PriorityMeister: Tail Latency QoS for Shared Networked Storage

Timothy Zhu, Alexey Tumanov, Michael A. Kozuch, Mor Harchol-Balter, Gregory R. Ganger
SoCC 2014

Presenter: Yosub Shin
Motivation

- Internet companies want better tail latencies
- 99.9th or even 99.99th percentiles matter
- e.g. Displaying Facebook newsfeed: Requires ~1000 RPC calls. If 999 calls return in 50ms and one call takes 3s, the end-to-end response time = 3s.
Challenges

• Bursty workloads cause queuing
• End-to-end latency is affected by multiple stages
  • Outgoing network packets
  • Storage requests
  • Incoming network packets
Cake

- Cake: Reactive feedback-control built for tail latency
- Bad for bursty workloads
- Handles only one latency-sensitive workload
- Impossible to apply network QoS

Prior Works

Non-bursty workload

Bursty workload

Cake performs bad at burst

# of SLO violations high

PM always performs good
Enter PriorityMeister

• Proactive QoS(Quality of Service) system

• Achieves end-to-end latency SLO(Service Level Objective)

I want Workload A to respond in 30ms, Workload B to respond in 50ms.

• Multi-tenant, multi-resource

• How?

  • Priority

  • Rate Limiting
Contributions of This Paper

• Algorithm that automatically determines priority and rate limits for each workload at each stage

• Built a real QoS system consisting of network and storage which outperforms existing approaches

• Robust to mis-estimation of storage or network performance and workload mis-behavior
Priority & Rate Limiting

**Intuition**

e.g. I want Workload A to respond in 30ms, Workload B to respond in 50ms

- **Priority**: In order to meet tight latency requirements
  - NOT to be confused with importance to the user

- **Rate Limiting**: To prevent starvation of lower priority workloads
Intuition

Requests from Workload A gets higher priority

- One can proceed only if he throws token into bucket
- One can throw token into bucket only if it is not full
- Bucket leaks at rate ‘r’ in unit time.

Bucket is full!
Can’t throw, should wait...

Bucket is not full!
Can throw and proceed...

Requests from Workload A gets higher priority

Priority

NIC / Storage Device

mail Request

Token
Components

- Queue at each component of machine (Network, Storage)
- Each stage has independent priorities and rate limits
Architecture Overview

Workflow

1. Users
   - Application requirements:
     - latency/throughput SLO
     - trace of access patterns
     - location of client & server

2. PriorityMeister controller
   - Precomputed storage/network profile data
   - Rate limits

3. Prioritizer algorithm
   - Priority ordering
   - Rate limits
   - Latency estimates

4. Latency analysis model
   - QoS parameters:
     - Priority ordering
     - Rate limits

5. Storage/network enforcers
User initiates system with SLO requirements
Workflow

User initiates system with SLO requirements

Generate rate limits

Architecture Overview

Application requirements:
- latency/throughput SLO
- trace of access patterns
- location of client & server

PriorityMeister controller

Workload analyzer

Profile data

Precomputed storage/network profile data

Latency analysis model

Prioritizer algorithm

QoS parameters:
- priority ordering
- rate limits

Storage/network enforcers

Rate limits

User

Workflow

User initiates system with SLO requirements

Generate rate limits

Determine priority orderings
Workflow

1. User initiates system with SLO requirements
2. Generate rate limits
3. Determine priority orderings
4. Send priorities and rate limits to enforcers
How to Limit Rates?

Leaky Token Bucket Model

- Token(s) == size of request
  - Storage: Amount of storage time required
  - Network: Number of transmitted bytes

- \((r, b)\) pair determines bucket’s behavior

- \(r\): leaking rate

- \(b\): bucket size(in #tokens)

- Throwing a new token into bucket only allowed when bucket not full
Workload Analysis

Assume highest priority for $W_A$, calculate $(r, b)$ pairs big enough for the trace to run under SLO.

Want to decide smallest $(r, b)$ pair s.t. lower priority workloads are allowed to run under their SLOs.

Larger bucket size ($b$) leads to higher tail latency in medium priority workload $W_B$.

Larger rate ($r$) leads to higher tail latency in low priority workload $W_C$.

Key Idea: Use multiple $(r, b)$ pairs on the blue line and allow throwing tokens into bucket only when tokens can be added to all $(r, b)$ buckets.
How to determine priorities?

Workflow

Determine priority orderings
Prioritizer Algorithm

• Input: workload SLOs, rate limits

• Output: priorities for each stage at each workload s.t. each workload’s estimated worst-case latency is less than SLO via Latency Analysis Model

• \(|(# \text{ workloads})|^{|(# \text{ stages})|} \) possibilities: too large 😞

• Polynomial search possible (w/ greedy algorithm)! ☺

1. Assign lowest priority workload first! (If workload can still satisfy SLO)

2. For unassigned workload w/ lowest violation:
   \((\text{estimated latency}) - (\text{SLO})\)

   • For stage w/ lowest latency, assign lowest priority
     (Intuition: Take best performing workload/stage, assign lowest priority s.t. worst-case latency is improved)
How to estimate worst-case latencies given priorities?

Latency Analysis Model

• Input: priorities assignment, rate limits

• Output: **worst-case latencies for each stage at each workload**

• \( \alpha(t) \): max. # bytes that arrive in any period of time \( t \)

• \( \beta(t) \): min. # bytes serviced in any period of time \( t \)

• Worst-case latency: max. horizontal dist. \( \alpha \) and \( \beta \)

• \( \alpha_w(t) = \min_i(r_i* t + b_i) \) (fastest rate at which rate limiter allows requests through)

• \( \beta_w(t) \): Calculated with Linear Programming
  (time, flow, rate limit, and work conservation constraints)
Experiments

• Tail latency performance
• Latency under bursty workloads
• Mis-behaving workloads
• Network bottlenecked workloads
  • Latency under estimator inaccuracy
  • Latency under varying SLO permutations
Experiments

Tail Latency Performance (1/2)

Only PriorityMeister always satisfies SLO

<table>
<thead>
<tr>
<th>PM</th>
<th>PriorityMeister</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cake</td>
<td>Reactive Control</td>
</tr>
<tr>
<td>bySLO</td>
<td>Priority for lower SLO</td>
</tr>
<tr>
<td>EDF</td>
<td>Earliest Deadline First</td>
</tr>
<tr>
<td>PS</td>
<td>Proportional Sharing</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Workload</th>
<th>Workload A</th>
<th>Workload B</th>
<th>Workload C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trace</td>
<td>Display Ad</td>
<td>MSN Storage</td>
<td>LiveMaps</td>
</tr>
</tbody>
</table>
Experiments

Tail Latency Performance (2/2)

(a) PM: all workloads’ SLOs satisfied
(b) Cake (reactive): blue SLO violated @ 84th-%ile
(c) bySLO: green SLO violated @ 91st-%ile

(d) EDF: green SLO violated @ 97th-%ile
(e) PS: blue SLO always violated

All policies except PM violates SLOs
Experiments

Latency Under Bursty Workload

Both satisfies SLO at 99%

Cake violates SLO > 99%, while PM satisfies SLO

Under extreme burstiness, both fails to satisfy SLO

\[ C_A^2 \]: Squared coefficient of variation of inter-arrival times
Higher value means burstier workload
Experiments

Mis-behaving Workloads

Initially both workloads satisfy SLOs.

Workload D mis-behaves (hogs network)

All policies except PM no longer satisfy SLO.
Experiments

Network Bottlenecked Workloads

PM assigns workload K lower network priority thus improving Workload C and J’s tail latencies

Scenario: Workload K runs on Ramdisk (better storage latency), and others run normal disk
SLO Level (C > J > K)
PriorityMeister

- Proactive end-to-end tail latency QoS system
- Combines *priorities* and *rate limits*
- Automatically configures itself
- Performs well under real world bursty workloads
Comments

- Pros
  - Unique system that provides good tail latency
  - Allows multi-tenant, multi-resources
  - Extensive experiments

- Cons
  - Prior computation of trace is not always possible
  - No mention on fault tolerance
  - No mention on how to ensure throughput SLO
Discussion

• What do we lose at cost of better tail-latency?

• What semantic meanings does rate\( (r) \) and bucket size\( (b) \) of leaky token bucket model have?

• Is it possible to combine reactive\( (\text{Cake}) \) and proactive\( (\text{PriorityMeister}) \) approach? Would it perform better than both?

• Experiment shows bySLO\( (\text{just assigning higher priority for lower SLO}) \) performs really well. Why so?
Thank You!
Backup Slides
How enforcers work?

Enforcers

• Storage enforcer
  • # tokens == Amount of storage time consumed by request
  • Queues on top of NFS

• Network enforcer
  • # tokens == Number of transmitted bytes by request
  • Enforce priority with network QoS level
Latency Analysis Model

**Linear Programming**

- Estimate $\beta_w(t)$: Maximize interference with higher priority workloads
- Let’s estimate $t = \beta_w^{-1}(y)$ instead
- For queue $q$, define $t_{in}^q, t_{out}^q, R_k^q, R_k'^q$

**Time constraints**

$t_{in}^q \leq t_{out}^q \quad t_{out}^q = t_{in}^q$

**Flow constraints**

$R_k^q \leq R_k'^q$

**Rate limit constraints**

$R_k'^q - R_k^q \leq r_i \times (t_{out}^q - t_{in}^q) + b_i$, $q^*$: workload $k$’s first queue

**Work conservation constraints**

$\sum_k (R_k'^q - R_k^q) = B_q \times (t_{out}^q - t_{in}^q)$

**Objective function:** $\max(t_{out}^{q_n} - t_{in}^{q_1})$, where $(R_w'^{q_n} - R_w^{q_1}) = y$
# Experiments

## Traces

<table>
<thead>
<tr>
<th>Workload label</th>
<th>Workload source</th>
<th>Estimated storage load</th>
<th>Estimated network load</th>
<th>Interarrival Variability, $C_A^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workload A</td>
<td>DisplayAds production trace</td>
<td>5%</td>
<td>5%</td>
<td>1.3</td>
</tr>
<tr>
<td>Workload B</td>
<td>MSN storage production trace</td>
<td>5%</td>
<td>5%</td>
<td>14</td>
</tr>
<tr>
<td>Workload C</td>
<td>LiveMaps production trace</td>
<td>55%</td>
<td>5%</td>
<td>2.2</td>
</tr>
<tr>
<td>Workload D</td>
<td>Exchange production trace (behaved)</td>
<td>10%</td>
<td>5%</td>
<td>23</td>
</tr>
<tr>
<td>Workload E</td>
<td>Exchange production trace (misbehaved)</td>
<td>&gt; 100%</td>
<td>15%</td>
<td>145</td>
</tr>
<tr>
<td>Workload F</td>
<td>Synthetic low burst trace</td>
<td>25%</td>
<td>5%</td>
<td>1</td>
</tr>
<tr>
<td>Workload G</td>
<td>Synthetic high burst trace</td>
<td>25%</td>
<td>5%</td>
<td>20</td>
</tr>
<tr>
<td>Workload H</td>
<td>Synthetic very high burst trace</td>
<td>25%</td>
<td>5%</td>
<td>40</td>
</tr>
<tr>
<td>Workload I</td>
<td>Synthetic medium network load trace 1</td>
<td>35%</td>
<td>20%</td>
<td>1</td>
</tr>
<tr>
<td>Workload J</td>
<td>Synthetic medium network load trace 2</td>
<td>45%</td>
<td>25%</td>
<td>1</td>
</tr>
<tr>
<td>Workload K</td>
<td>Synthetic ramdisk trace</td>
<td>N/A</td>
<td>35%</td>
<td>3.6</td>
</tr>
<tr>
<td>Workload L</td>
<td>Synthetic large file copy</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Other QoS Policies

- Proportional sharing (ps)
  - Each workload gets equal share of storage time
- Cake
  - Dynamically adjust proportional shares to meet latency SLOs
- Earliest Deadline First (EDF)
  - Deadline = workload's SLO
- Prioritization by SLO (bySLO)
  - Simply assign workload priorities in order of workload latency SLOs
Experiments

Latency Under Estimator Inaccuracy

**Accurate Estimator** (same as tail latency performance experiment)

**Inaccurate Estimator** (Token counting does not reflect reality well)

PM works well for both accurate / inaccurate estimators
Experiments

Latency Under Varying SLO Permutations

For different permutations of SLO levels, only PM satisfies SLOs for all permutation