Low Latency via Redundancy

Ashish Vulimiri, Philip Brighten Godfrey, Radhika Mittal, Justine Sherry, Sylvia Ratnasamy, Scott Shenker

Presenter: Meng Wang
Low Latency Is Important

- Injecting just 400 milliseconds of artificial delay into Google search results caused the delayed users to perform 0.74% fewer searches after 4-6 weeks.
- A 500 millisecond delay in the Bing search engine reduced revenue per user by 1.2%, or 4.3% with a 2-second delay.
- Human-computer interaction studies similarly show that people react to small differences in the delay of operations.
Tail Latency Is Critical

Completion 30ms

Tail Latency!

Parallel task #1

Parallel task #2

10ms
Redundancy

- Initiate the **same** operation **multiple** times
- Use the **first** result which completes
- Overall latency is the **minimum** of two
- Reduces both **mean** and **tail** latency
Redundancy Is Eschewed

- Only employed in a few systems
  - Deal with failures in DTNs
  - In a multi-homed web proxy overlay
  - In limited cases in distributed job execution frameworks

- Reason for being eschewed:
  - Duplicate every operation doubles system utilization!
  - Default assumption in system design: Doing less work is best???
When to Use Redundancy

- **Tradeoff**
  - Benefit: receive the fastest of multiple options (*decreasing* latency)
  - Cost: double system utilization (*increasing* latency)

- **Queueing analysis**

- **Evaluation in different applications**

- **Guide to cost-effectiveness**
Queueing Analysis

Queueing model
- N independent, identical servers with same service time distribution
- Requests arrive in Poisson process
- Send each request to k servers, chosen uniformly at random

Assumptions
- Redundancy adds no penalty for clients
  - We will talk about client side overhead later
- Queues at servers evolve independently of each other
  - Not quite accurate!
  - Reasonable approximation when N is large
    - 3% inaccuracy in mean response time with N = 10
    - 0.1% inaccuracy in mean response time with N = 20
Queueing Analysis

- A first example of the effect of replication
  - Response time = queueing delay + service time
  - Threshold load
Queueing Analysis

- A first example of the effect of replication
  - P99.9 latency reduced by 5x
Queueing Analysis

Theorem 1: Within the independence approximation, if the service times at every server are i.i.d. exponentially distributed, the threshold load is 33%.

- Suppose mean service time = $1 \text{ sec}$, request arrival rate = $\rho$
- Without replication
  - $\text{M/M/1}$ queue with departure rate 1 and arrival rate $\rho$
  - Mean response time = \( \frac{1}{1-\rho} \)
- With replication
  - $\text{M/M/1}$ queue with departure rate 1 and arrival rate $2\rho$
  - Mean response time = \( \frac{1}{2(1-2\rho)} \)
- Benefit > Cost when \( \frac{1}{1-\rho} < \frac{1}{2(1-2\rho)} \), thus $\rho < \frac{1}{3}$
Queueing Analysis

- Threshold load **cannot exceed** 50% load in any system
  - Otherwise server load exceeds 100%
- We intuitively expect replication to help more as the service time distribution becomes more **variable**
Queueing Analysis

- Threshold load cannot exceed 50% load in any system
  - Otherwise server load exceeds 100%
- We intuitively expect replication to help more as the service time distribution becomes more variable
- The threshold load with deterministic service time turns out to be ~26%
- Conjecture 1: Deterministic service time is the worst case for replication: there is no service time distribution in which the threshold load is below the (≈ 26%) threshold when the service time is deterministic.
Queueing Analysis

Conjecture 1: Deterministic service time is the worst case for replication: there is no service time distribution in which the threshold load is below the (≈ 26%) threshold when the service time is deterministic.

- **Hard to prove**
  - General response time distribution is hard to handle analytically, when service time is not exponentially distributed
  - We need to understand the shape of the entire distribution, not just its first few moments
Queueing Analysis

- Approximations
  - Myers and Vernon: perform well for light-tailed service time distributions
    - Theorem 2: The threshold load is minimized when the service time distribution is deterministic.
  - Olvera-Cravioto et al.: heavy-tailed $\alpha < 1 + \sqrt{2}$
    - Theorem 3: If the service time distribution is regularly varying with tail index $\alpha < 1 + \sqrt{2}$, then the threshold load is $> 30\%$
Queueing Analysis

- Sampling from the space of all unit-mean discrete probability distributions with support \{1, 2, ..., S\}
  - Uniformly at random
  - Dirichlet distribution (higher variance)
Queueing Analysis

- Client-side overhead
  - CPU utilization
  - kernel processing
  - network overhead
  - ...

- May be comparable to latency
  - Large file transfers
  - Very quick memory accesses
Queueing Analysis

- Effect of client-side overhead
  - More variable distributions are more forgiving of overhead
  - Client side overhead must be at least somewhat smaller than mean request latency
Queueing Analysis

Summary of queueing analysis

- If redundancy adds no client-side cost, there is strong evidence to suggest that no matter what the service time distribution, the threshold load has to be more than 25%.
- In general, the higher the variability in the service-time distribution, the larger the performance improvement achieved.
- Client-side overhead can diminish the performance improvement due to redundancy. In particular, the threshold load can go below 25% if redundancy adds a client-side processing overhead that is significant compared to the server-side service time.
Disk-backed Database

- **Experiment setup**
  - Mean file size = 4 KB
  - File size distribution = deterministic
  - Cache:disk ratio = 0.1
  - 4 servers and 10 clients
  - P99.9 latency reduced by 2.2x

![Graph showing mean response time and 99.9th percentile response time with and without cache.]
Disk-backed Database

- Experiment setup
  - Pareto file size distribution
- Bottleneck: locate, rather than load
  - Data sizes are small
Disk-backed Database

- **Experiment setup**
  - Mean file size 0.04 KB
- **Bottleneck:** locate, rather than load
  - Data sizes are small
Disk-backed Database

- Experiment setup
  - Mean file size 400 KB
- Client extra overhead is heavy!
Disk-backed Database

- **Experiment setup**
  - Cache:disk ratio 0.01 instead of 0.1
  - Make service time more variable
- **P99.9 improvement grows**
  - 2.2x -> 2.5x
Disk-backed Database

- Experiment setup
  - Cache:disk ratio 2 instead of 0.1
- Service time is short
- Client overhead is comparable
Disk-backed Database

- Experiment setup
  - Run on EC2
- Service time is variable due to all kinds of contentions

![Graph showing mean response time and 99.9th percentile response time]

- Rate 1000 queries/sec/node: CDF
  - Fraction later than threshold
  - Response time (ms)
Memcache

- Experiment setup
  - Memcached in-memory database
Memcache

- Real vs Stub
  - Low load
  - Stub
    - No query processing
    - Replication increase 9% of real mean response time
  - The service time distribution is not very variable
Replication in the Network

- **Networks hotpots**
  - Many data centers assign network flows to paths based on hash
  - Hotpots: elephants flows can be assigned to same link
  - Can result in congestion
  - Current solution: dynamically re-assign flows in response to hotpots

- **Alternative Solution: Redundancy**
  - Replicates the first few packets of each flow along an alternate route
    - Aimed at short workflows
    - The completion times of large flows depend on their aggregate throughput rather than individual per-packet latencies
  - Replicated packets are assigned a lower priority than the original packets
Replication in the Network

- Evaluation (1/3)
  - obtain the largest improvement at intermediate loads
    - Low load: less congestion
    - High load: every path can be congested
  - Performance improvement falls as the delay-bandwidth product increases
    - Queueing delay makes up a smaller proportion of the total flow completion time
Replication in the Network

- Evaluation (2/3)
  - Great tail reduction at high load
  - TCP’s minRTO = 10ms
Replication in the Network

- Evaluation (3/3)
  - More improvement in medium, less in tail
  - They claim this is because tail occurs at those instants of high congestion when most of the links along the flow’s default path are congested.
Connection Establishment

- Without correlation
  - Replication should reduce the probability of it being lost from $p$ to $p^2$
  - 0.0048 to $\sim 10^{-6}$

- With correlation
  - 0.0048 to $\sim 0.0007$
  - Still reduces 7x

- Replication can be helpful!
Connection Establishment

- **Idealized network model**
  - 3-packet TCP handshake
  - Each packet can be delivered successfully after \((\text{RTT} / 2)\) seconds with probability \(1 - p\), and lost with probability \(p\).
    - \(p = 0.0048\) when sending one copy of each packet
    - \(p = 0.0007\) when sending two copies of each packet
  - Timeout = 3 seconds for SYN and SYN-ACK
  - Timeout = 3 x RTT for ACK

- **Improvement**
  - Reduces expected completion time by at least 25ms
  - Reduces p99.9 completion time by at least 880ms

- **Cost-benefit analysis**
  - Reducing latency is useful as long as it improves latency by 16 ms/KB
  - 50 bytes long per packet; 0.15KB for all 3 packets
  - 170-6000 ms/KB!
DNS

- Experiment setup
  - 10 DNS servers; 15 PlanetLab nodes
  - Two stage experiment
    - Rank all 10 DNS servers in terms of mean response time
    - Repeatedly pick a random name and perform a random one of 20 possible trials
      - No replication: one of 10 DNS servers
      - Replication: N copied to top N servers
  - Timeout = 2ms
Result: CDF figure

- Querying 10 DNS servers
  - The fraction of queries later than 500 ms is reduced by 6.5×
  - The fraction later than 1.5 sec is reduced by 50×.
DNS

- Result: latency reduction
  - Substantial with only 2 copies!
  - 50-62% reduction with 10 copies
DNS

How many servers should we use?

- We compare marginal increase in latency savings from each extra server against the 16 ms/KB benchmark
- 5 servers if we care more about mean
  - The absolute mean latency savings from sending 10 copies of every query is 0.1 sec / 4500 extra bytes ≈ 23 ms/KB
- 10 servers if we care more about tail
Conclusion

- Tradeoff between latency reduction and cost of overhead
- Abstract characterization: Queueing analysis
- Evaluations in different applications/scenarios
- Guide to cost-effectiveness
Questions

- Is there any benefit to replicating more than just once? For example can we perform the same analysis with 3x the original requests?
- If resources are highly used as explored in past papers (above the threshold load), will we ever get a chance to exploit redundancy?
- Can we perform partial redundancy (replicate 50% of the requests) to still achieve benefits when the load is above 50%?
- Will it add the load of the datacenters? For example, two datacenters may process the same request at the same time.