Randomized load balancing, caching and Big-O math

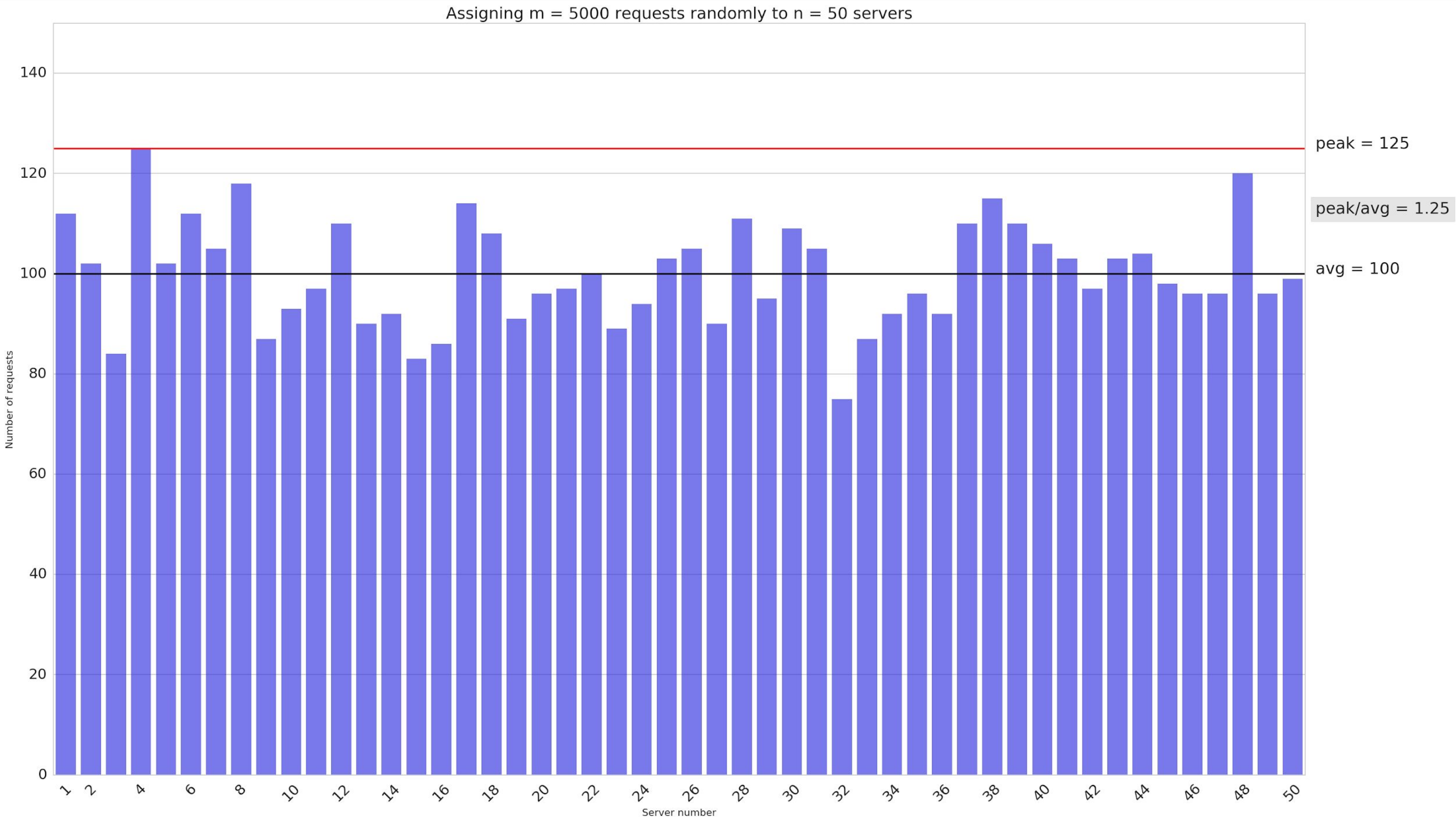
Julius Plenz <plenz@google.com> June 2018

A necessary disclaimer about explaining maths live and with slides

Balls into bins

$\Leftrightarrow$

Requests into servers


Assigning m = 5000 requests randomly to n = 50 servers
peak = 125
peak/avg = 1.25
avg = 100
Number of requests
Server number
0
20
40
60
80
100
120
140
1
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40
42
44
46
48
50

Peak-to-average load is important

- Need to **provision** resources for peak usage
- Small peak-to-average load means **lower cost**
- Goal: Make it **small** and **predictable**!

Can we predict the peak load? *)

*) with high probability

Theorem 1. *Let M be the random variable that counts the maximum number of balls in any bin, if we throw m balls independently and uniformly at random into n bins. Then $\Pr[M > k_\alpha] = o(1)$ if $\alpha > 1$ and $\Pr[M > k_\alpha] = 1 - o(1)$ if $0 < \alpha < 1$, where*

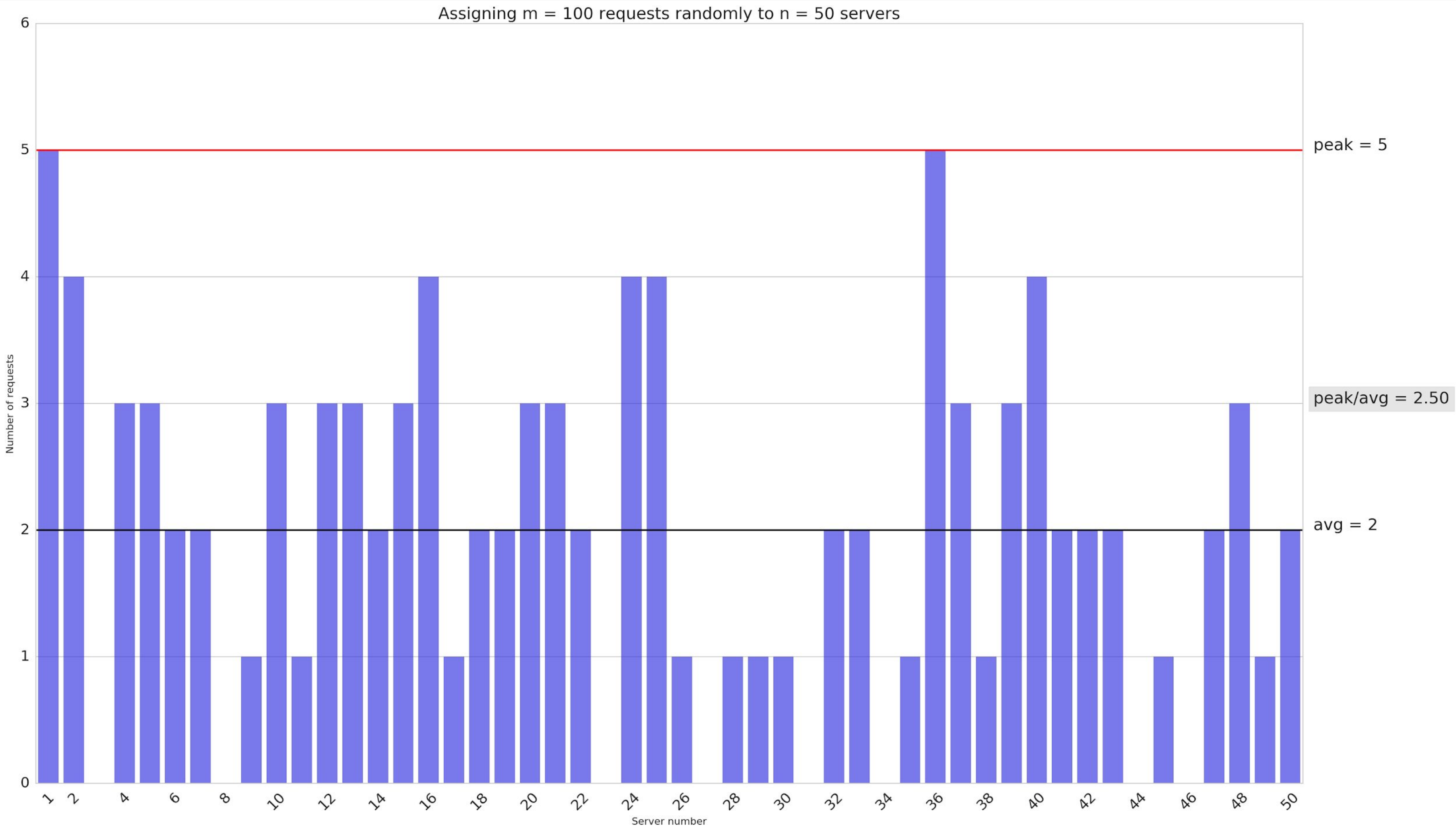
$$k_\alpha = \begin{cases} \frac{\log n}{\log \frac{n \log n}{m}} \left(1 + \alpha \frac{\log^{(2)} \frac{n \log n}{m}}{\log \frac{n \log n}{m}}\right), & \text{if } \frac{n}{\text{polylog}(n)} \leq m \ll n \log n, \\ (d_c - 1 + \alpha) \log n, & \text{if } m = c \cdot n \log n \text{ for some constant } c, \\ \frac{m}{n} + \alpha \sqrt{2 \frac{m}{n} \log n}, & \text{if } n \log n \ll m \leq n \cdot \text{polylog}(n), \\ \frac{m}{n} + \sqrt{\frac{2m \log n}{n} \left(1 - \frac{1}{\alpha} \frac{\log^{(2)} n}{2 \log n}\right)}, & \text{if } m \gg n \cdot (\log n)^3. \end{cases}$$

Here d_c denotes a suitable constant depending only on c, cf. the proof of Lemma 3.

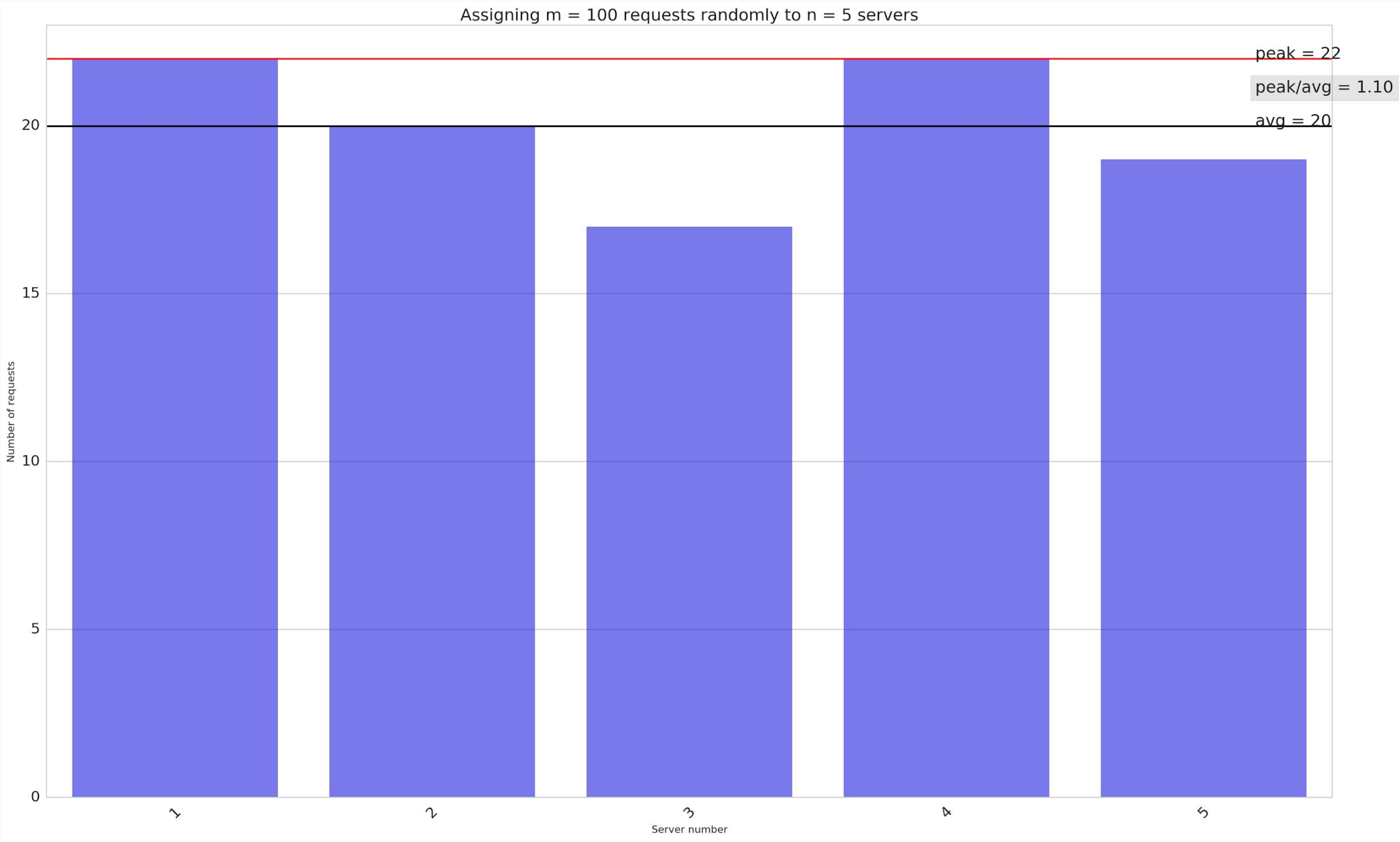
m: Requests n: Servers

"Balls into Bins" paper

Consequence #1: More requests per server are good



Assigning m = 100 requests randomly to n = 50 servers
peak = 5
peak/avg = 2.50
avg = 2
Number of requests
Server number
0
1
2
3
4
5
6
1
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40
42
44
46
48
50


Assigning m = 100 requests randomly to n = 5 servers
peak = 22
peak/avg = 1.10
avg = 20
Number of requests
0
5
10
15
20
1
2
3
4
5
Server number

(From “Balls into Bins”)

$$k_\alpha = \begin{cases} \frac{\log n}{\log \frac{n \log n}{m}} \left(1 + \alpha \frac{\log^{(2)} \frac{n \log n}{m}}{\log \frac{n \log n}{m}}\right), & \text{if } \frac{n}{\text{polylog}(n)} \le m \ll n \log n, \end{cases}$$

)

Peak-to-average ratio:

O(log *n*)

Oh no!

(Image source)

(From “Balls into Bins”)

$$k_\alpha = \begin{cases} (d_c - 1 + \alpha)\log n, & \text{if } m = c \cdot n \log n \text{ for some constant } c, \end{cases}$$

Peak-to-average ratio:

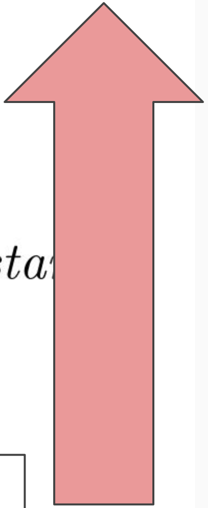
O(1)

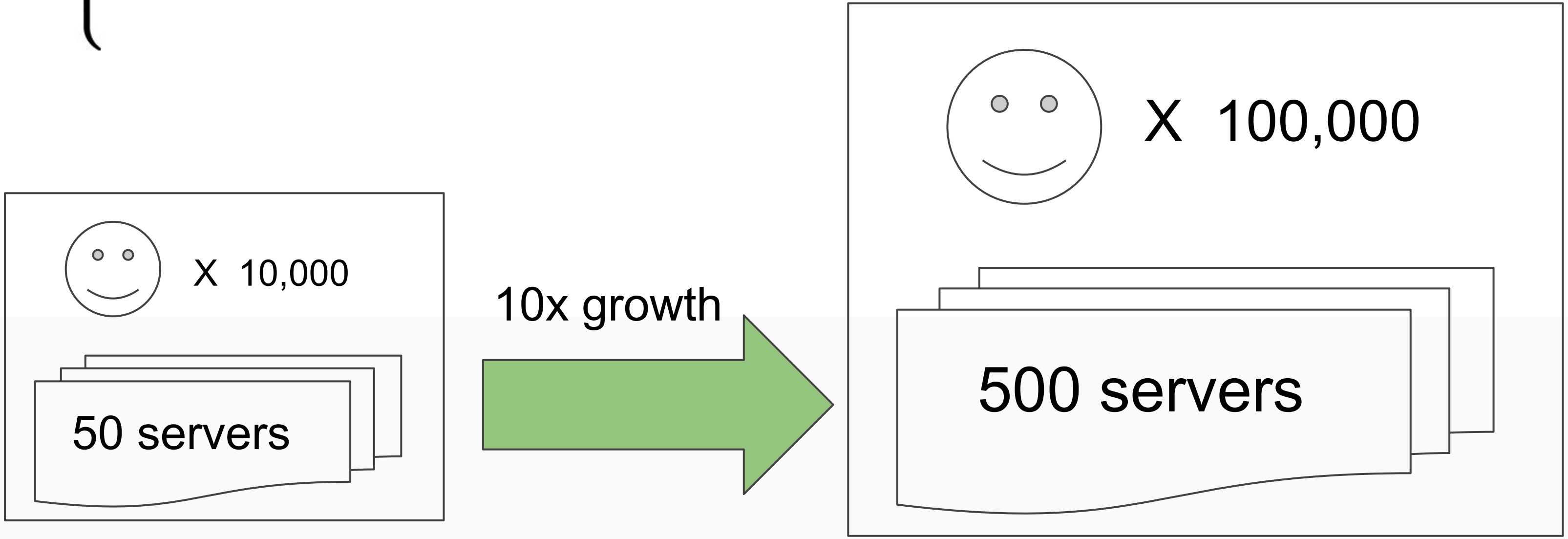
(Image source)

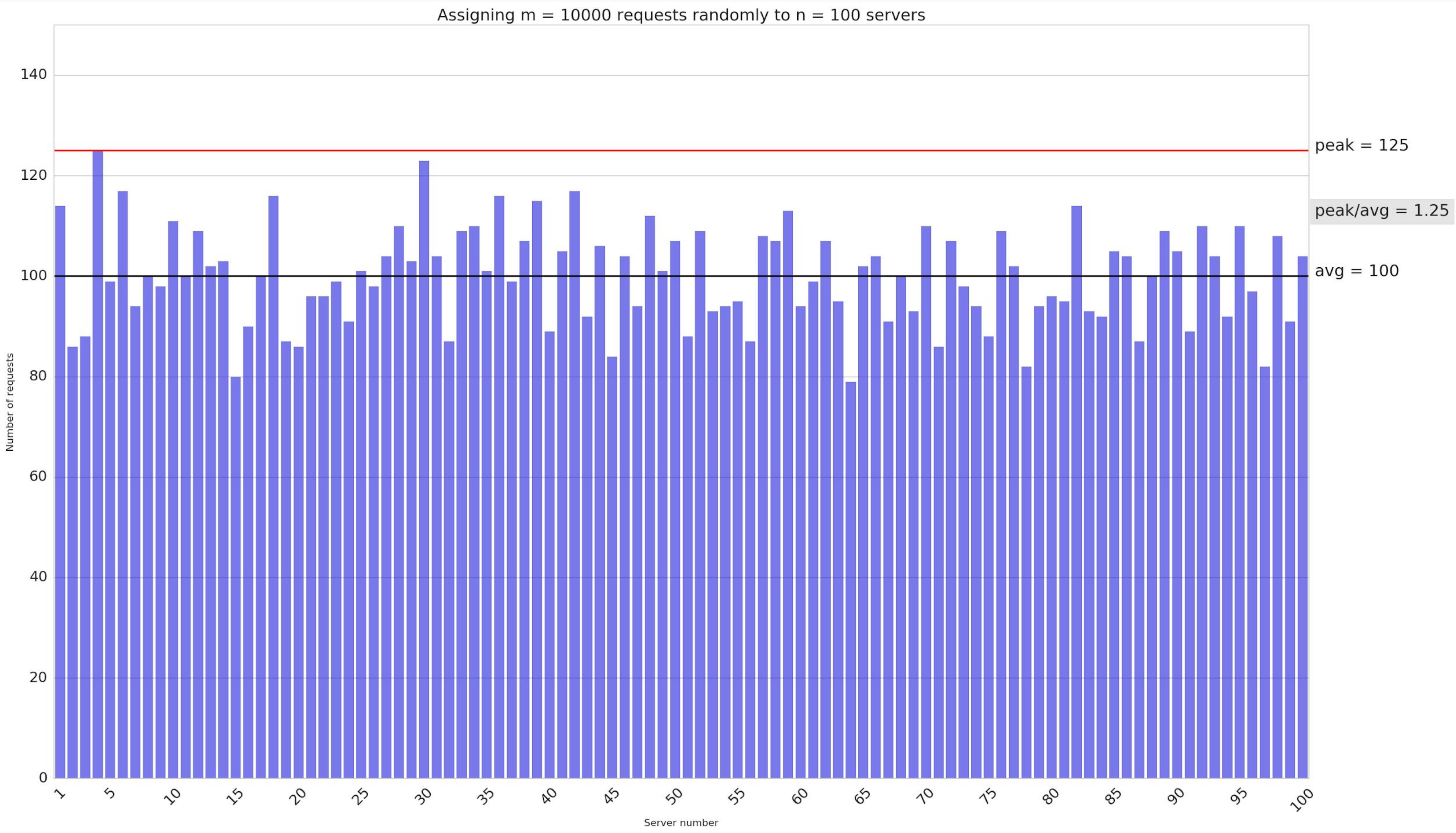
Consequence #2: Don’t scale servers and requests linearly 1:1

(From [“Balls into Bins”]())

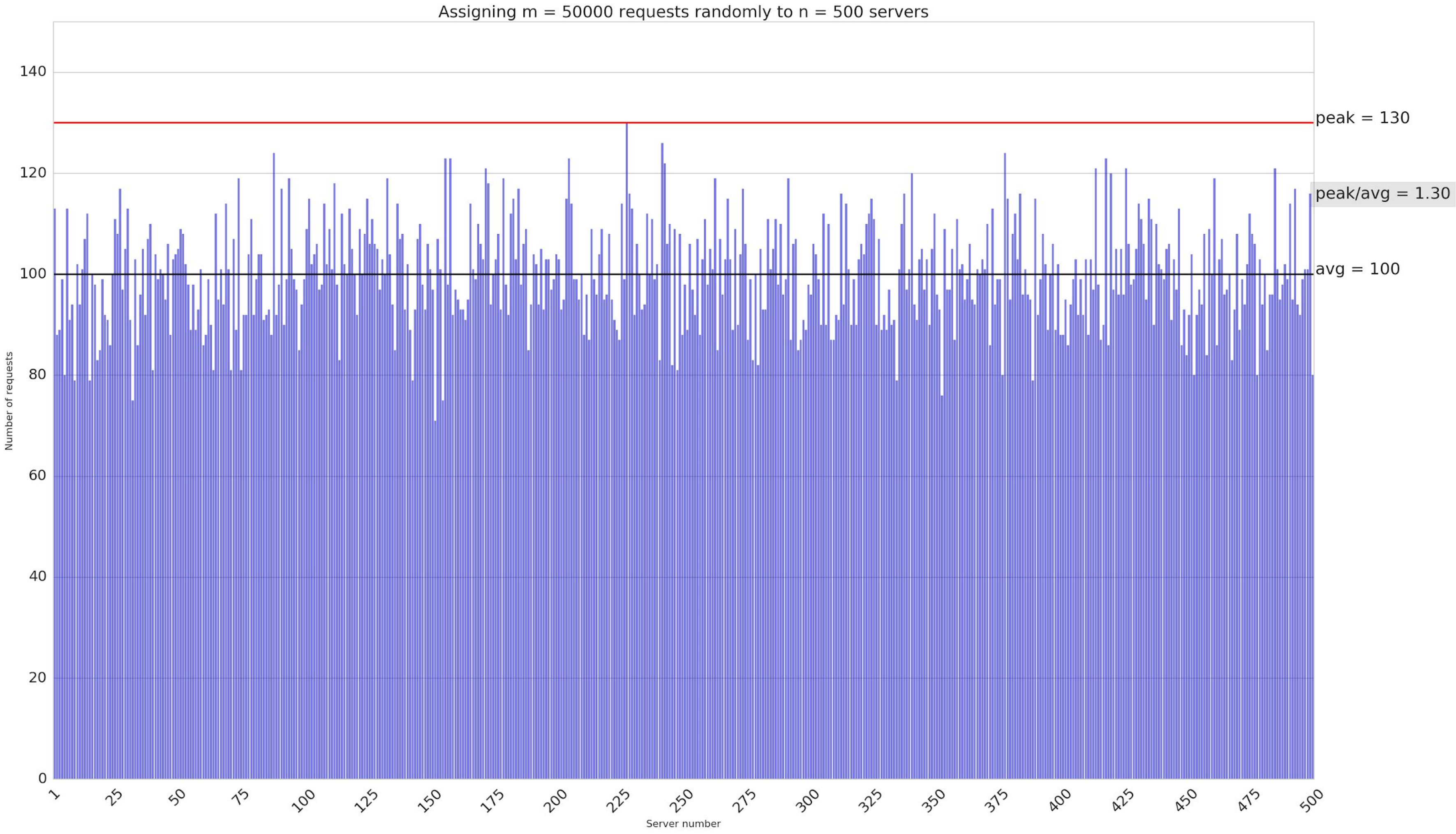
$$k_\alpha = \begin{cases} \dfrac{\log n}{\log \frac{n \log n}{m}} \left(1 + \alpha \dfrac{\log^{(2)} \frac{n \log n}{m}}{\log \frac{n \log n}{m}}\right), & \textit{if } \dfrac{n}{\mathrm{polylog}(n)} \leq m \ll n \log n, \\ (d_c - 1 + \alpha) \log n, & \textit{if } m = c \cdot n \log n \textit{ for some consta} \end{cases}$$

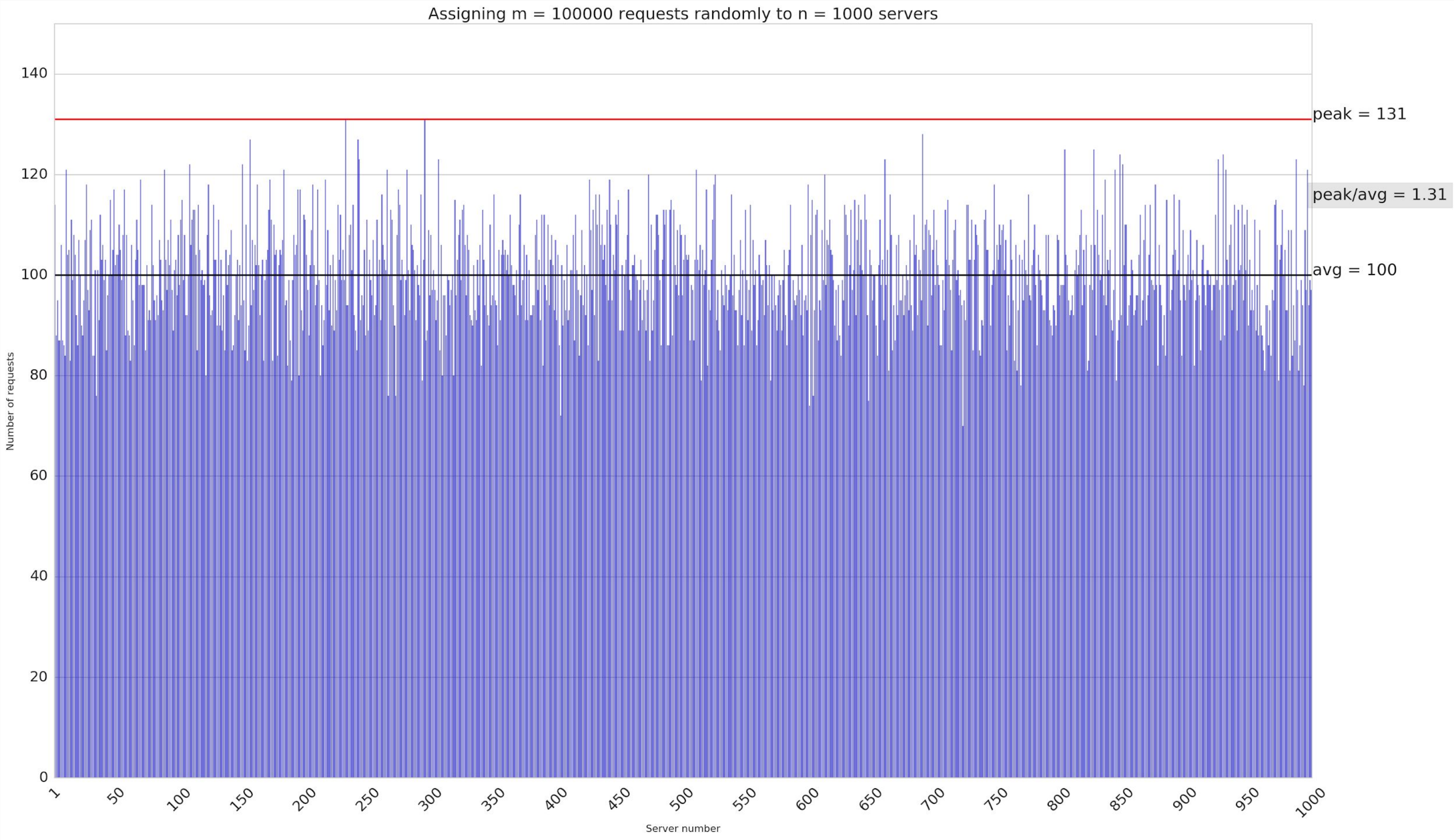






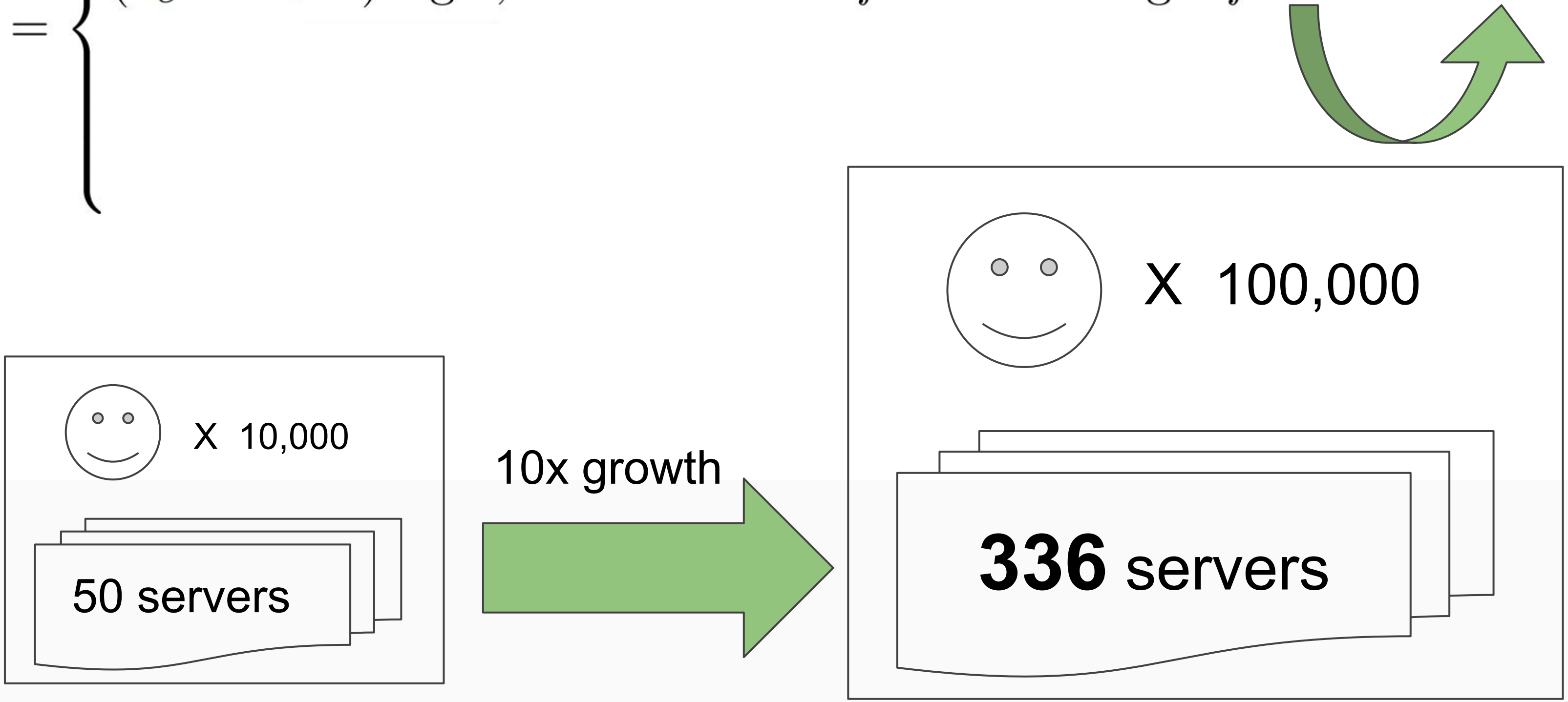
Assigning m = 10000 requests randomly to n = 100 servers
peak = 125
peak/avg = 1.25
avg = 100
Number of requests
Server number
0
20
40
60
80
100
120
140
1
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95
100


Assigning m = 50000 requests randomly to n = 500 servers
peak = 130
peak/avg = 1.30
avg = 100
Number of requests
Server number
0
20
40
60
80
100
120
140
1
25
50
75
100
125
150
175
200
225
250
275
300
325
350
375
400
425
450
475
500


Assigning m = 100000 requests randomly to n = 1000 servers
peak = 131
peak/avg = 1.31
avg = 100
Number of requests
Server number
0
20
40
60
80
100
120
140
1
50
100
150
200
250
300
350
400
450
500
550
600
650
700
750
800
850
900
950
1000

(From “Balls into Bins”)

$$k_\alpha = \begin{cases} \frac{\log n}{\log \frac{n \log n}{m}} \left(1 + \alpha \frac{\log^{(2)} \frac{n \log n}{m}}{\log \frac{n \log n}{m}}\right), & \text{if } \frac{n}{\text{polylog}(n)} \le m \ll n \log n, \\ (d_c - 1 + \alpha) \log n, & \text{if } m = c \cdot n \log n \text{ for some constant } c, \end{cases}$$

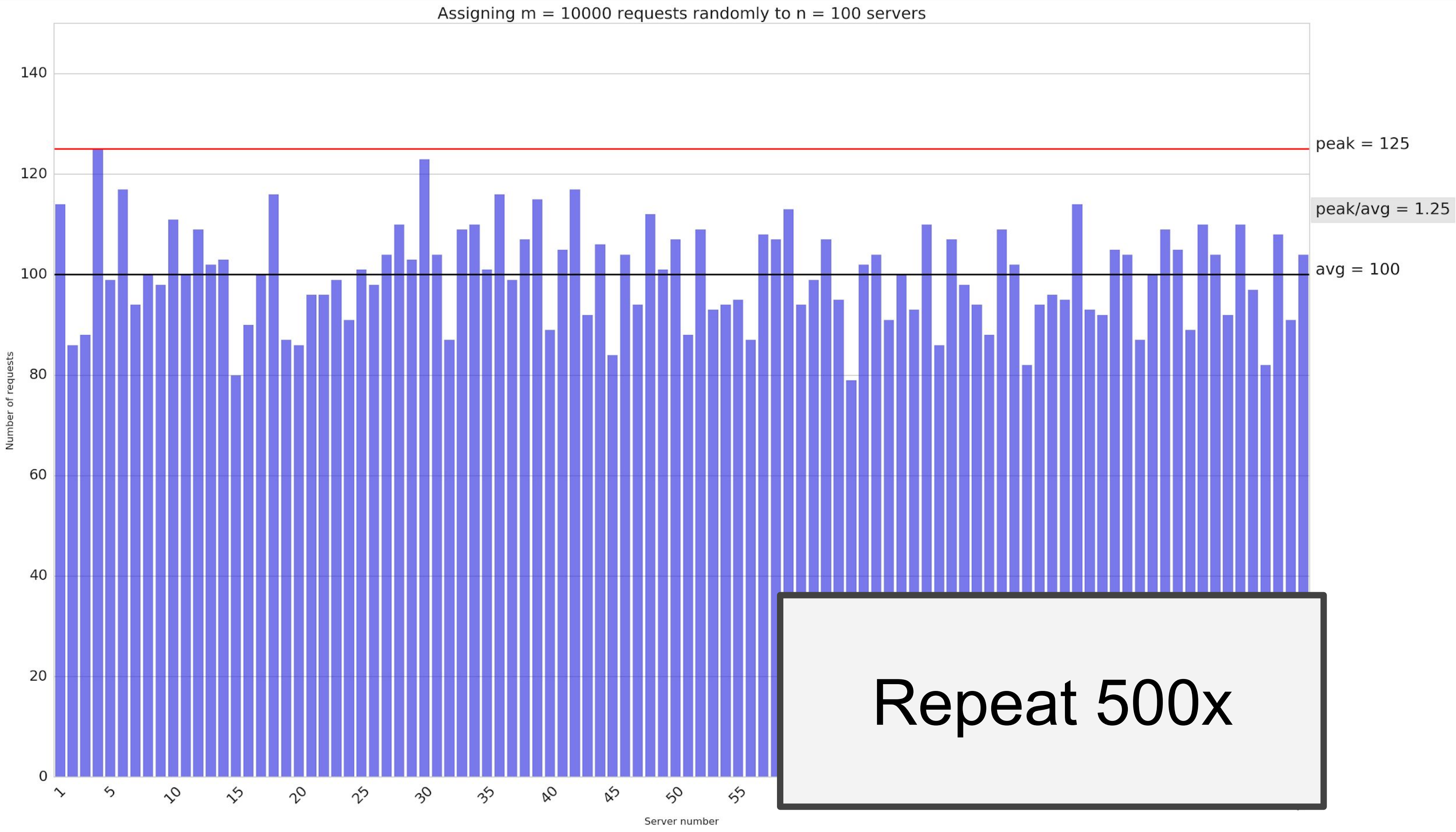


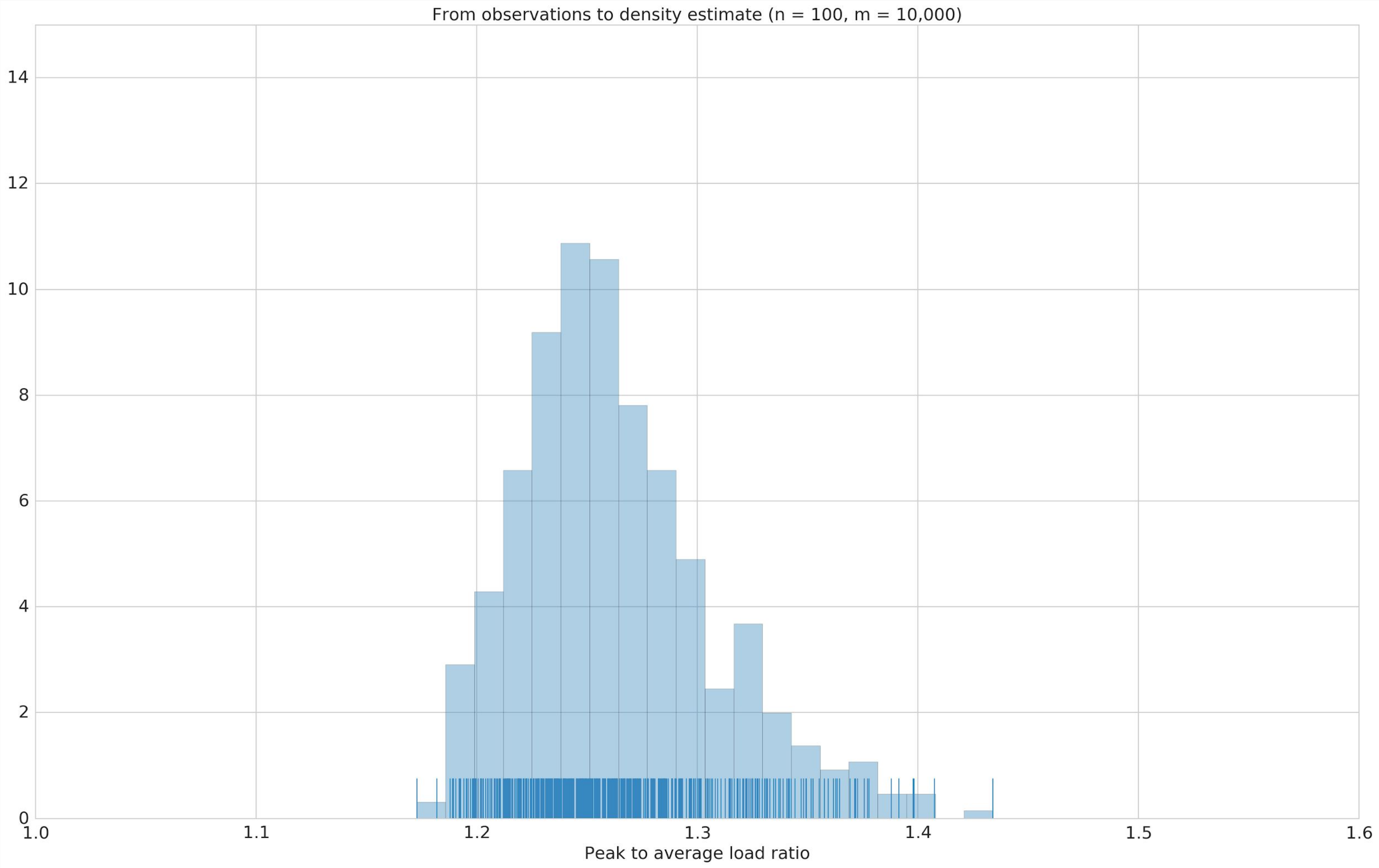
How can I calculate this myself?

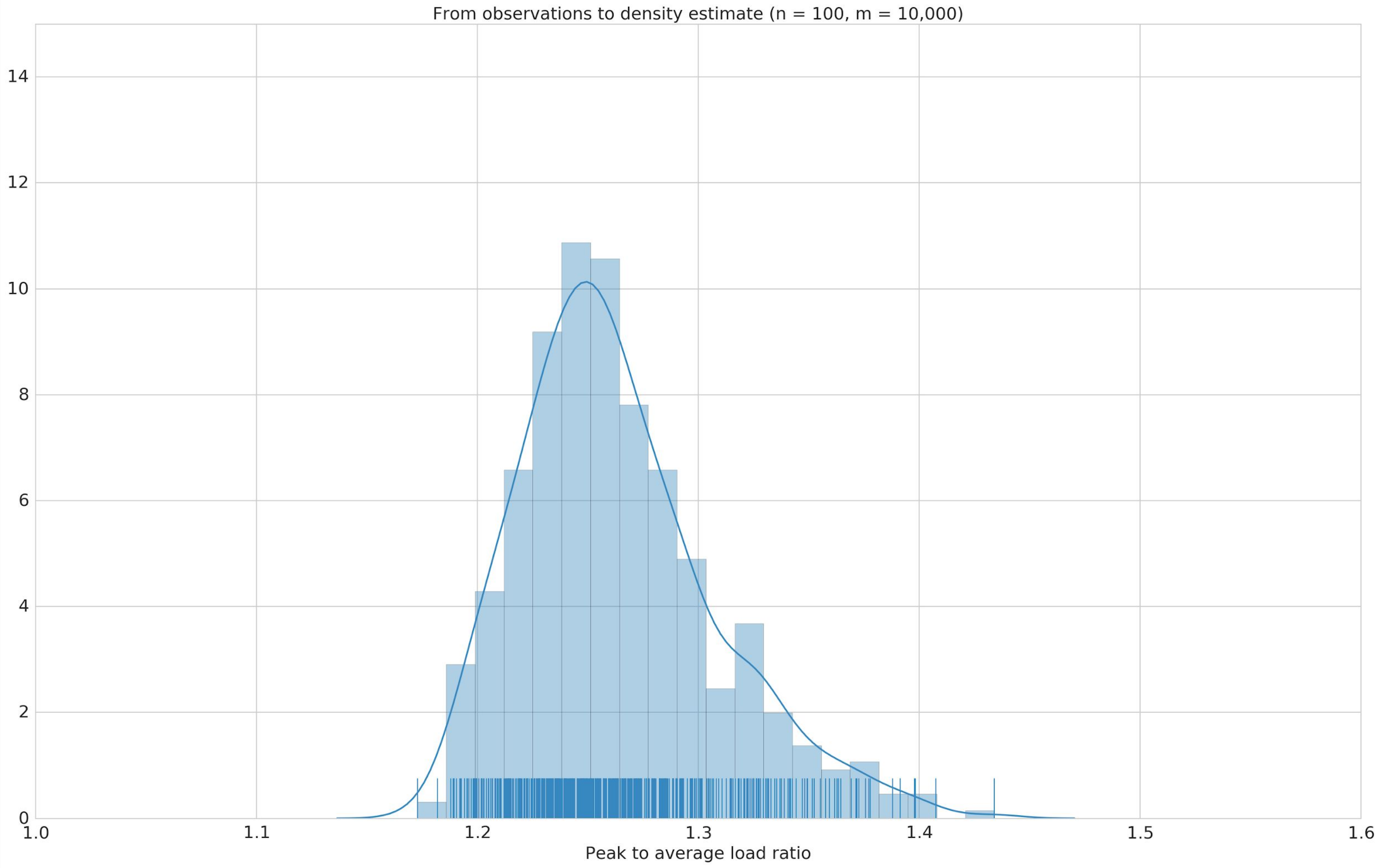
```
>>> import numpy as np
>>> import scipy
>>> current_m = 10000
>>> current_n = 50
>>> c = m / (n * np.log(n))

>>> target_m = 10 * current_m
>>> target_n = np.exp(np.real(
...   scipy.special.lambertw(target_m / c)))

>>> print target_n
336.21
```

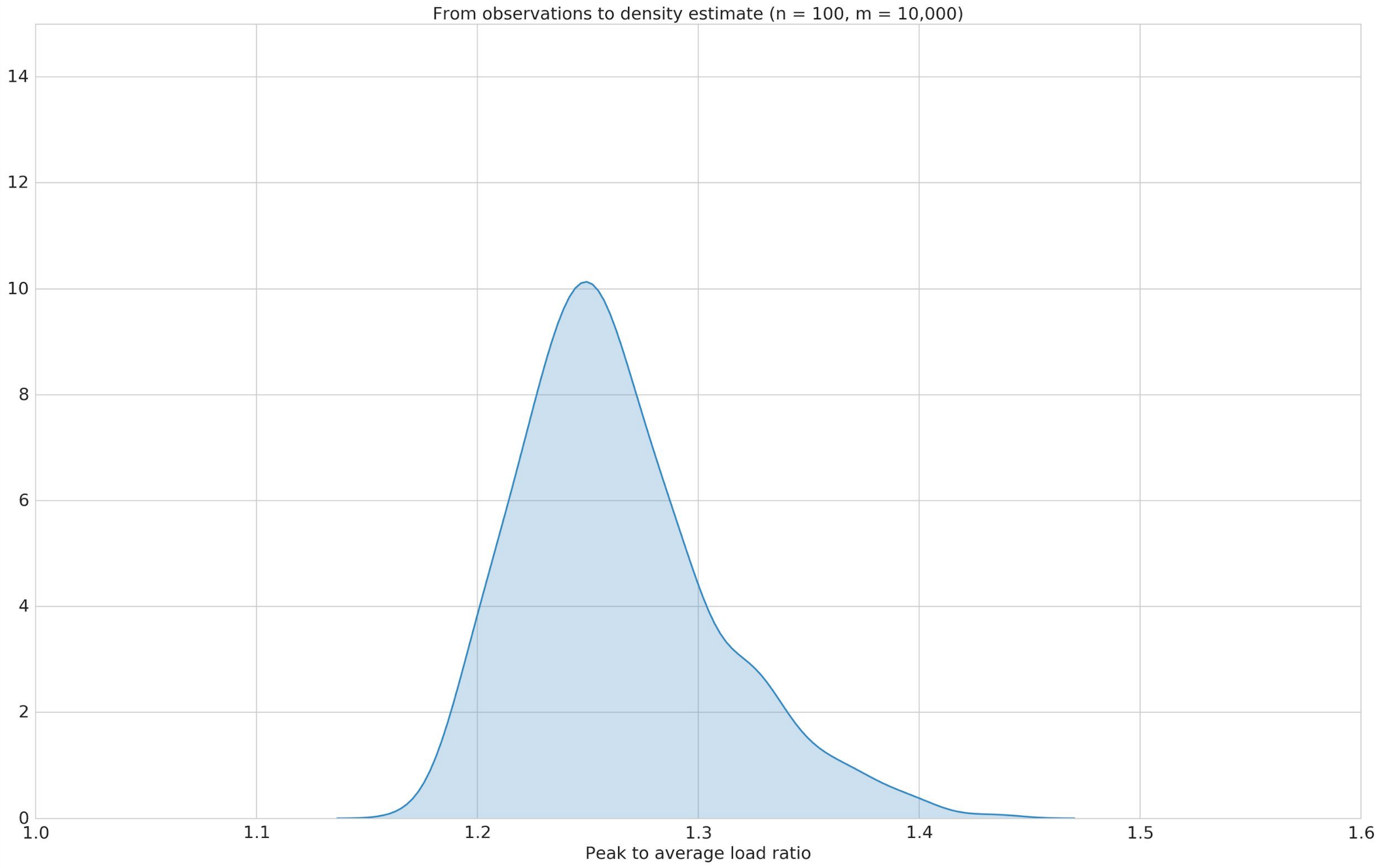
Give me more than anecdotal *) evidence!

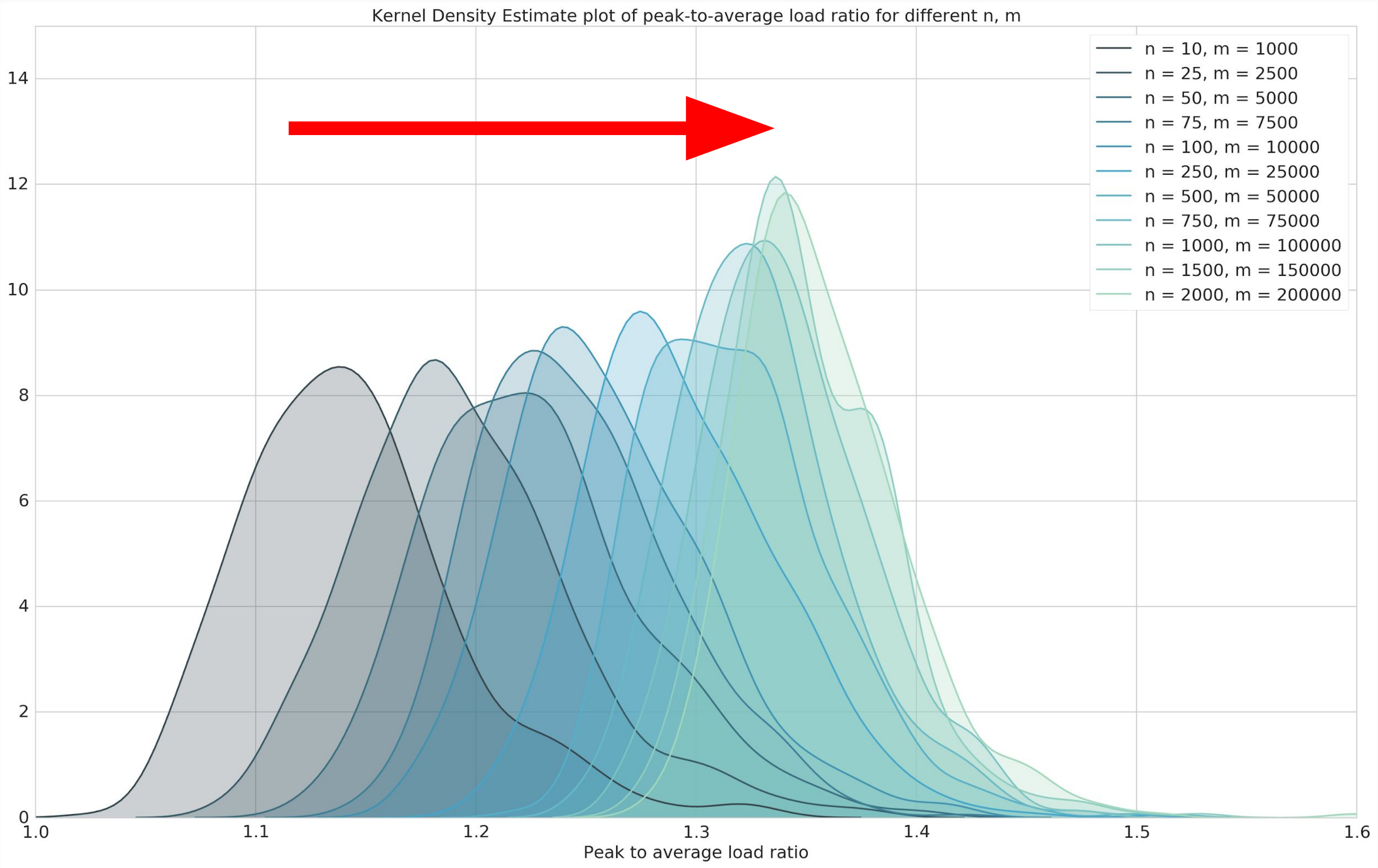
*) created with a random number generator

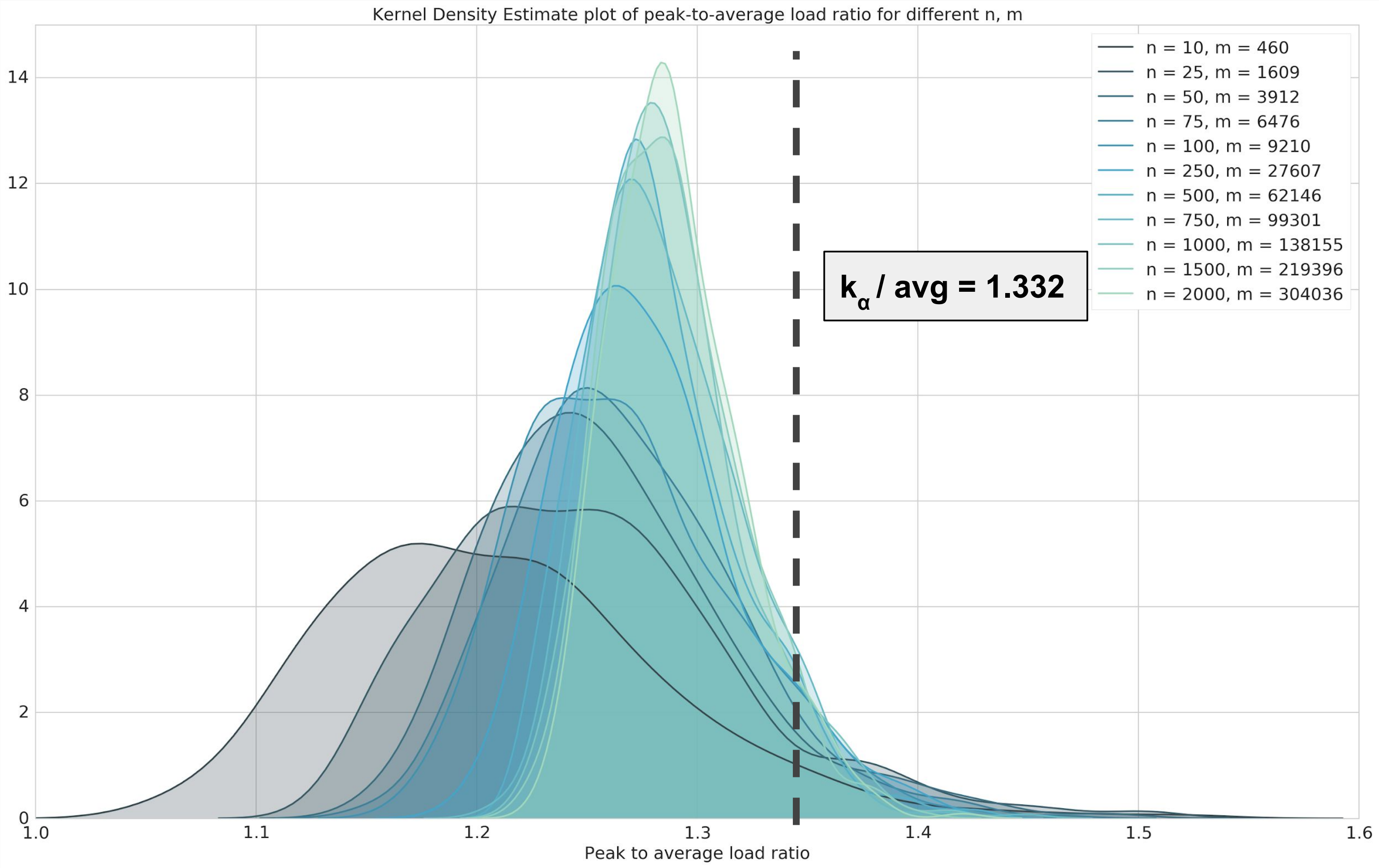

Assigning m = 10000 requests randomly to n = 100 servers
peak = 125
peak/avg = 1.25
avg = 100
Number of requests
Server number
Repeat 500x


From observations to density estimate (n = 100, m = 10,000)
0
2
4
6
8
10
12
14
1.0
1.1
1.2
1.3
1.4
1.5
1.6
Peak to average load ratio


From observations to density estimate (n = 100, m = 10,000)
0
2
4
6
8
10
12
14
1.0
1.1
1.2
1.3
1.4
1.5
1.6
Peak to average load ratio


From observations to density estimate (n = 100, m = 10,000)
14
12
10
8
6
4
2
0
1.0
1.1
1.2
1.3
1.4
1.5
1.6
Peak to average load ratio


Kernel Density Estimate plot of peak-to-average load ratio for different n, m
n = 10, m = 1000
n = 25, m = 2500
n = 50, m = 5000
n = 75, m = 7500
n = 100, m = 10000
n = 250, m = 25000
n = 500, m = 50000
n = 750, m = 75000
n = 1000, m = 100000
n = 1500, m = 150000
n = 2000, m = 200000
0
2
4
6
8
10
12
14
1.0
1.1
1.2
1.3
1.4
1.5
1.6
Peak to average load ratio


Kernel Density Estimate plot of peak-to-average load ratio for different n, m
n = 10, m = 460
n = 25, m = 1609
n = 50, m = 3912
n = 75, m = 6476
n = 100, m = 9210
n = 250, m = 27607
n = 500, m = 62146
n = 750, m = 99301
n = 1000, m = 138155
n = 1500, m = 219396
n = 2000, m = 304036
k_α / avg = 1.332
0
2
4
6
8
10
12
14
1.0
1.1
1.2
1.3
1.4
1.5
1.6
Peak to average load ratio

How can I calculate the likely peak-to-average ratio myself?

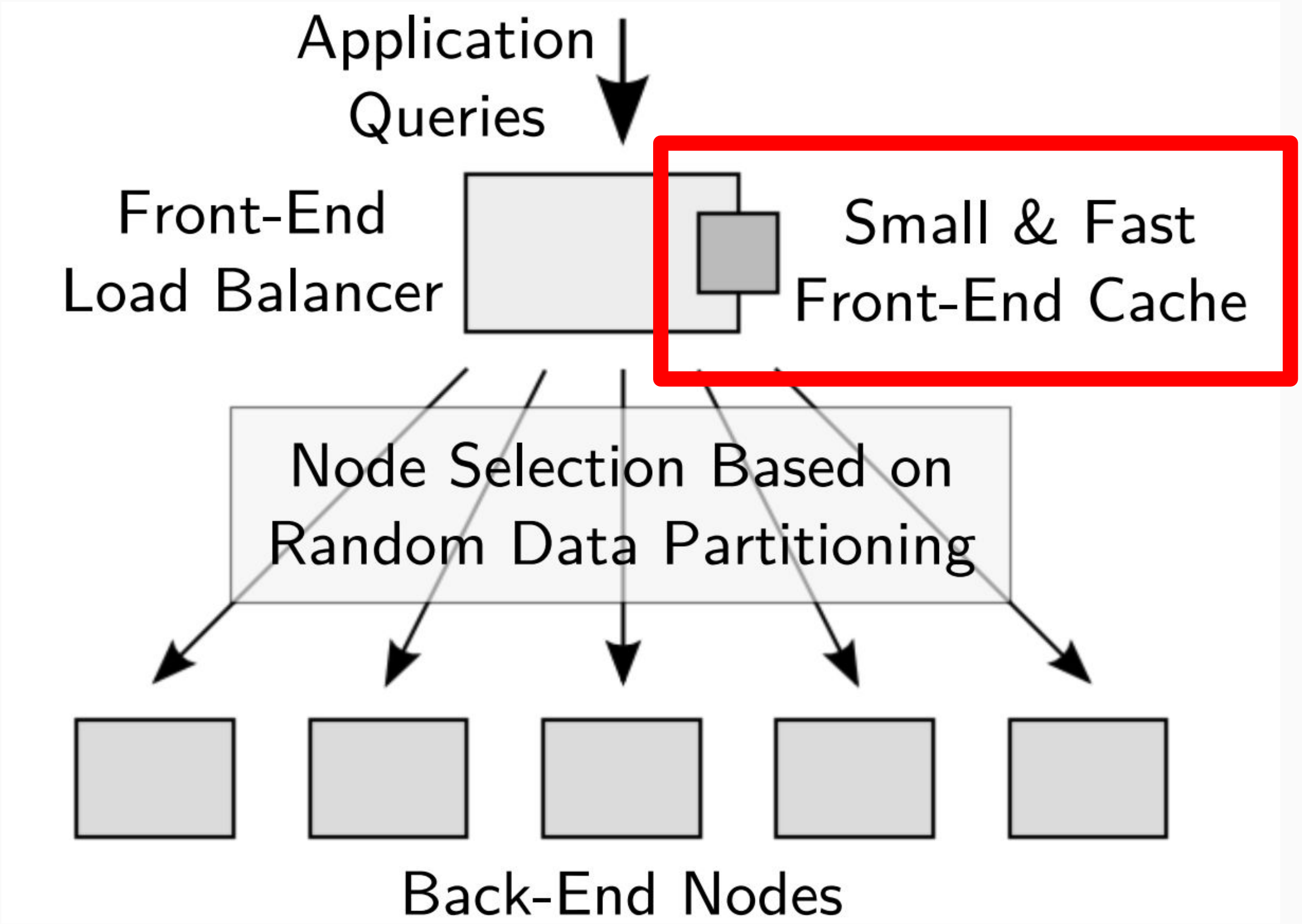
```
>>> import numpy as np
>>> import scipy

>>> c = 20       # Per choice of prev example
>>> max(
...       np.real(c * np.exp(
...          1 + scipy.special.lambertw(
...              (1 - c) / (c * np.e),
...              k=k)))
...      for k in (0, -1)) / c
1.332
```

Bounding the peak to average load ratio for a key-value store

Randomized Server

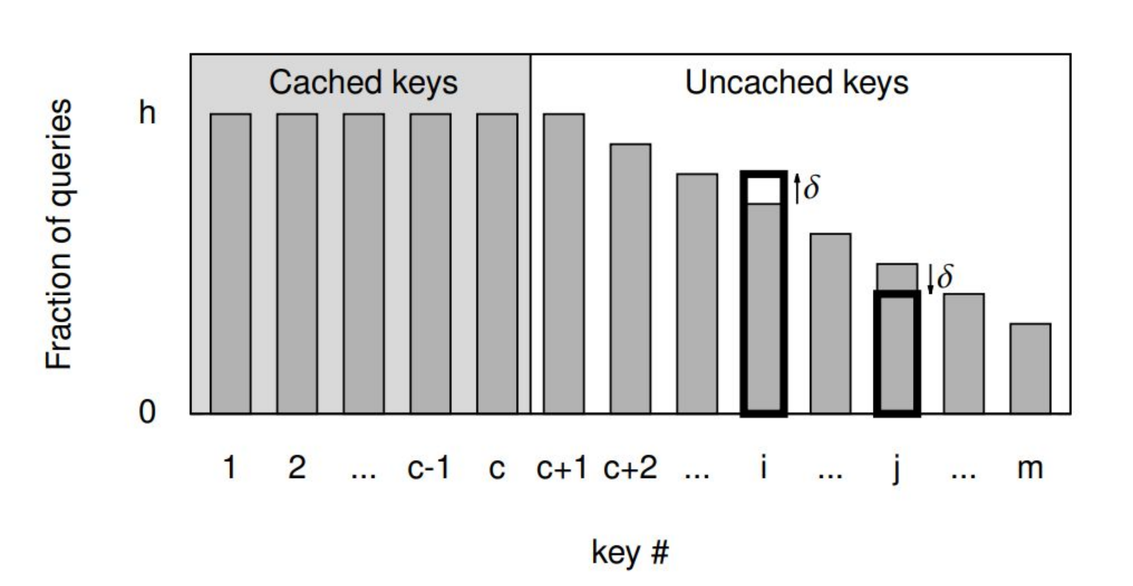
⇔

Randomized Location



From [“Small Cache, Big Effect”] paper

How many items should we cache?



From “Small Cache, Big Effect” paper

$$k_\alpha = \begin{cases} \\ \frac{m}{n} + \alpha\sqrt{2\frac{m}{n}\log n}, & \textit{if } n\log n \ll m \le n \cdot \mathrm{polylog}(n), \\ \end{cases}$$

Many, many more keys than servers.

Now do some clever substitutions

(From “Balls Into Bins”)

You should cache *O*(*N*•log(*N*)) keys!

Cache of Size $O(n \log n)$ If we choose a cache size of $c = k \cdot n \log n + 1$ where k is a constant factor, the load bound shown in Eq. (10) becomes constant in the system size:

$$\frac{1}{2}\left(1 + \sqrt{1 + \frac{2\alpha^2}{k}}\right) \quad (11)$$

From [“Small Cache, Big Effect”] paper

Recap

Takeaway #1

Randomized Load Balancing is very good if you have many “things”

Takeaway #2

Randomized Load Balancing becomes worse if you scale your system in the wrong way

Takeaway #3

Pay attention to the size of your cache when you scale your system

Thanks!

Questions?