,992 - 3,995 = 7,997 queries in control group vs 8,770 queries in test group – which brings the buckets a lot more closer to each other, and more consistent with what Oliver saw in his analysis, wherein the buckets were even within the web users but disparate within the API users.

Top users by contribution to their respective bucket

We computed the contribution weight of each user to their bucket and picked out the top 10 users within each bucket.

group	source	queries per user	weight
control	web	3995	0.078
control	api	1188	0.023
control	api	1087	0.021
control	api	696	0.014
control	api	646	0.013
control	api	543	0.011
control	api	495	0.010
control	api	312	0.006
control	api	231	0.004
control	api	209	0.004
test	api	819	0.018
test	api	640	0.014
test	api	639	0.014
test	api	461	0.010
test	web	230	0.005
test	api	210	0.005
test	api	203	0.005
test	api	201	0.004
test	api	184	0.004

group	source	queries per user	weight
test	api	177	0.004

To little surprise, the list was dominated by API users. There is an outlier web user with 3995 queries to their name. We were curious whether there existed a relationship between bucketing and source among the users with the most contribution, so we picked top 20 users and employed Fisher's Exact Test to test the independence.

```
##          source
## group     api web
##   control  20   1
##   test     15   5
```

```
##
##  Fisher's Exact Test for Count Data
##
## data:  .
## p-value = 0.09
## alternative hypothesis: true odds ratio is not equal to 1
## 95 percent confidence interval:
##    0.621 330.093
## sample estimates:
## odds ratio
##       6.39
```

We failed to reject the hypothesis ($p = 0.09$) that source is independent of bucketing. We also computed the contribution weight of each user to their source group and picked out the top 10 users within each source category.

Top users by contribution to their respective source group

source	group	queries per user	weight
api	control	1188	0.016
api	control	1087	0.014
api	test	819	0.011
api	control	696	0.009
api	control	646	0.009
api	test	640	0.008
api	test	639	0.008
api	control	543	0.007
api	control	495	0.007
api	test	461	0.006
web	control	3995	0.192
web	test	230	0.011
web	test	147	0.007
web	test	113	0.005
web	test	85	0.004
web	test	83	0.004
web	control	51	0.002
web	control	36	0.002

source	group	queries per user	weight
web	test	32	0.002
web	test	26	0.001

It almost looked like the bucketing was actually kind of even among the top 10 users. So we decided to confirm this with Fisher's Exact Test for the top 20 most contributing users within each source category.

```
##       group
## source control test
##    api      12    8
##    web       7   13
```

```
##
##  Fisher's Exact Test for Count Data
##
## data:  .
## p-value = 0.2
## alternative hypothesis: true odds ratio is not equal to 1
## 95 percent confidence interval:
##   0.653 12.203
## sample estimates:
## odds ratio
##       2.71
```

We fail to reject the hypothesis ($p = 0.2$) that bucketing is independent of source among the top 20 API users and top 20 web users.