CSE 100 Project

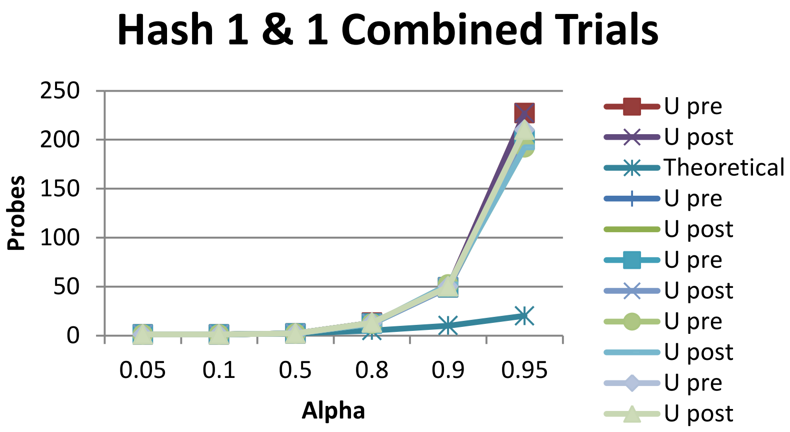
Vibhor Jain
Lawrence Lam
Robert Wang

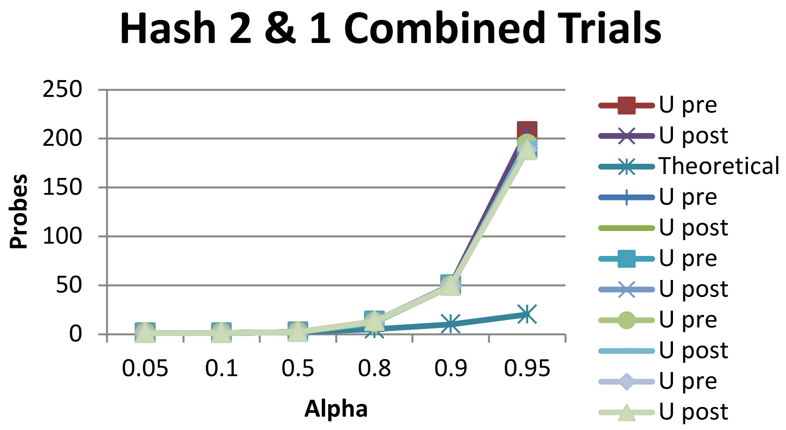
Random Keys

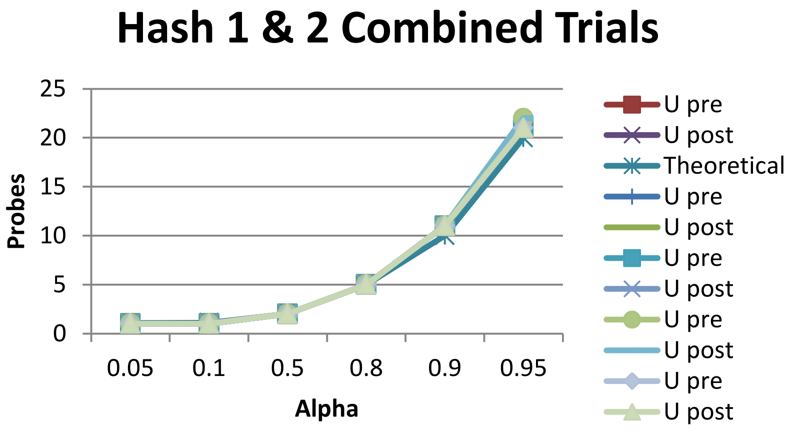
- Using a hash table with open addressing, we used several hash function combinations and alpha values to generate, hash, and measure the runtimes of searching for keys
- Collected data across 5 replicates

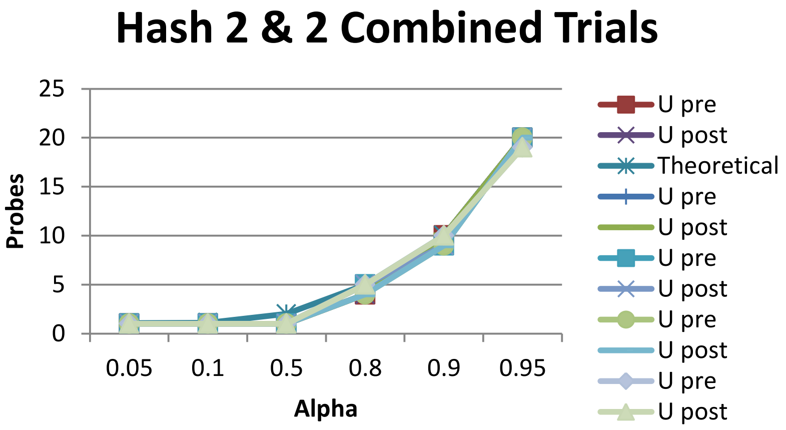
Hash Functions

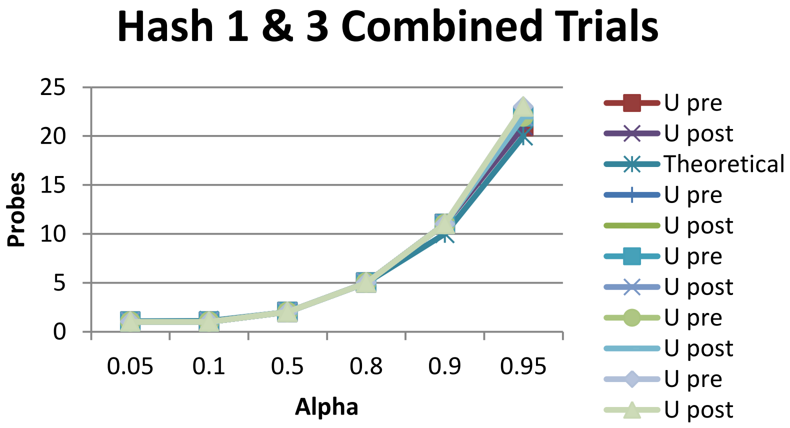
- Two types were used in combination
- Primary (1 & 2)
  - Division method: $H1(k) = k \bmod m$
  - Multiplication method: $H1(k) = \text{floor}(m(kA – \text{floor}(kA))$
    - $A = (\text{sqrt}(5) – 1)/2$
- Secondary (1, 2, & 3)
  - Linear probing: $H2(k) = 1$
  - Double hashing:
    - $H2(k) = 2k+1$
    - $H2(k) = 2k$

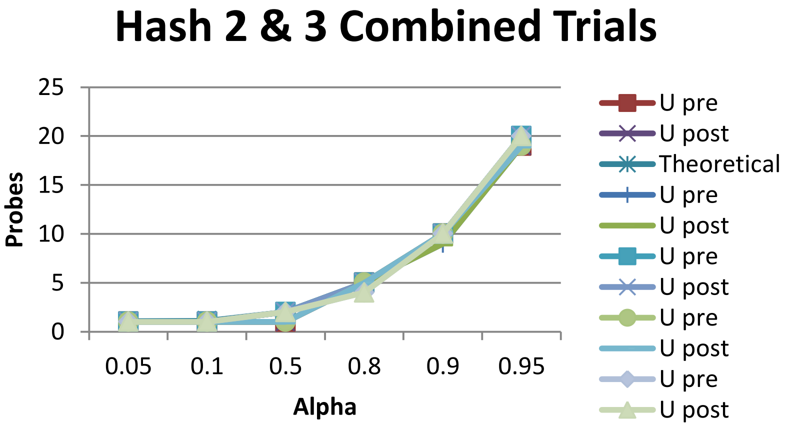

Hash 1 & 1 Combined Trials


Hash 2 & 1 Combined Trials


Hash 1 & 2 Combined Trials

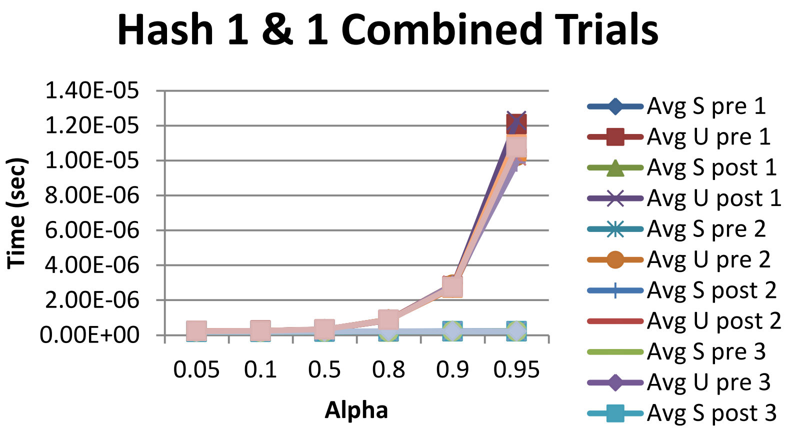

Hash 2 & 2 Combined Trials

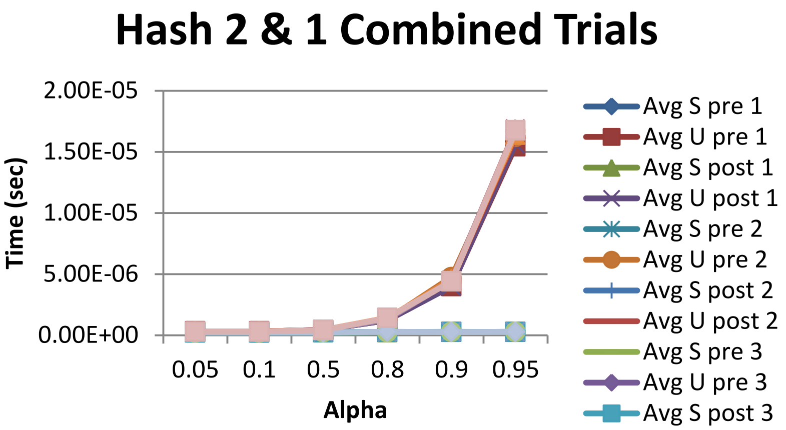

Hash 1 & 3 Combined Trials

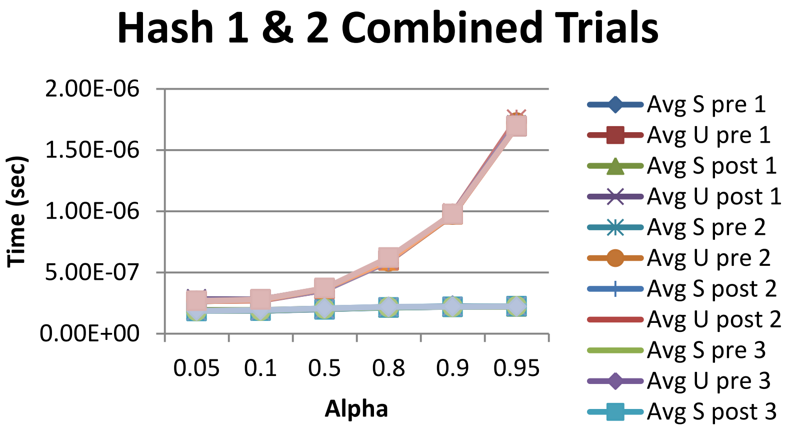

Hash 2 & 3 Combined Trials

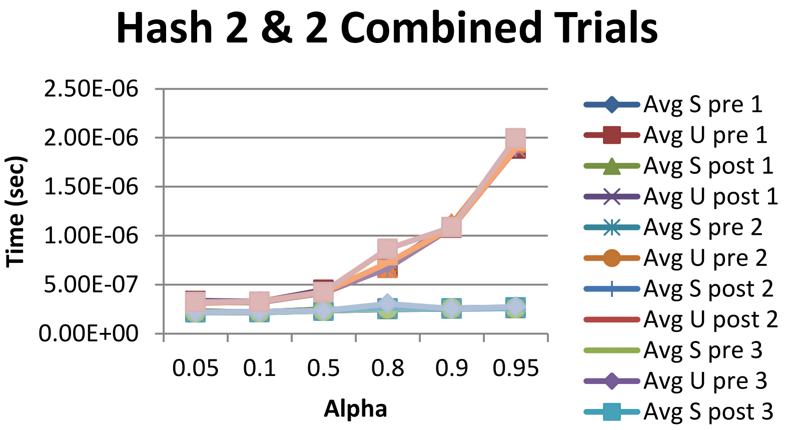
Probe Results

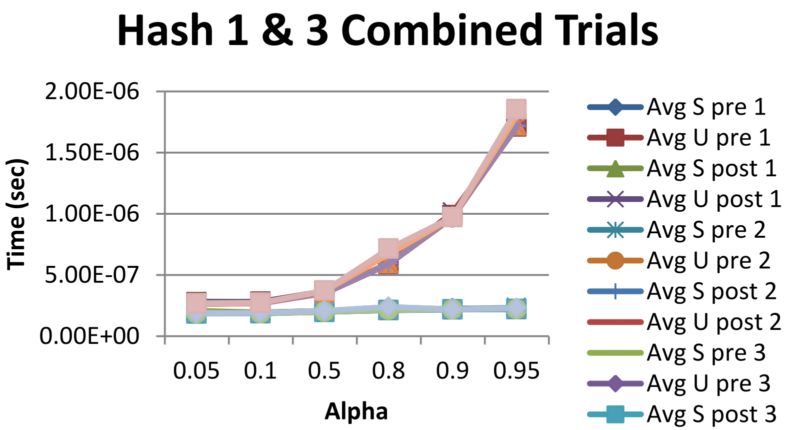
- With a load factor of alpha = n/m, the expected number of probes to determine whether or not a key is in the hash table is 1/(1-alpha) [Thm. 11.6]
- With most of the combinations, we find that the experimental results fit very well with the theoretical estimates
- However, double hashing with the use of linear probing results in an extremely longer search period
- This is most likely because the distribution of the keys is not truly uniform, which is an assumption of Thm. 11.6

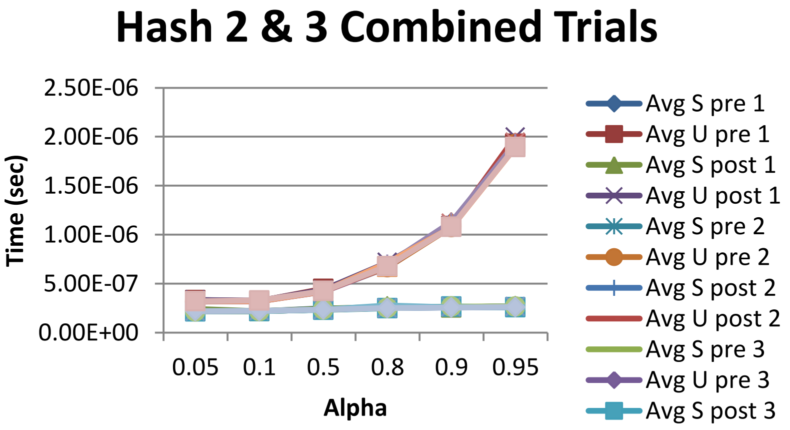

Hash 1 & 1 Combined Trials
Time (sec)
1.40E-05
1.20E-05
1.00E-05
8.00E-06
6.00E-06
4.00E-06
2.00E-06
0.00E+00
0.05
0.1
0.5
0.8
0.9
0.95
Alpha
Avg S pre 1
Avg U pre 1
Avg S post 1
Avg U post 1
Avg S pre 2
Avg U pre 2
Avg S post 2
Avg U post 2
Avg S pre 3
Avg U pre 3
Avg S post 3


Hash 2 & 1 Combined Trials
Time (sec)
2.00E-05
1.50E-05
1.00E-05
5.00E-06
0.00E+00
0.05
0.1
0.5
0.8
0.9
0.95
Alpha
Avg S pre 1
Avg U pre 1
Avg S post 1
Avg U post 1
Avg S pre 2
Avg U pre 2
Avg S post 2
Avg U post 2
Avg S pre 3
Avg U pre 3
Avg S post 3


Hash 1 & 2 Combined Trials
Time (sec)
2.00E-06
1.50E-06
1.00E-06
5.00E-07
0.00E+00
0.05
0.1
0.5
0.8
0.9
0.95
Alpha
Avg S pre 1
Avg U pre 1
Avg S post 1
Avg U post 1
Avg S pre 2
Avg U pre 2
Avg S post 2
Avg U post 2
Avg S pre 3
Avg U pre 3
Avg S post 3

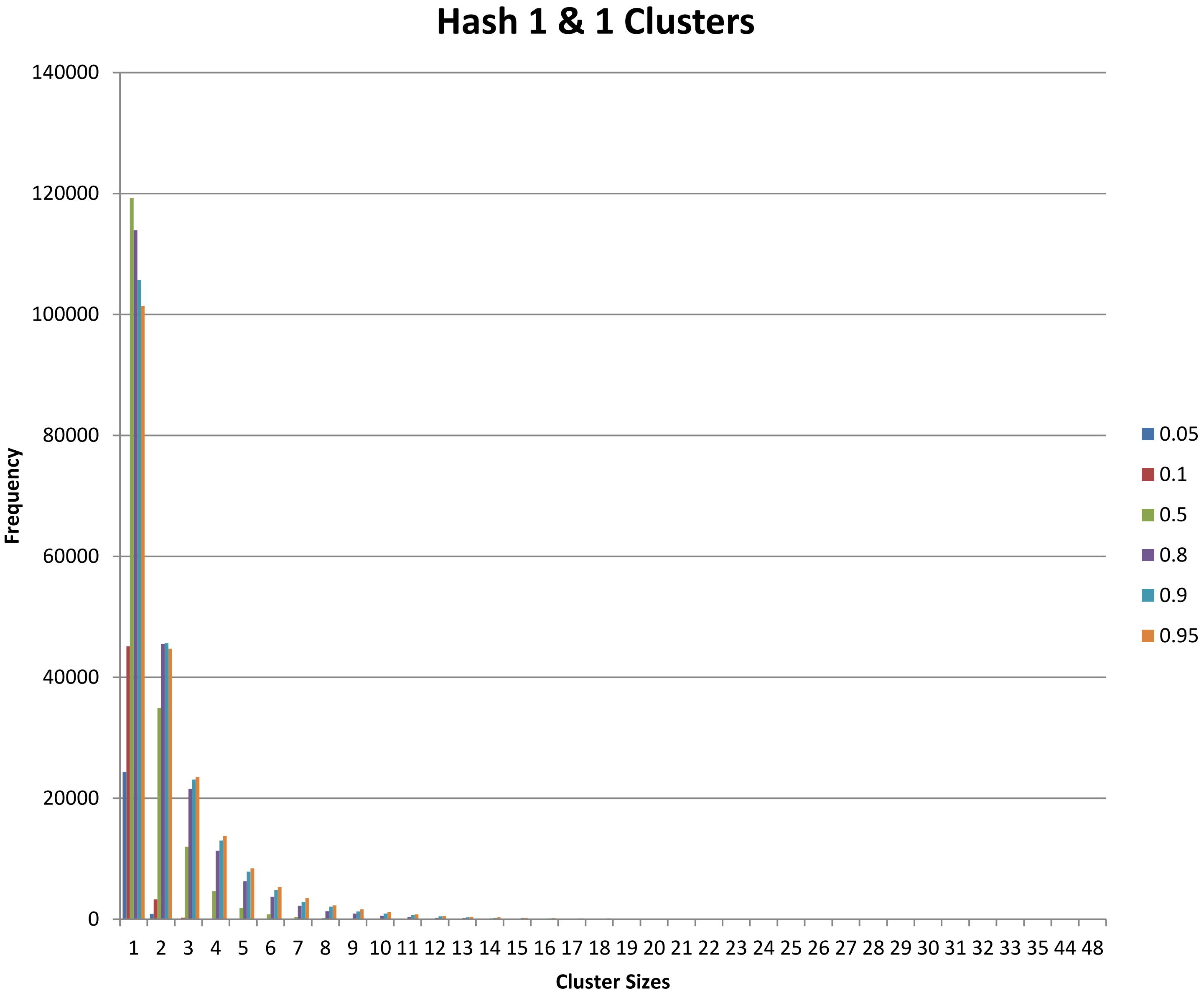

Hash 2 & 2 Combined Trials
Time (sec)
2.50E-06
2.00E-06
1.50E-06
1.00E-06
5.00E-07
0.00E+00
0.05
0.1
0.5
0.8
0.9
0.95
Alpha
Avg S pre 1
Avg U pre 1
Avg S post 1
Avg U post 1
Avg S pre 2
Avg U pre 2
Avg S post 2
Avg U post 2
Avg S pre 3
Avg U pre 3
Avg S post 3

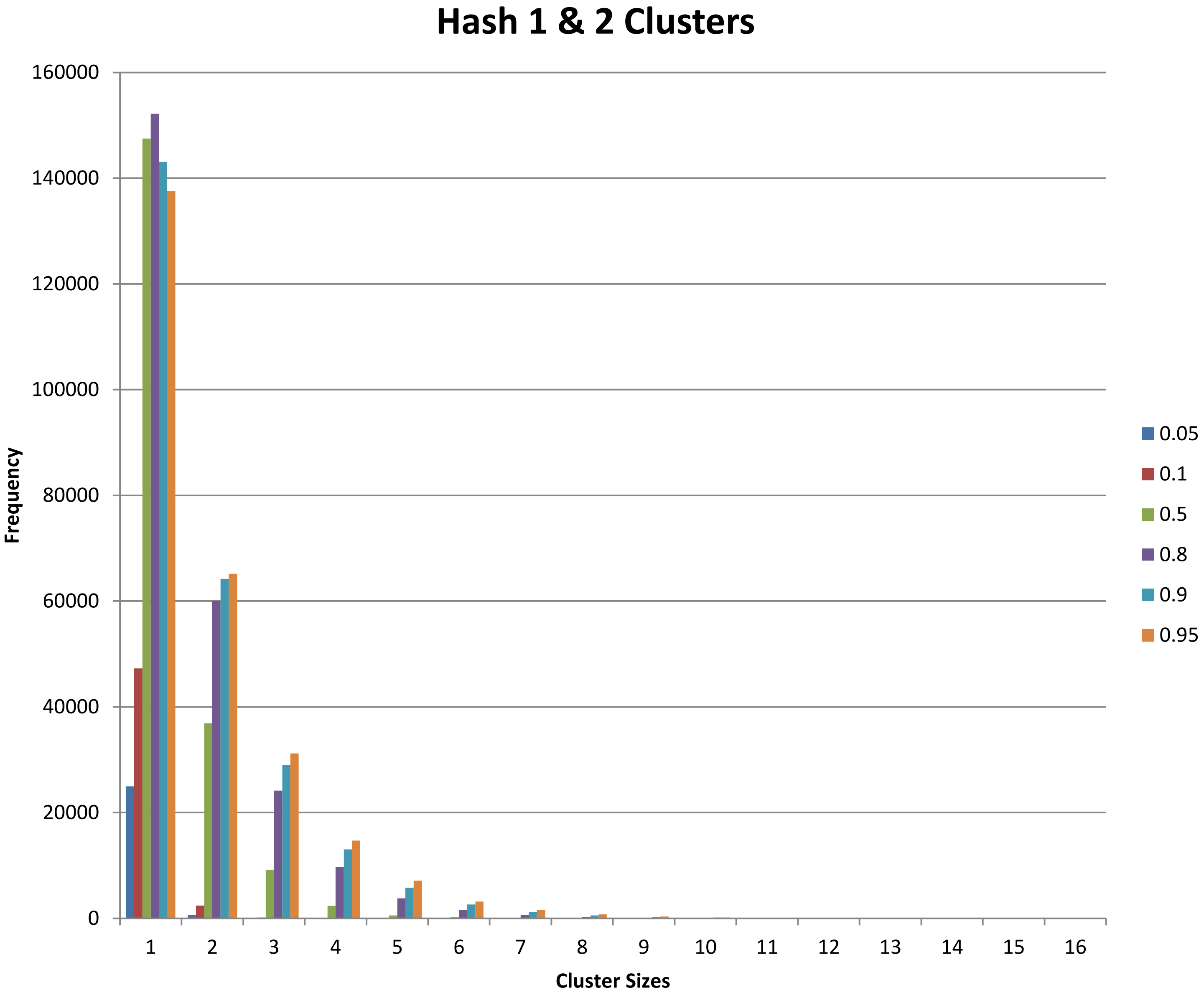

Hash 1 & 3 Combined Trials
Time (sec)
2.00E-06
1.50E-06
1.00E-06
5.00E-07
0.00E+00
0.05
0.1
0.5
0.8
0.9
0.95
Alpha
Avg S pre 1
Avg U pre 1
Avg S post 1
Avg U post 1
Avg S pre 2
Avg U pre 2
Avg S post 2
Avg U post 2
Avg S pre 3
Avg U pre 3
Avg S post 3

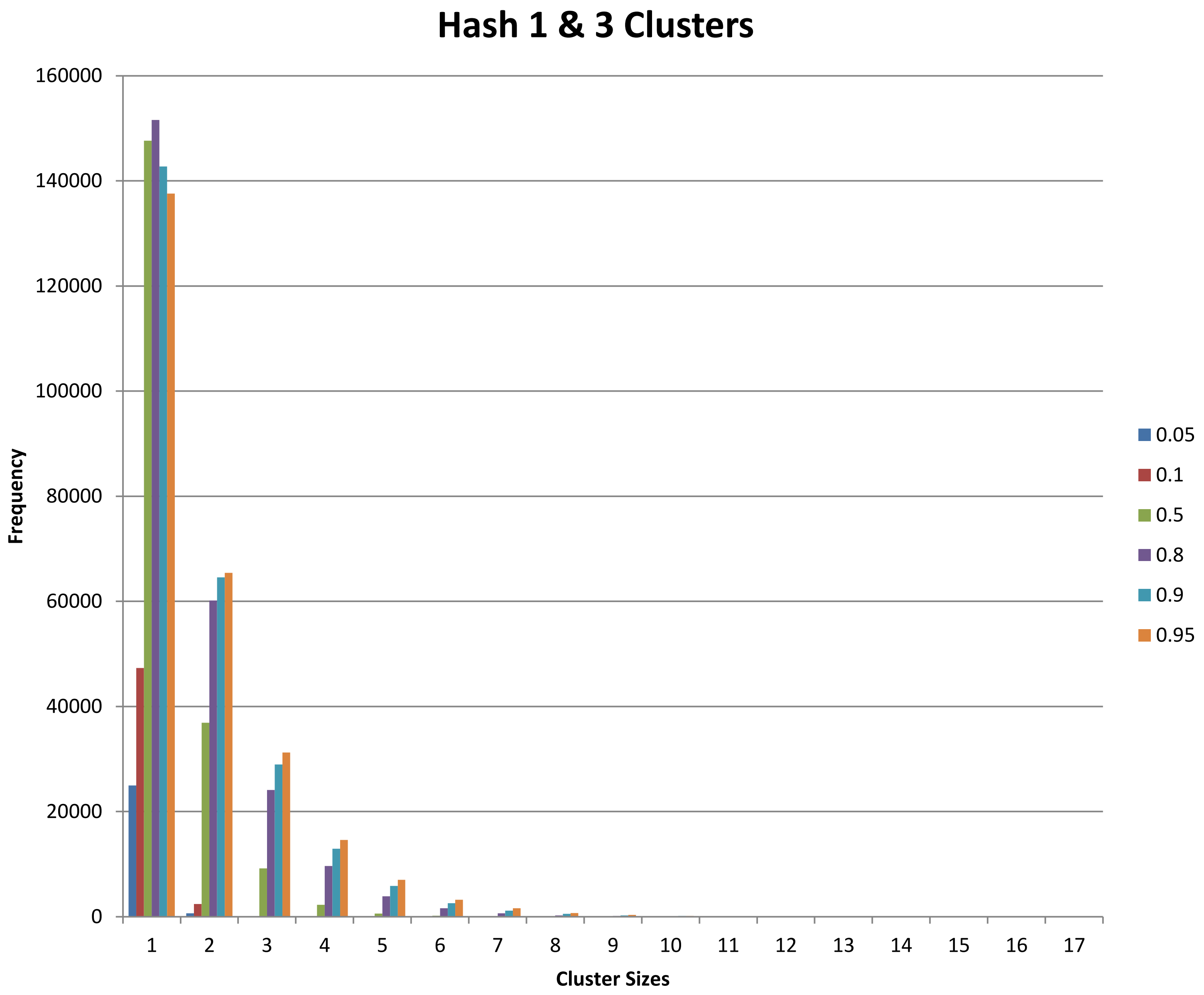

Hash 2 & 3 Combined Trials
Time (sec)
2.50E-06
2.00E-06
1.50E-06
1.00E-06
5.00E-07
0.00E+00
0.05
0.1
0.5
0.8
0.9
0.95
Alpha
Avg S pre 1
Avg U pre 1
Avg S post 1
Avg U post 1
Avg S pre 2
Avg U pre 2
Avg S post 2
Avg U post 2
Avg S pre 3
Avg U pre 3
Avg S post 3

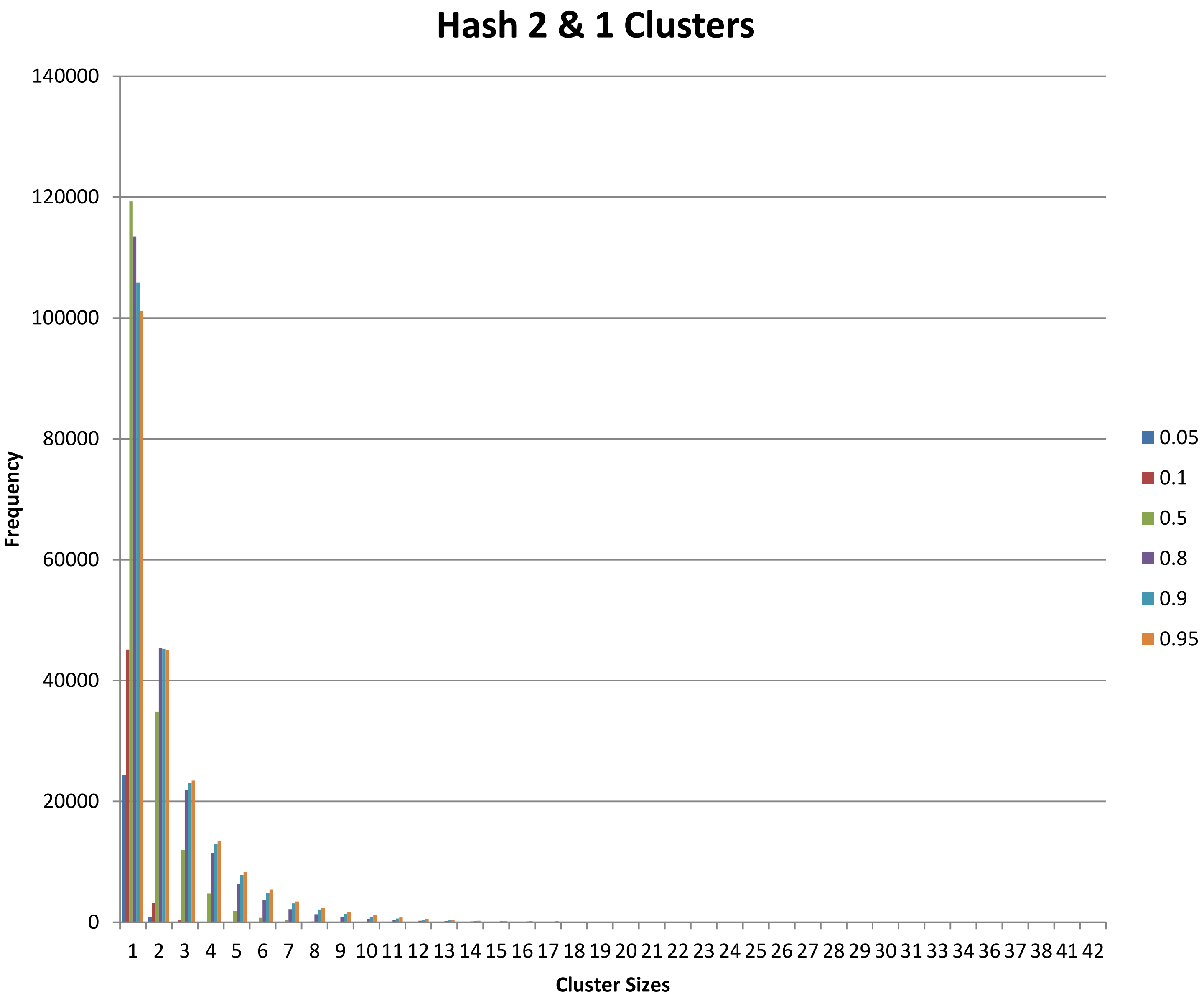
Runtime Results

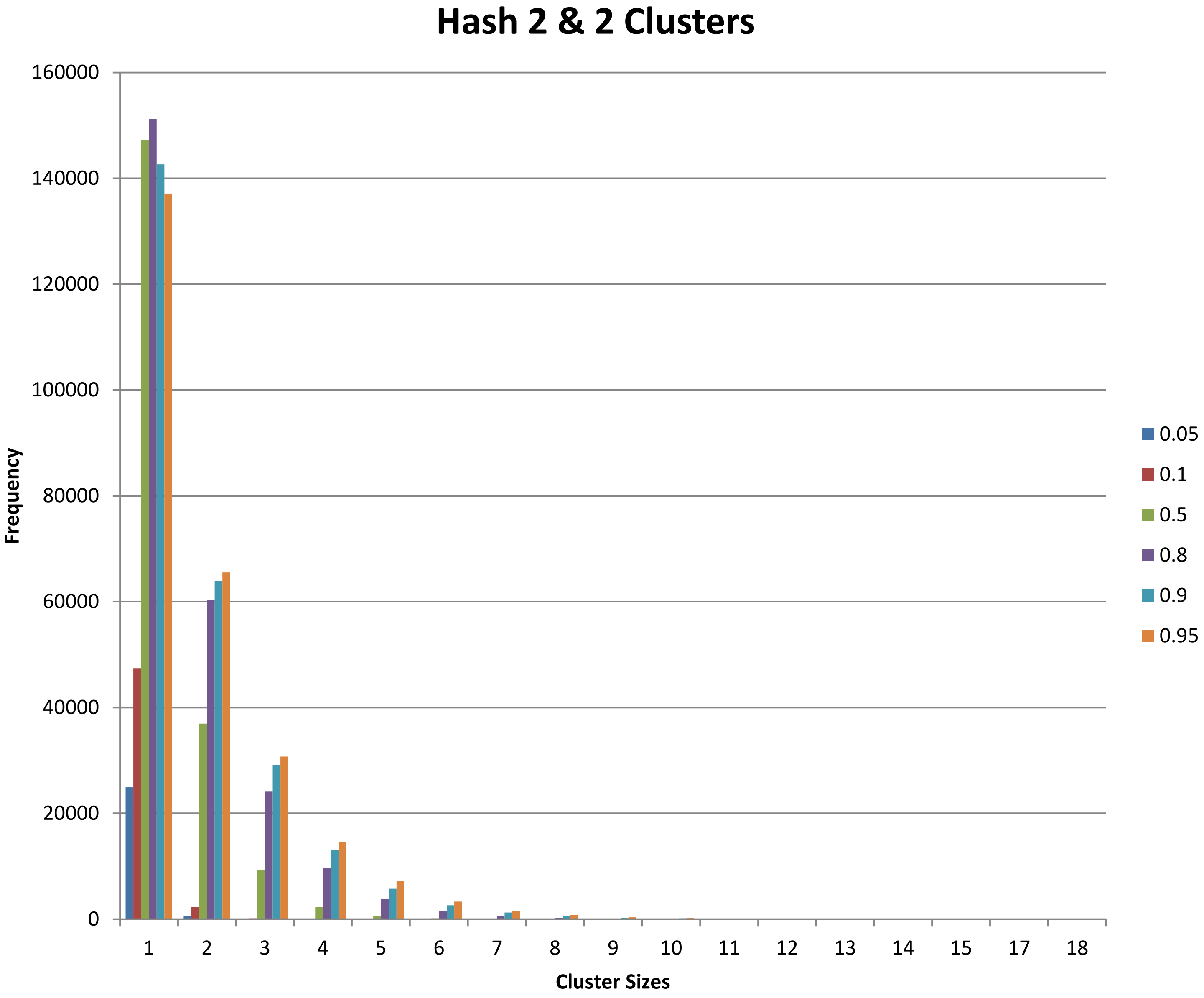
- On average, successful searches tend to be roughly an order of magnitude faster than unsuccessful searches
- As the alpha values increase more slots in the hash table are filled, leading to more collisions
- This requires more probes for each unsuccessful search and causes the runtime to scale exponentially
- Also like before, we notice that linear probing causes the runtime to increase by another order of magnitude
- Regardless of whether or not we do deletions, the runtimes do not appear to differ any significant amount

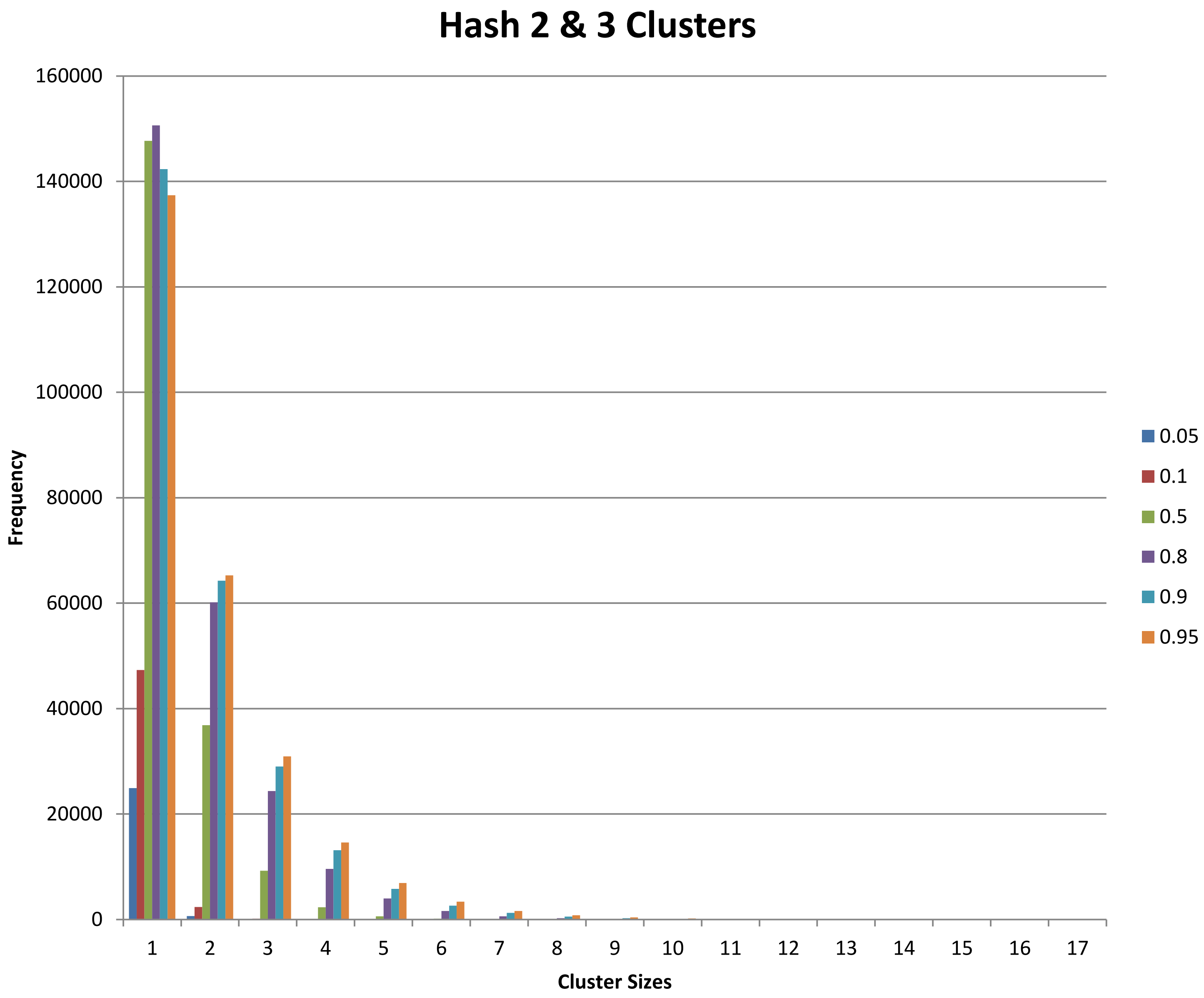

Hash 1 & 1 Clusters
Frequency
140000
120000
100000
80000
60000
40000
20000
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
35
44
48
Cluster Sizes
0.05
0.1
0.5
0.8
0.9
0.95


Hash 1 & 2 Clusters
160000
140000
120000
100000
80000
60000
40000
20000
0
Frequency
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
Cluster Sizes
0.05
0.1
0.5
0.8
0.9
0.95


Hash 1 & 3 Clusters
Frequency
160000
140000
120000
100000
80000
60000
40000
20000
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
Cluster Sizes
0.05
0.1
0.5
0.8
0.9
0.95


Hash 2 & 1 Clusters
Frequency
140000
120000
100000
80000
60000
40000
20000
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 33 34 36 37 38 41 42
Cluster Sizes
0.05
0.1
0.5
0.8
0.9
0.95


Hash 2 & 2 Clusters
Frequency
160000
140000
120000
100000
80000
60000
40000
20000
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
17
18
Cluster Sizes
0.05
0.1
0.5
0.8
0.9
0.95


Hash 2 & 3 Clusters
Frequency
160000
140000
120000
100000
80000
60000
40000
20000
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
Cluster Sizes
0.05
0.1
0.5
0.8
0.9
0.95

Cluster Results

- Overall the cluster distributions appear to be the same between the secondary hash functions 2 and 3
- Linear probing causes everything to sort of bunch up together with large clusters, preventing a nice uniform distribution of the keys
- These results seem to correlate with our other results for probing and runtimes

Additional Work

- Data for statistics like min, max, and standard deviations are included in text files for each replicate
- Using non-randomly generated keys i.e. increasing or decreasing order causes linear probing to take an enormously large amount of time per key
- This is due to the fact that everything is bunched up together and thus the number of keys that need to be probed after the first collision increases exponentially as the alpha grows
- Runtimes for non-random keys increases by several magnitudes of order, going from ~10 microseconds at worst to >0.01 seconds per key

Specs

- Experiments were run on engapps
- CPU:
  - Intel(R) Xeon(R) CPU E7- 4830 @ 2.13GHz
  - CPU MHz 1067.000
  - 1 processor
  - 24576 KB cache

Roles

- Vibhor
  - Rand & non-rand key generation, analysis
- Lawrence
  - Hash functions, cluster lengths, chaining implementation, analysis
- Robert
  - Main code, hashtable header & implementation, data collection & graphs, analysis