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 ⇔ Requests into servers
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

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

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 $n = 5$ servers

- Peak: 22
- Peak/Avg: 1.10
- Avg: 20
$k^* = \begin{cases} 
\frac{\log n}{\log m} \left( 1 + \frac{\log (2)^{\log n}}{\log m} \right), \\
\text{if } \frac{n}{\text{polylog}(n)} \leq m \ll n \log n.
\end{cases}$

Peak-to-average ratio:

$O(\log n)$

(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: $O(1)$

(From "Balls into Bins")
Consequence #2: Don’t scale servers and requests linearly 1:1
\[
k_{\alpha} = \begin{cases} 
\frac{\log n}{\log \frac{n \log n}{m}} \left( 1 + \alpha \log^2 \frac{n \log n}{m} \right), & \text{if } \frac{n}{\text{polylog}(n)} \leq m \ll n \log n, \\
(d_c - 1 + \alpha) \log n, & \text{if } m = c \cdot n \log n \text{ for some constant } c.
\end{cases}
\]
Assigning $m = 10000$ requests randomly to $n = 100$ servers

- Peak: 125
- Peak/avg: 1.25
- Average: 100
Assigning $m = 50000$ requests randomly to $n = 500$ servers

- Peak = 130
- Peak/avg = 1.30
- Avg = 100
Assigning $m = 100000$ requests randomly to $n = 1000$ servers

- Peak: 131
- Peak/avg: 1.31
- Average: 100

Server number

Number of requests
(From "Balls into Bins")

\[
k_\alpha = \begin{cases} 
\frac{\log n}{\log \frac{n \log n}{m}} \left(1 + \alpha \frac{\log (n \log n)}{\log \frac{n \log n}{m}}\right), & \text{if } \frac{n}{\text{polylog}(n)} \leq m \ll n \log n, \\
(d_c - 1 + \alpha) \log n, & \text{if } m = c \cdot n \log n \text{ for some constant } c,
\end{cases}
\]
How can I calculate this myself?

```python
>>> 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
- Average: 100
- Peak/Average: 1.25

Repeat 500x
From observations to density estimate ($n = 100, m = 10,000$)
Kernel Density Estimate plot of peak-to-average load ratio for different n, m

\[ k_\alpha / \text{avg} = 1.332 \]
How can I calculate the likely peak-to-average ratio myself?

```python
>>> 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
Many, many more keys than servers.

\[ k_\alpha = \begin{cases} 
\frac{m}{n} + \alpha \sqrt{2 \frac{m}{n} \log n}, \\
\text{if } n \log n \ll m \leq n \cdot \text{polylog}(n), 
\end{cases} \]

Now do some clever substitutions

(From “Balls Into Bins”)

\[ k_\alpha = \begin{cases} 
\frac{m}{n} + \alpha \sqrt{2 \frac{m}{n} \log n}, \\
\text{if } n \log n \ll m \leq n \cdot \text{polylog}(n), 
\end{cases} \]
You should cache $O(N \cdot \log(N))$ 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:

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

(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?