Kraken: Leveraging Live Traffic Tests to Identify and Resolve Resource Utilization Bottlenecks in Large Scale Web Services

Authors:

Kaushik Veeraraghavan, Justin Meza, David Chou, Wonho Kim, Sonia Margulis, Scott Michelson, Rajesh Nishtala, Daniel Obenshain, Dmitri Perelman, Yee Jiun Song

Facebook Inc

Presented by:
Gabriela Gierasimiuk - University of Warsaw, Poland

Agenda

1. Introduction
2. New system
3. Design
4. Implementation
5. Evaluation
6. Conclusion

1.

INTRODUCTION

Multiple data centers handle load generated by **1.7 billion** of users (in 2016)

Resource utilization challenges

- Evolving workload
- Infrastructure heterogeneity
- Changing bottlenecks

We need to:

- handle peak load
- assess the capacity of a complex system

It is important to use resources efficiently.

How to achieve that?

Common approaches

1. Load modeling: simulate how system behaves at high load

2. Load testing: benchmark using synthetic workloads

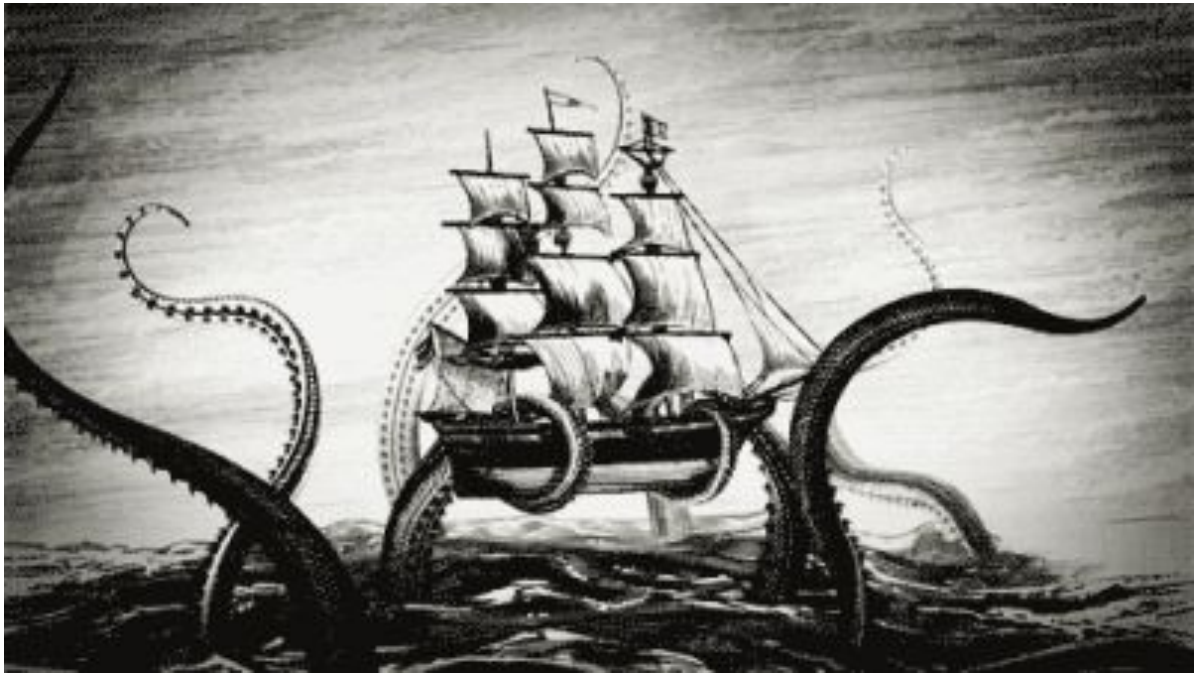
Live user traffic

- most current workload
- accurate distribution of reads and writes
- no need for a custom test setup

2.

NEW SYSTEM

Kraken



Kraken

- live traffic load tests
- good user experience while testing
- reliably tracking system health

Load test

- directs user traffic at a target cluster or region
- success: hitting the utilization targets without crossing the latency or error rate thresholds

In case of failure we got data that allows:

- make next test safer
- increase capacity

Problems found after first year

- non-linear response where a small traffic shift directed at a data center could trigger an error spike
- complex dependencies between systems - difficulty to find system that initially failed

Achievements

- identify and remediate regressions, address load imbalance and resource exhaustion across Facebook's fleet

- increase the capacity of Facebook's infrastructure from 70% to over 90% of theoretical capacity

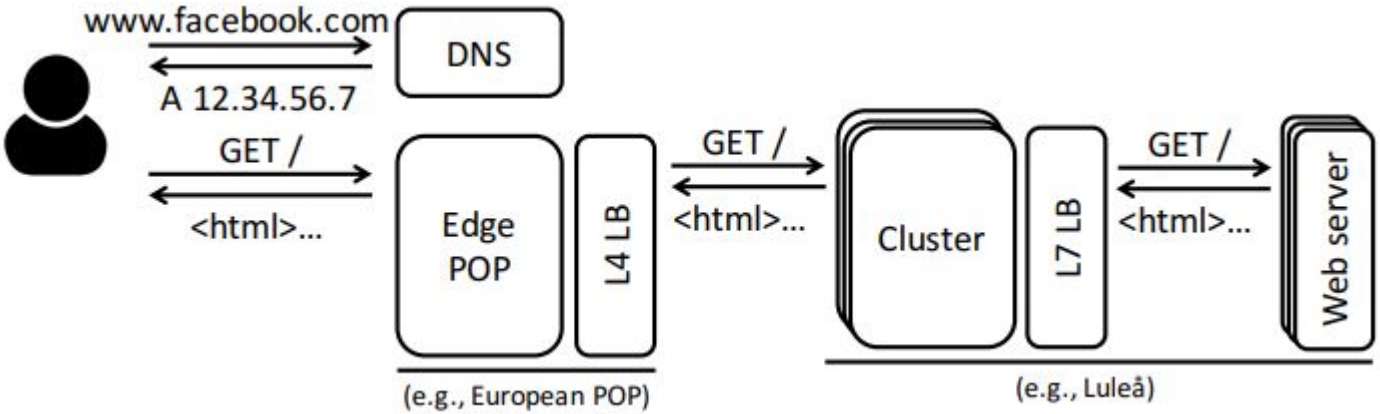
Limitation

- servers have to be stateless
- load must be controllable by re-routing requests
- downstream services respond to shifts in upstream service load

3.

DESIGN

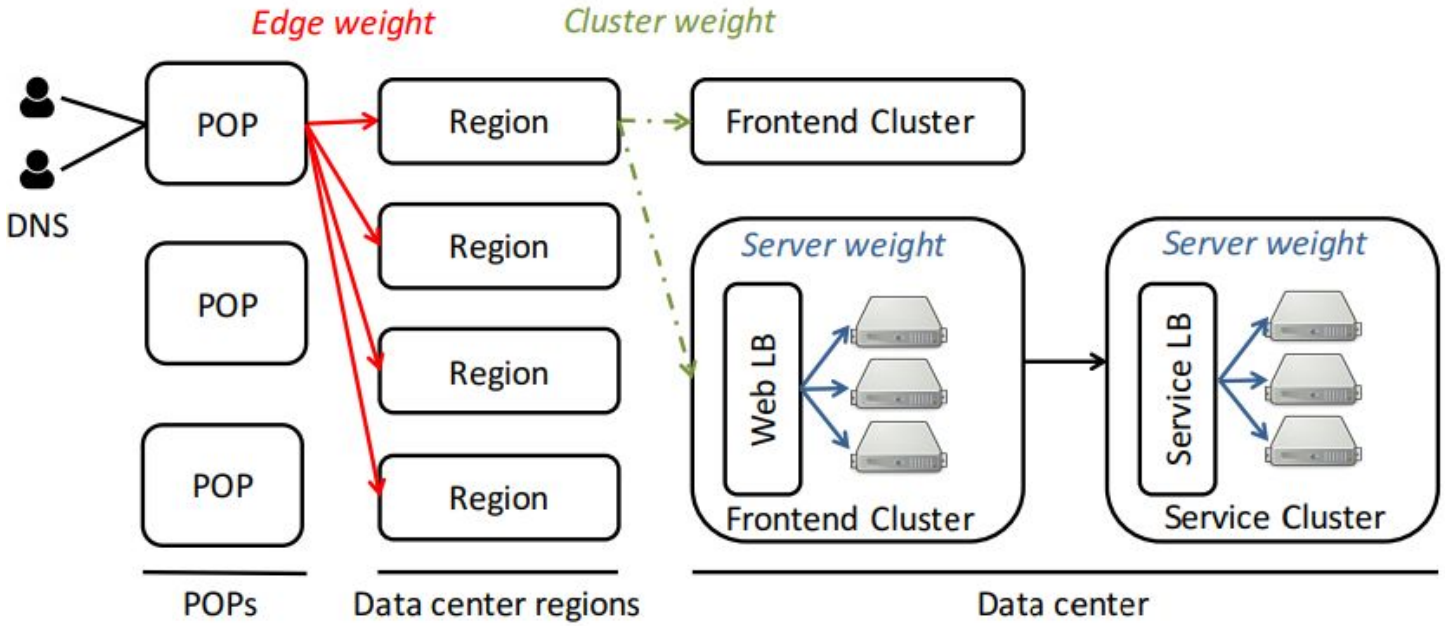
Managing user request



Traffic management

- 1–3 data centers in close proximity are grouped into a “region”
- machines in each data center are grouped to:
  - “frontend” clusters of web servers
  - “backend” clusters of storage systems
  - service clusters (service is as a set of subsystems that provide a particular product)
- each cluster has a few thousand different machines

Traffic management



Health monitoring

First idea:

query Gorilla for the metrics of all important systems, compute the next traffic shift using results

Problem:

tuning individual systems instead of the overall user experience

Solution:

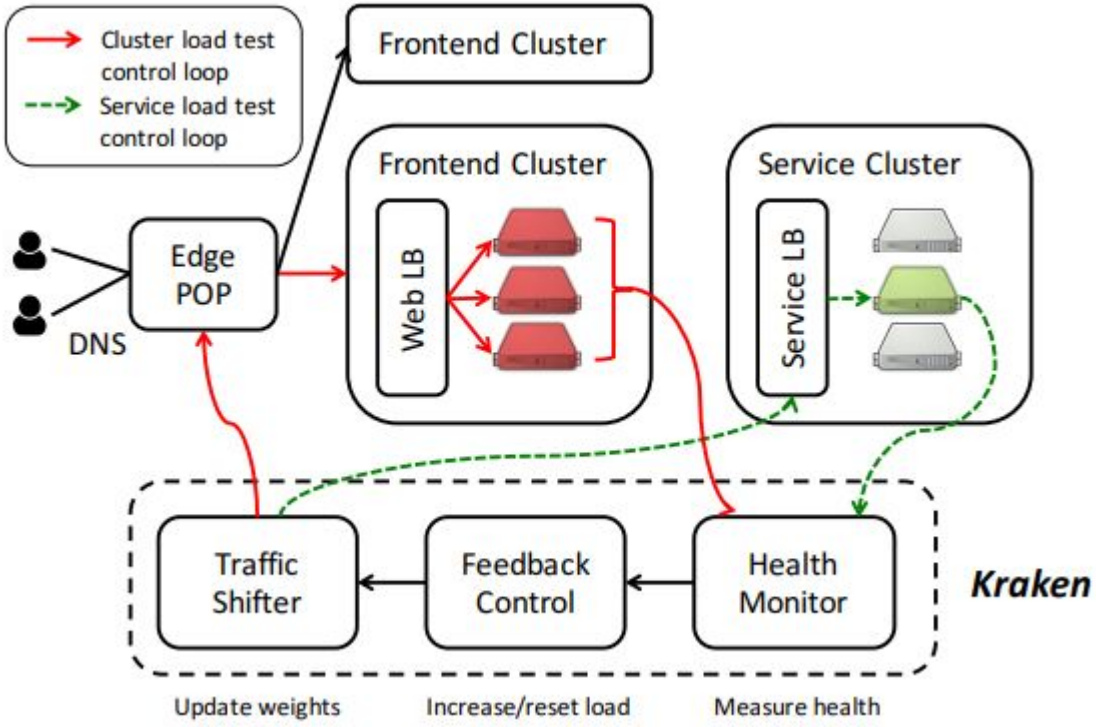
use metrics that measure user experience

Metrics

- the web server's 99th percentile response time
- HTTP fatal error rate
- median queuing delay on web servers
- the 99th percentile CPU utilization on cache machines

Each metric has a threshold, reaching threshold stops Kraken, before the system becomes unhealthy

Kraken traffic management



Fixing bottlenecks

- allocating additional capacity to the failing system
- system reconfiguration
- creating and deploying new load balancing algorithms
- performance tuning
- system redesign (rare cases)

4.

IMPLEMENTATION

Traffic shifting module

- Proxygen - an open source software L4 and L7 load balancer, reads configuration files with customized edge and cluster weights for each POP
- those weights determine the fraction of user traffic to direct at each frontend cluster
- by adjusting cluster weights, we can increase the relative fraction of traffic a cluster receives compared to its peers
- similarly for edge weights and regions.

Traffic shift

1. Kraken takes the target of the test as input
2. updates the routing file stored in the configuration store with this change
3. the configuration store notifies the Proxygen load balancers in a remote POP of the existence of the new configuration file

It takes about 120 seconds end-to-end for Kraken to initiate a traffic shift

- 60 seconds for Kraken to update weights and execute the traffic shift
- Gorilla, aggregates metrics in 60 second intervals

Health monitoring module

1. receives the system being tested as an input
2. queries Gorilla for metrics
3. compare results with thresholds

Service type	Metrics
Web servers	CPU utilization, latency, error rate, fraction of operational servers
Aggregator–leaf	CPU utilization, error rate, response quality
Proxygen [39]	CPU utilization, latency, connections, retransmit rate, ethernet utilization, memory capacity utilization
Memcache [31]	Latency, object lease count
TAO [10]	CPU utilization, write success rate, read latency
Batch processor	Queue length, exception rate
Logging [23]	Error rate
Search	CPU utilization
Service discovery	CPU utilization
Message delivery	CPU utilization

Feedback control

There is a trade-off between load test speed and system health.

Practice shows that initial load increase increments of around 15% is a good balance

Kraken reduces traffic shifts when health metric approach a threshold.

External conditions

1. Request spike
   - planned (national holidays, social and sporting events) - **do not run tests**
   - unexpected - **abort tests, distribute load**

2. Major faults in system operation (e.g. kernel crash) - **abort tests, distribute load to healthy clusters**

3. External faults such as a network partition and power loss - **as above**

5.

EVALUATION

Questions to answer

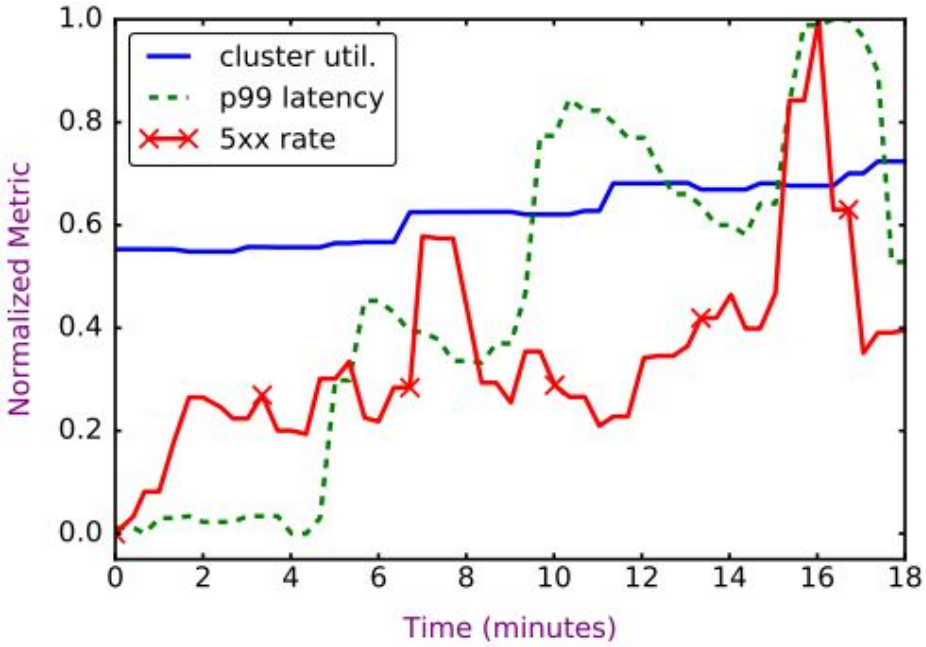
1. Does Kraken allow us to validate capacity measurements at various scales?

2. Does Kraken provide a useful methodology for increasing utilization?

Measuring an individual web server’s capacity

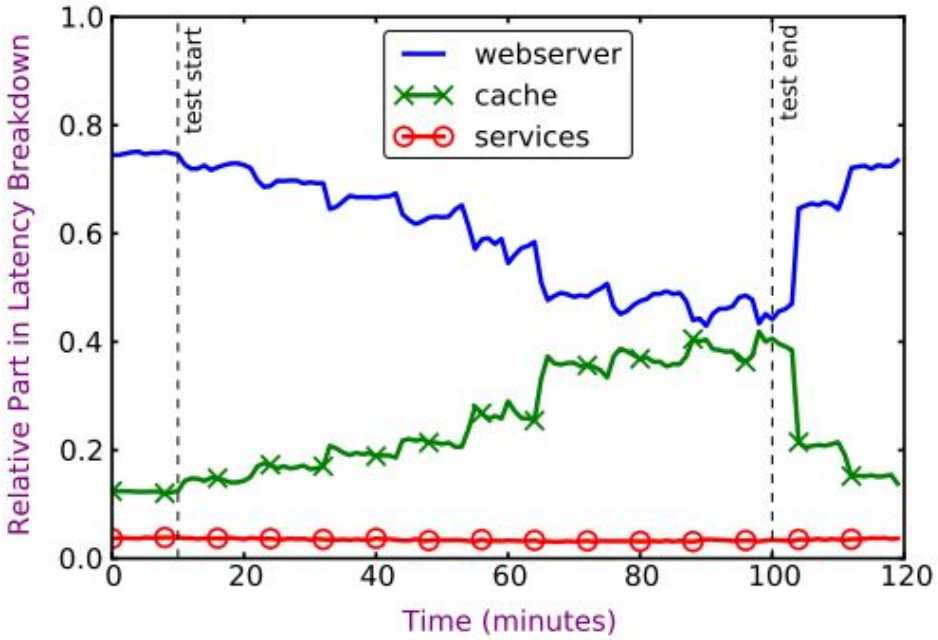
- error rate and latency to identify peak capacity
- 32 servers turned out to be the best number in terms of variance
- baseline server capacity - the average of the 32 servers

Measuring a cluster's capacity



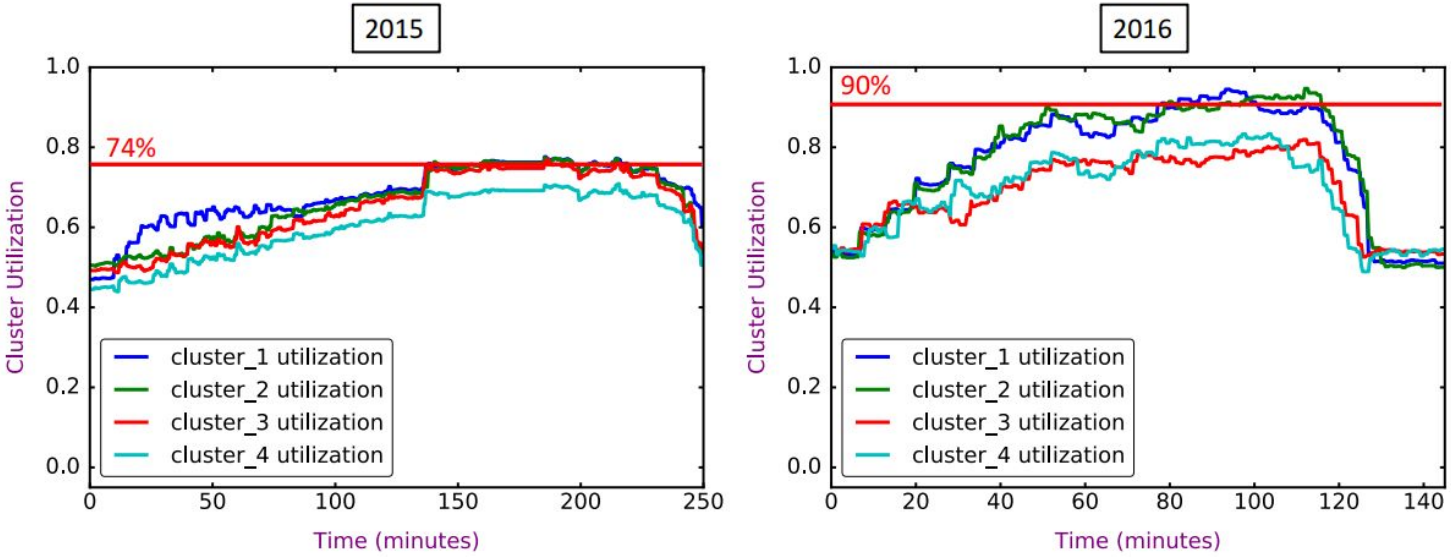
75% instead of the target utilization of 93% -> **unsuccessful**

What cluster capacity was low?



cache latency increases from 15% to around 40% -> **cache is a bottleneck**

Increasing utilization



Hash weights for cache

- **Problem:** a small number of cache servers were driven out of CPU
- **Reason:** clusters stored a lot of frequently-accessed data (e.g., popular content), uniform hash used to cache servers
- **Fix:** hash weight based on the frequency of access

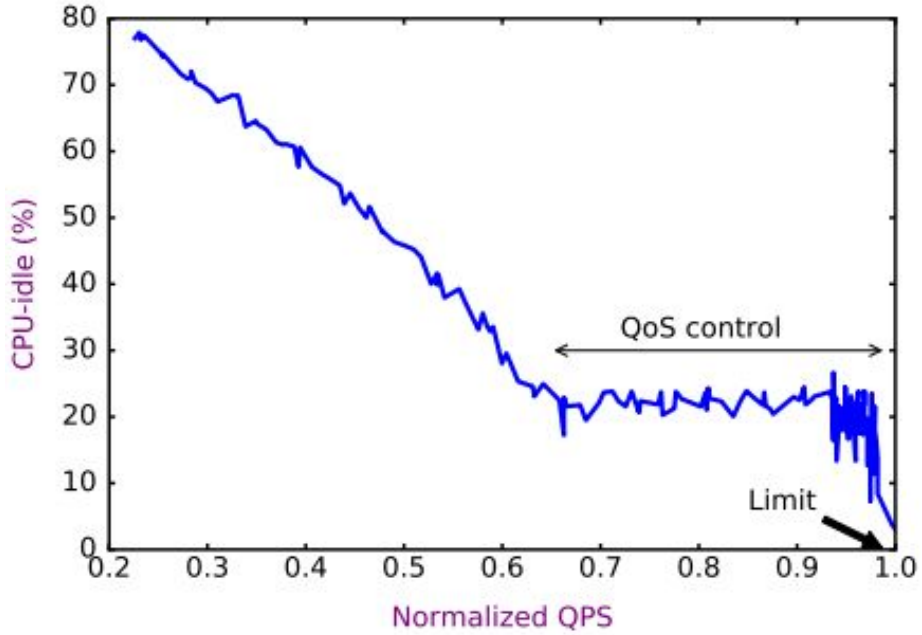
Network saturation

- **Problem:** spike in network traffic saturates two top-of-rack network switches
- **Reason:** a large number of servers on these racks were responsible for retrieving posts on the News Feed
- **Fix:** a new network monitoring tool

Service-level load testing

- allows to find service-level issues
- service owners can test their systems independently
- developers can identify performance issues in their services without needing to wait until the next regional test

Kraken load testing News Feed



The News Feed aggregators dynamically adjust request quality

6.

CONCLUSION

Large scale web services modelling challenges

- rapidly evolving systems
- distributed across the world
- changing workloads

Kraken handles those challenges

- Kraken empirically load test capacity on every level
- Test results allows to identifying bottlenecks and iteratively improve infrastructure utilization
- User load capacity increased by over 20% using the same hardware.

Questions?