Latency at Scale

Rong Chen

ACK: Some slides borrowed from: C3 (NSDI'15), TimeTrader (MICRO'15)
Content based on: The Tail at Scale (CACM'13)

What is Latency?

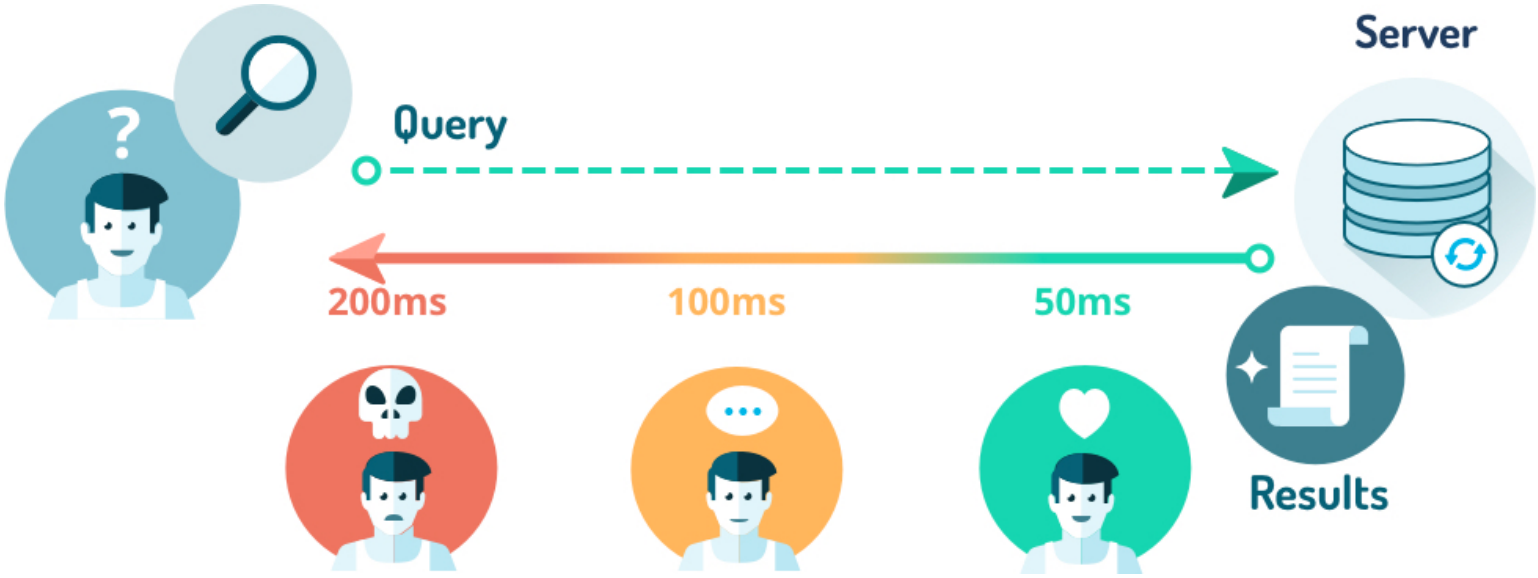
Src: http://z63.org/?p=37321

What is Tail Latency?

Where is Latency?

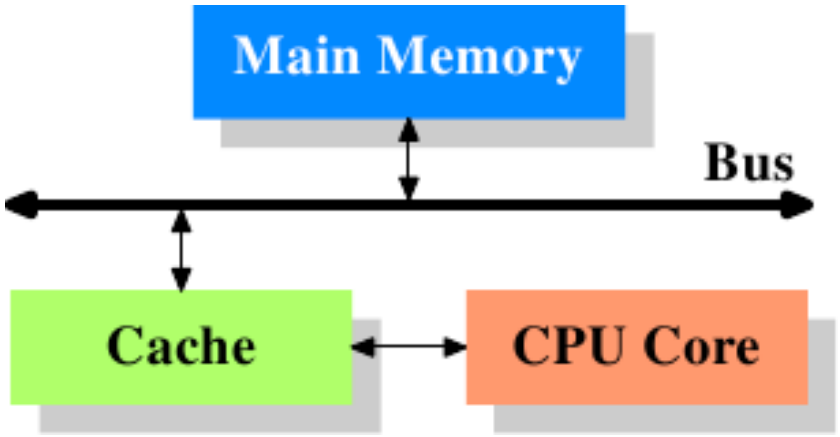
- CPU and its cache
- Client and server over a network
- Application and disk
- Anywhere a system does work

Why latency is important?



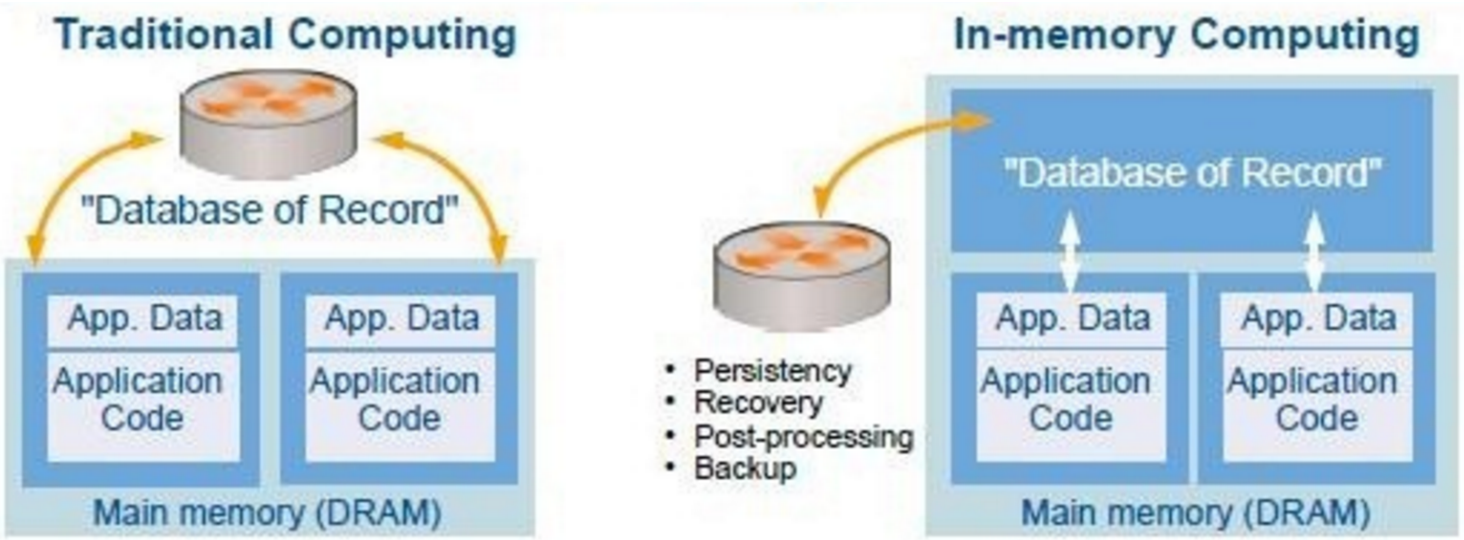
How we reduce latency before?

- CPU Cache



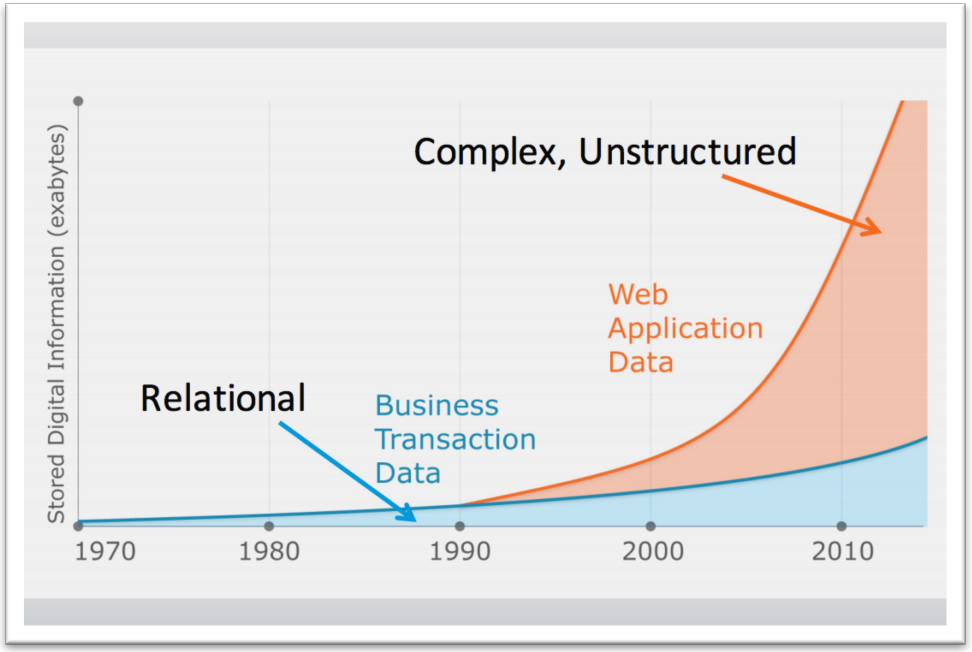
How we reduce latency before?

- In-memory Computation



What is the new trend?

- Data is growing exponentially



What is the new trend?

- Searching large data is becoming prevalent
  - **Online Search (OLS)**: **Interactively** query and access data
    - Key component of many web applications, **not only** *Web Search*
    - e.g., Facebook, Twitter, Advertisements

New challenge under the trend

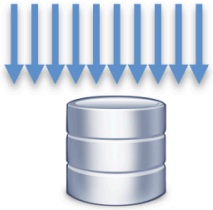
- Data are now spanning thousands of servers
- Consulting data is growing to multi-terabytes scale
- Latency critical applications are becoming popular

Self-driving Car

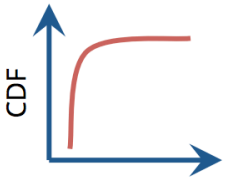
VR Instruments

Online Shopping

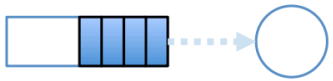
Why latency exists?



Resource contention



Skewed access patterns



Queueing delays

Background activities

Sources: Resource Contention

- Machines are shared by different applications

- Application might content for shared resources
  - CPU cores
  - processor caches
  - memory bandwidth
  - network bandwidth

Sources: Skewed Access Patterns

- Some records are more likely to be accessed than others
  - Hot topics in social network
  - On-sale items in online shopping

Sources: Queueing Delays

- Queueing are used in multiple layers of servers and network switches
  - Amplify the queueing delays

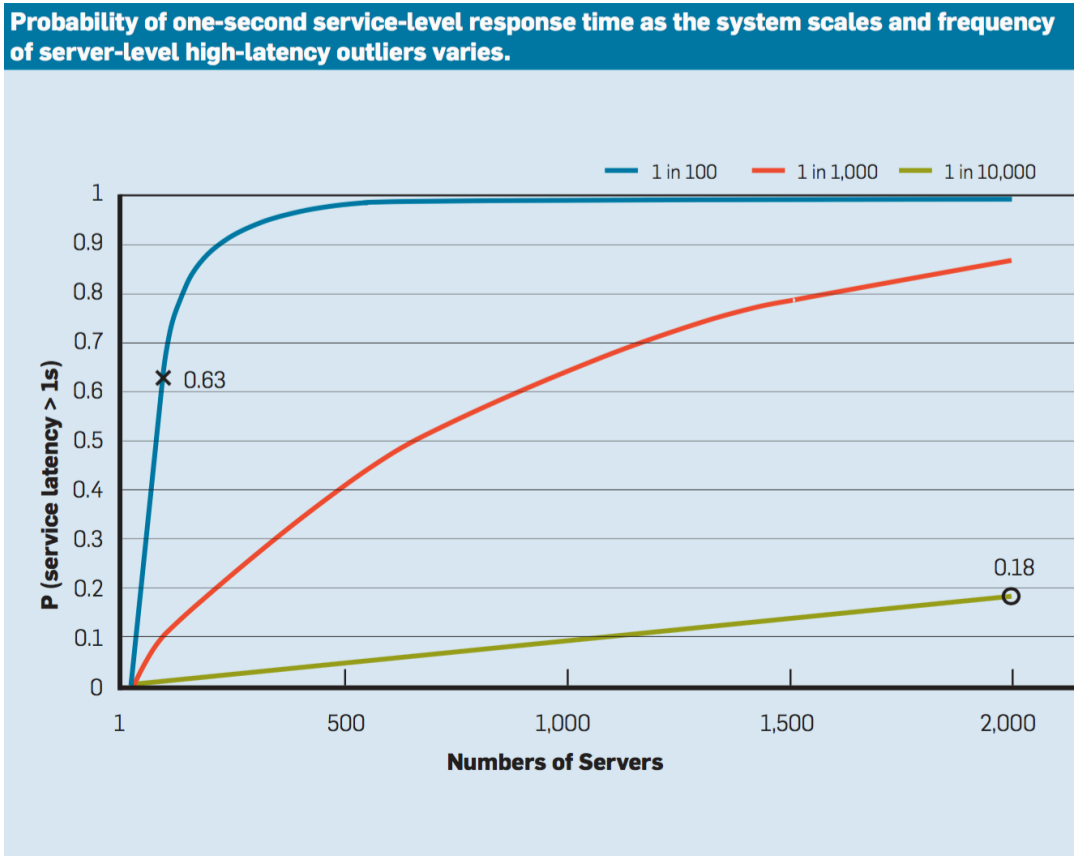
Sources: Background Activities

- Background daemons scheduled periodically
  - data reconstruction in distributed file systems
  - log compactions in storage systems
  - garbage collection in some programming languages

- Cause periodic spikes in respond time / latency

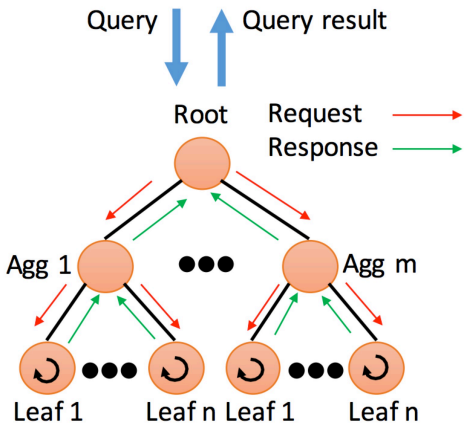
Latency Amplified By Scale

- Even rare performance hiccups affect *a significant fraction* of all requests in the large-scale settings.



Typical Case: OLS Architecture

- Request-Compute-Reply
  - Request: root -> leaf
  - Compute: leaf node
  - Reply: leaf -> root



- *Root* servers distribute a request through *intermediate* servers to a very large number of *leaf* servers.

Typical Case: OLS Architecture

- Effect of large fan-out on latency distributions

Table 1. Individual-leaf-request finishing times for a large fan-out service tree (measured from root node of the tree).

	50%ile latency	95%ile latency	99%ile latency
One random leaf finishes	1ms	5ms	10ms
95% of all leaf requests finish	12ms	32ms	70ms
100% of all leaf requests finish	40ms	87ms	140ms

Reducing Component Variability

- Differentiating service classes and higher-level queuing
- Reducing head-of-line blocking
- Managing background activities and synchronized disruption

Living with Latency Variability

Latency variability is inevitable

Tail-tolerant techniques are necessary

- Within request short-term adaptations
- Cross request long-term adaptations

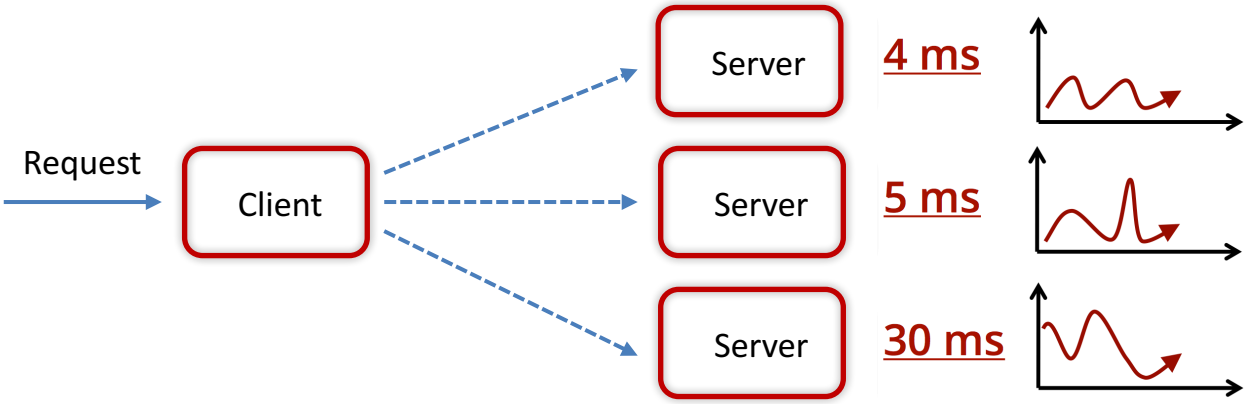
Within Request Short-Term Adaptations

- Web services often deploy **replicas** of data items to provide additional throughput and maintain availability

Leverage the existing replicas to reduce latency variability

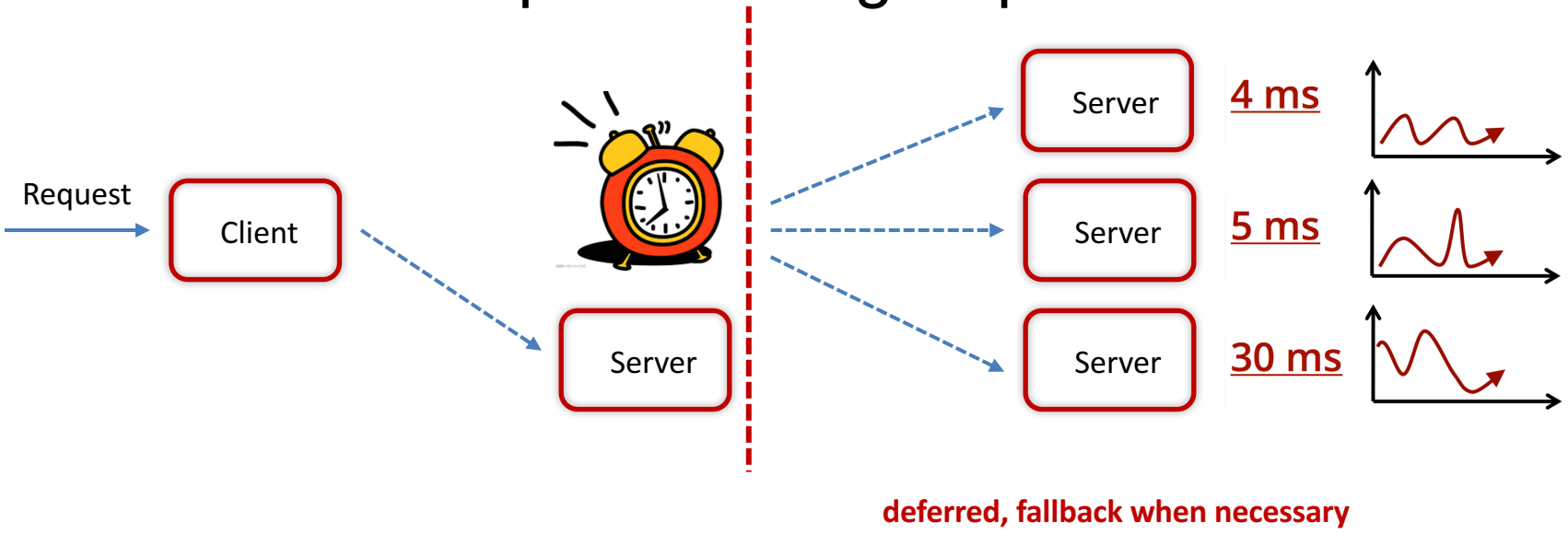
Hedged Requests

- Issue the same request to multiple replicas
- Use the results from whichever replica responds *first*



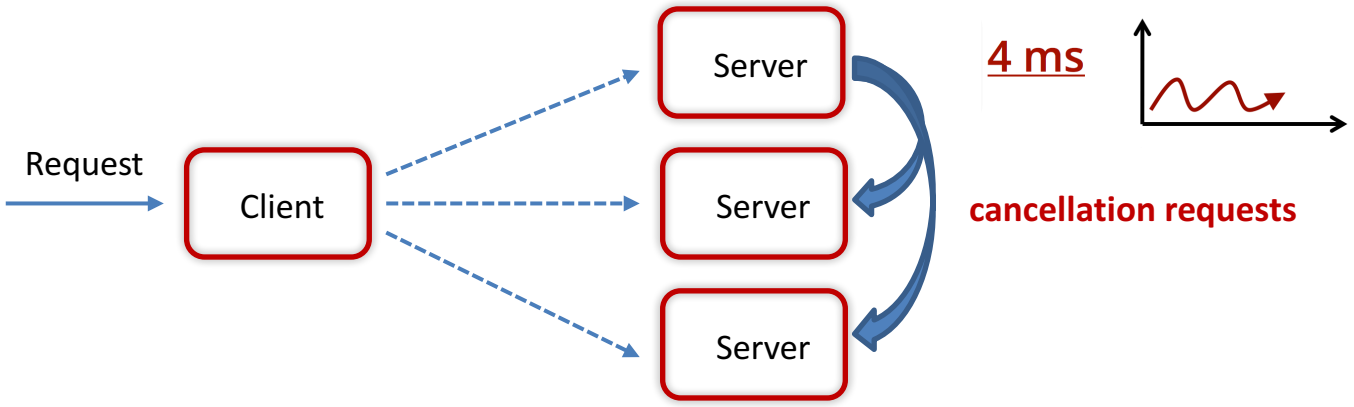
Hedged Requests

- Need to ensure the load only increases *modestly*
- Defer the step of sending requests to all servers



Tied Requests

- Further optimization based on Hedged Requests
- Faster cancellation of requests
  - Variability mainly comes from *queueing delay* rather than *execution delay*



Latency Improvement

Table 2. Read latencies observed in a BigTable service benchmark.

	Mostly idle cluster			With concurrent terasort		
	No hedge	Tied request after 1ms		No hedge	Tied request after 1ms	
50%ile	19ms	16ms	(–16%)	24ms	19ms	(–21%)
90%ile	38ms	29ms	(–24%)	56ms	38ms	(–32%)
99%ile	67ms	42ms	(–37%)	108ms	67ms	(–38%)
99.9%ile	98ms	61ms	(–38%)	159ms	108ms	(–32%)

Alternatives

- Probe remote queues first, submit to the *least-loaded* server

- Common pitfalls
  - load levels can *change* between probe and request time
  - request service times can be difficult to estimate
  - clients can create hot spots by all picking the same (least-loaded) server.
- Related Work
  - C3: Cutting Tail Latency in Cloud Data Stores via Adaptive Replica Selection (NSDI'15)

More general than you think

- Not restricted to simple replication
- Applicable to more-complex coding schemes
- Related Work
  - Erasure Coding in Windows Azure Storage (ATC'12)

Cross-Request Long-Term Adaptations

- Reduce latency variability caused by coarser-grain phenomena

- Static resource partition is usually not sufficient
  - Machines are neither *uniform* nor *constant* overtime
  - Outliers can cause data-induced *load imbalance*

Micro-partitions

- Generate many more partitions than the number of machines
- Dynamically assign and balance the partitions to machines
  - e.g. Tablets in BigTable
    - typically each machine managing 20 ~ 1,000 tablets
- Improve failure-recovery speed as well
- Similar notations: virtual server partition, virtual processor partition

Selective Replication

- Enhancement of Micro-partition
- Predict possible load imbalance and create additional replicas
  - e.g., Google's web search system will make additional copies of popular and important documents in multiple micro-partitions

Latency-induced Probation

- Temporarily exclude particularly slow machines
- Continue to issue shadow requests to the excluded servers
- Reincorporated excluded machines when the problem abates
- Counterintuitive
  - removal of serving capacity from a live system during periods of high load actually improves latency

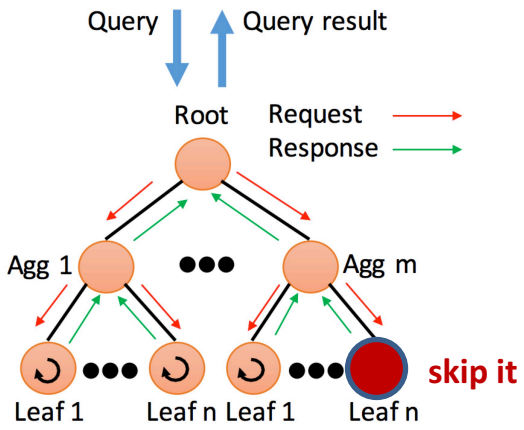
Large Information Retrieval Systems

- Speed is a key quality metric
  - Returning good results quickly is *better* than returning the best results slowly
- Techniques applied to deal with imprecise result
  - Good-enough schemes
  - Canary requests

Good-enough Schemes

- Return once *a sufficient fraction* of all the leaf servers has responded

- Skip nonessential subsystems to improve responsiveness
  - e.g. Results from ads or spelling-correction systems are easily skipped for Web searches if they do not respond in time



Canary Requests

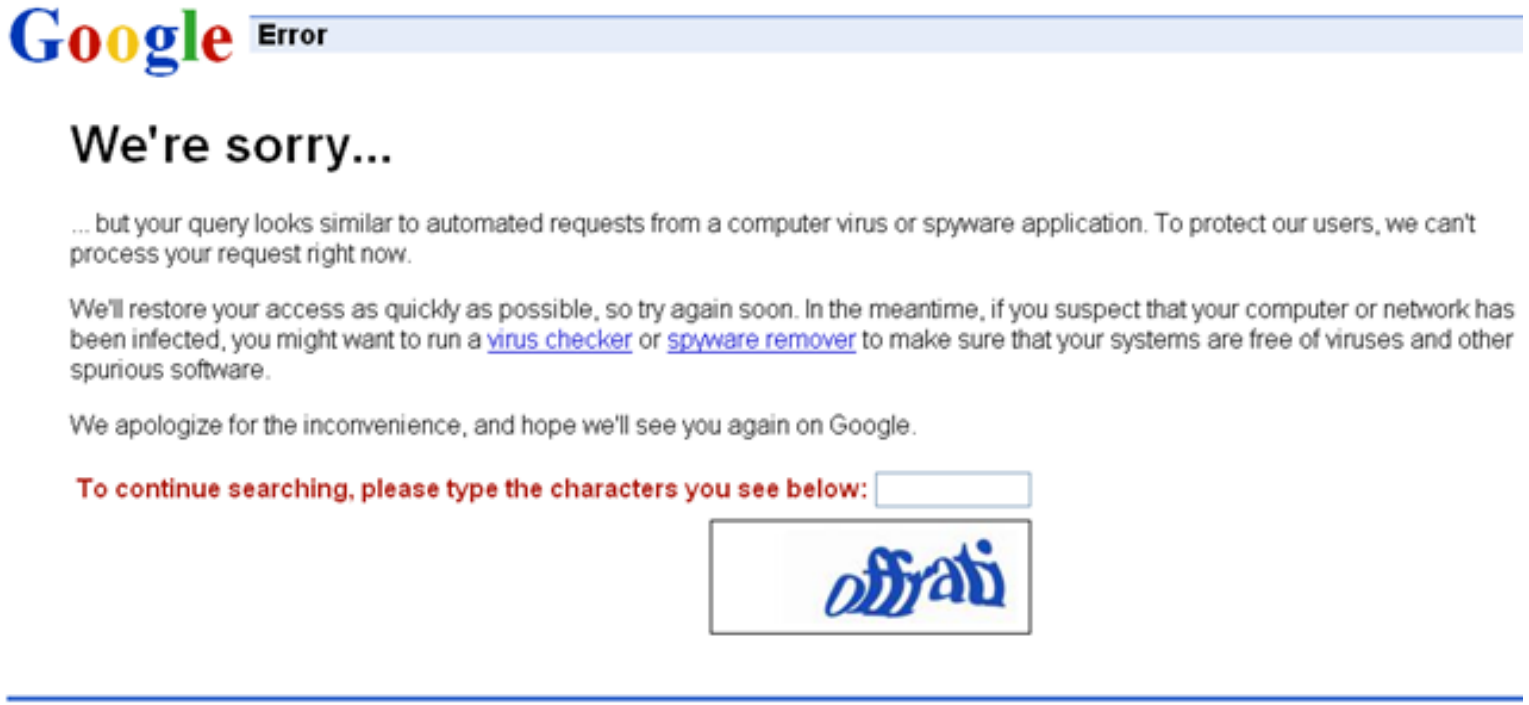
- Request incurs unexpected crash or long delays

Canary Requests

- Provide additional safety
- Slightly increase in latency
  - Wait for single or two servers to respond is much *less* variable than in the large fan-out settings

Canary Requests

- One vivid example from daily life



Mutations

- Tolerating latency variability for operations that mutate state is somewhat *easier*
  - The scale of latency-critical modifications in these services in generally *small*
  - Updates can often performed *off the critical path*, after responding to user
  - Many services can tolerate *inconsistent update* models
  - Services require the consistent updates usually go through Paxos, which is inherently tail-tolerant

Hardware Trends and Their Effects

- Trends that further *hurt* the latency
  - Variability at the hardware level is likely to be higher
  - Device heterogeneity
  - Increasing system scale

- Trends that help *mitigate* the latency
  - Higher bisection bandwidth
  - Lower per-message overheads

Case Study: C3 (NSDI'15)

C3

Adaptive replica selection mechanism that is robust to service time heterogeinity

Impact of Replica Selection in Practice?

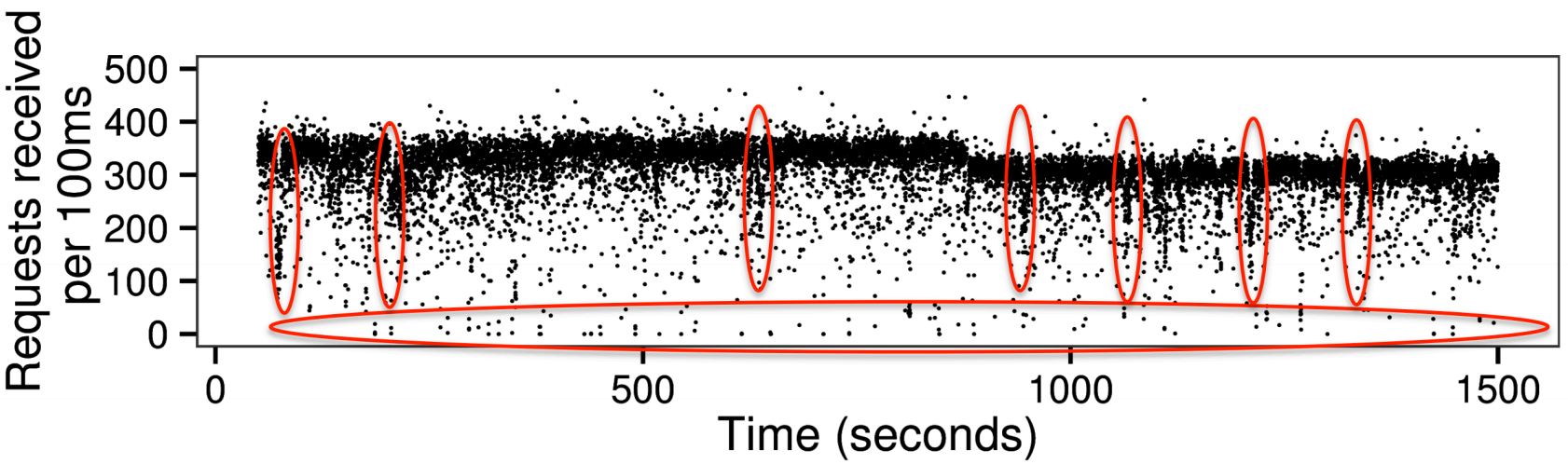
Dynamic Snitching

Uses history of read latencies and I/O load for replica selection

Experimental Setup

- Cassandra cluster on Amazon EC2
- 15 nodes, m1.xlarge instances
- Read-heavy workload with YCSB (120 threads)
- 500M 1KB records (larger than memory)
- Zipfan key access pattern

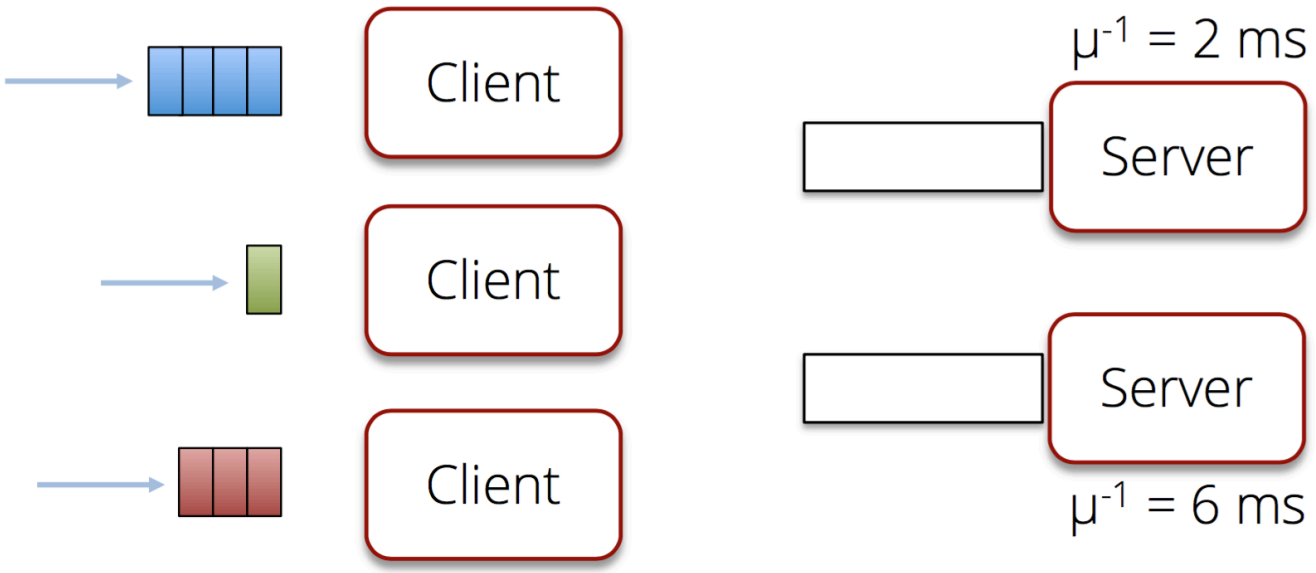
Cassandra Load Profile



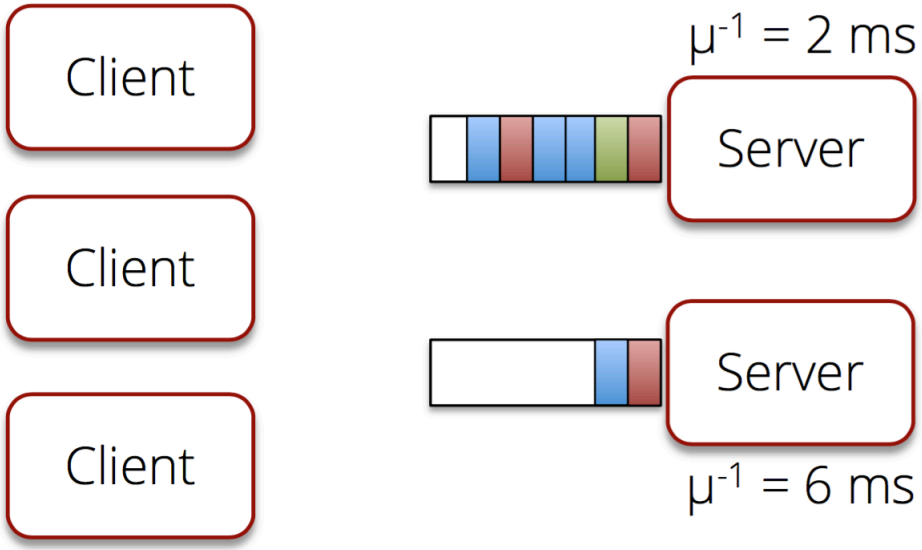
Also observed that

99.9th percentile latency ~ 10x median latency

Replica Ranking in C3



Replica Ranking in C3

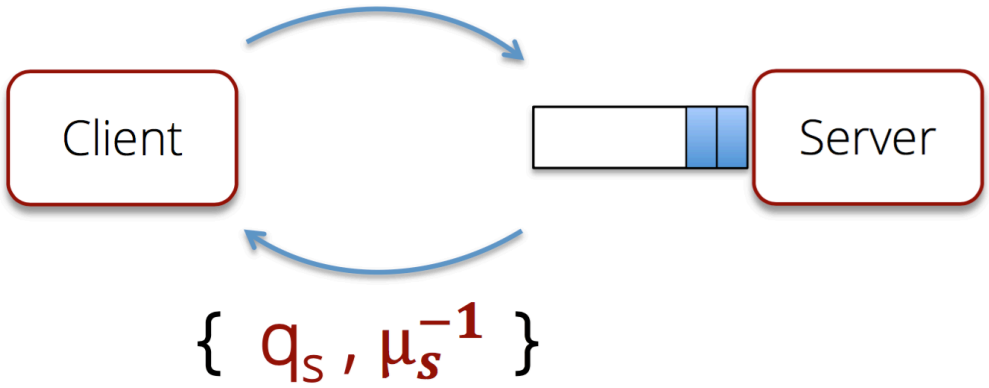


Balance product of queue-size and service time

{ $q \cdot \mu^{-1}$ }

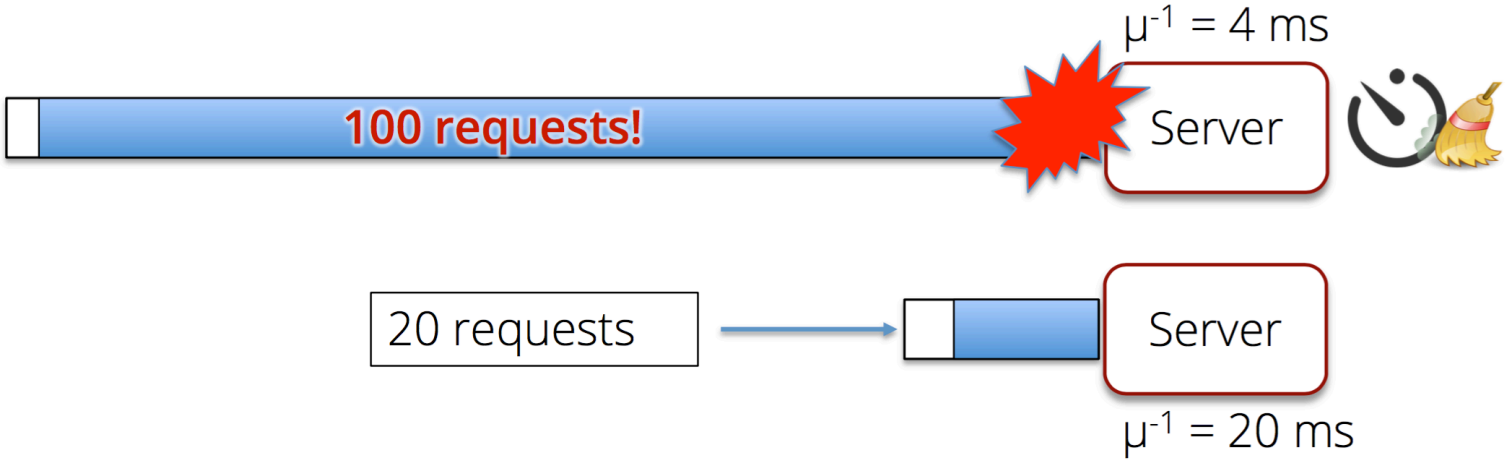
Server-side Feedback

- Servers piggyback **$\{q_s\}$** and **$\{u_s^{-1}\}$** in every response



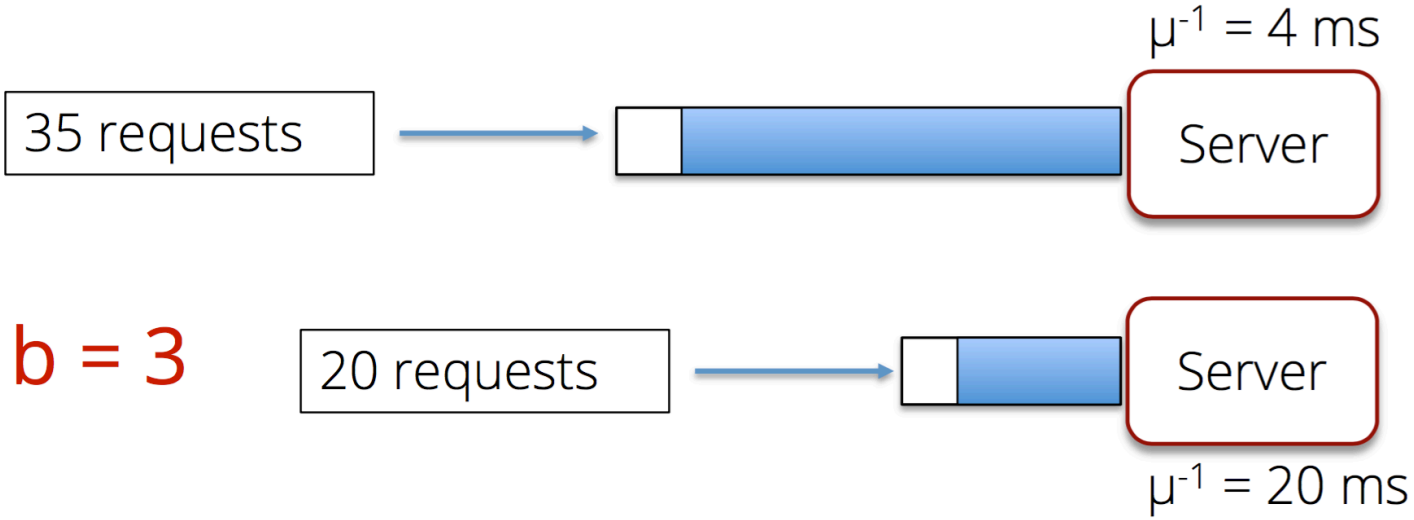
Select server with min q_s x u_s^{-1}

- Potentially long queue sizes
- What if a GC pause happens?

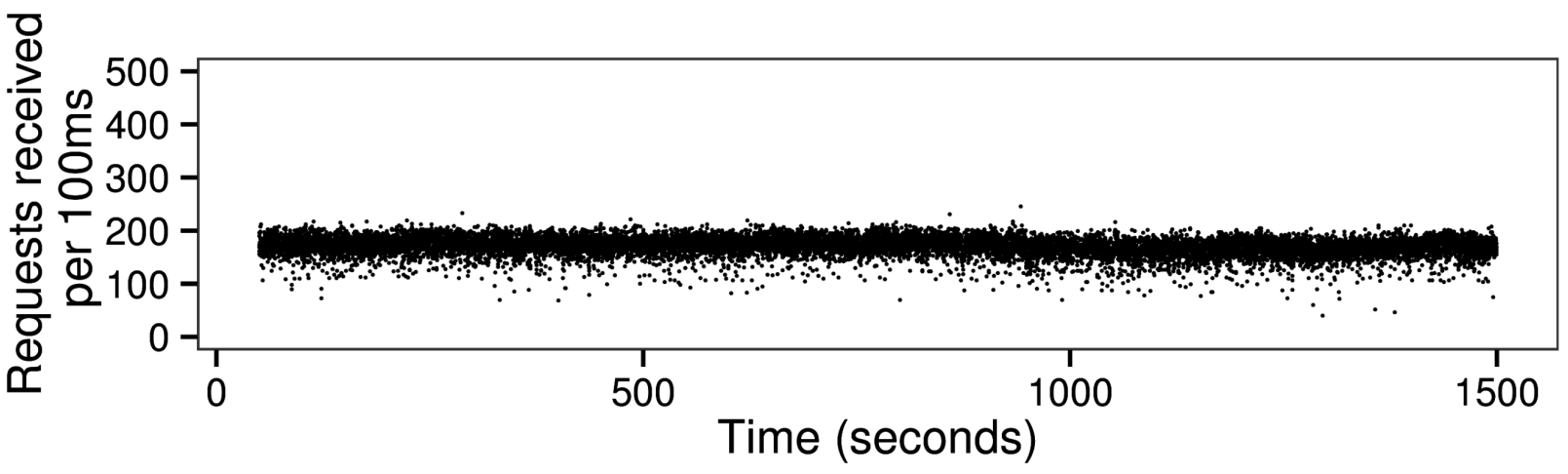


Penalizing Long Queues

- Select server with min $(q_s)^b \times u_s^{-1}$



Load Conditioning in C3



Conclusion

- Even rare performance hiccups affect a significant fraction of all requests in large-scale distributed systems
- Eliminating all sources of latency variability in large-scale systems is impractical, especially in shared environments
- Using an approach analogous to fault-tolerant computing, tail-tolerant software techniques form a predictable whole out of less-predictable parts