Maestro:
Quality of Service in Large Disk Arrays

Arif Merchant
Mustafa Uysal
Pradeep Padala
Xiaoyun Zhu
Sharad Singhal
Kang G. Shin

Google
vmware
docomo USA Labs
vmware
hp Labs
University of Michigan
Storage consolidation and QoS

- Reduced costs
- High utilization
- Better performance

Storage QoS management = QoS + consolidation
Meet desired level of service while sharing resources
A typical multi port shared storage system

Goal:
Ther > 1000 req/sec

Goal:
RT < 100ms

N sets of per-App I/O queues

N port QoS schedulers

N I/O ports

Cache and low-level controllers

Shared Disk Array
Port QoS scheduler

- Provides **concurrency knob**
- Limits back-end IO requests per application id
- Higher concurrency ⇒ higher throughput

**Concurrencies:**
- App1=4
- App2=2
How does the concurrency knob work?

Conc = 2
Port I Scheduler
No more IOs before at least 1 request finishes

Conc = 1
Port II Scheduler
No more IOs until 1 request finishes

Back end controllers

Concurrency allows us to control max outstanding requests from a queue
How to achieve application targets?

**Throughput targets**
- **App1**
  - Thr > 3000 req/sec
- **App2**
  - Thr > 1000 req/sec

**Response time targets**
- **App1**
  - Rt < 100ms
- **App2**
  - Rt < 50ms

**Combination**
- **App1**
  - Rt < 100ms
- **App2**
  - Thr > 1000 req/sec

Start with concurrencies in 3:1 ratio, adjust.

How to set the knobs?
Should we set them 1:2 or 1:4?

No intuition

How to set knobs if we wanted to prioritize?

How to set knobs for multiple ports?
Enter *Maestro*

- Meets performance targets automatically
  - Adaptive Feedback Controller
  - Throughput or latency targets; priorities

- Goals
  - If possible meet all targets (underload)
  - In overload, degrade in inverse proportion to priority
  - Example:

<table>
<thead>
<tr>
<th></th>
<th>Priority</th>
<th>Target</th>
<th>Underload</th>
<th>Overload</th>
</tr>
</thead>
<tbody>
<tr>
<td>App1</td>
<td>2</td>
<td>1000 IO/s</td>
<td>&gt; 1000 IO/s</td>
<td>800 IO/s (20% degradation)</td>
</tr>
<tr>
<td>App2</td>
<td>1</td>
<td>3000 IO/s</td>
<td>&gt; 3000 IO/s</td>
<td>1800 IO/s (40% degradation)</td>
</tr>
<tr>
<td>App3</td>
<td>4</td>
<td>50 ms</td>
<td>&lt; 50 ms</td>
<td>55 ms (10% degradation)</td>
</tr>
</tbody>
</table>
Outline

• Motivation and background
• Design
  – Application controller
  – Single port controller
  – Multi port controller
• Evaluation
  – Experimental setup
  – Methodology
  – Evaluation metrics
  – Results
• Previous work
• Our contributions
Maestro – inside the box

Sensors

App 1 controller
App 2 controller
App 3 controller
App M controller

Requested allocations

Arbiter

Global allocations

Port Allocator

Per-port allocations

Priorities

Demands

Maestro

Port 1
Port 2
Port N

Disk array

QoS Scheduler
QoS Scheduler
QoS Scheduler
Application controller

Computes **concurrency values** to achieve particular target for a single application

Per application:

1. Build linear regression model
   Concurrency ➔ Performance.

2. Compute required concurrency for target performance.

3. Truncate change from current if too large.
Arbiter: How to allocate among competing apps?

For throughput target:
Relative error = 1 - \frac{\text{Throughput estimate}}{\text{Throughput target}}

Minimize
\[
\max_{i,j} \left( P_i^{\text{relative error}}(i) - P_j^{\text{relative error}}(j) \right)
\]
(Equalize priority-weighted relative errors)

Compute global concurrences for all applications
How to distribute for multiple ports?

• Demand of app: **average outstanding requests** of app per port
• Distribute concurrency among multiple ports using **demand**

Global concurrency for app = \( G \)

Demand of app through port (1) and (2) = \( d_1, d_2 \)

Concurrency for app port (1) = \( \frac{d_1}{d_1 + d_2} \times G \)

Concurrency for app port (2) = \( \frac{d_2}{d_1 + d_2} \times G \)

Distribute **in proportion of** application demands
Outline

• Motivation and background
• Design
  – Application controller
  – Single port controller
  – Multi port controller
• Evaluation
  – Experimental setup
  – Methodology
  – Evaluation metrics
  – Results
• Previous work
• Our contributions
Experimental setup

- Applications
- Shared storage
- Cache and low-level controllers
- Port QoS Scheduler
- Port QoS Scheduler
- Port QoS Scheduler
- Maestro controller

HP XP 1024 Disk Array

- 16 ports
- ~40 LUNs
- spread across
- ~160 disks
Evaluation

• Workloads:
  • Synthetic
    • Intensity – 5 to 100 threads, 16KB IOs (90% read)
    • Port affinity – accessing multi ports non-uniformly
    • Throughput and response time targets
    • Changing loads and priorities
  • Real world application traces: SAP and TPCC

• What are we evaluating?
  • Functionality: Whether Maestro achieves targets
  • Robustness: How it adapts to changing conditions
  • Real workloads: Does it work for real?
Functionality: performance differentiation

- Both workloads initially meet targets
- One workload increases load halfway through
- Both degraded proportionately (equal priorities)
Mixed target metrics

- **Light** target: 25MB/s (normalized perf. = throughput/target)
- **Change** target: 10ms (normalized perf. = target/latency)
- **Change** workload intensity decreases in steps
- Both degraded proportionately in overload
• High priority *light* workload protected from *burst*
• Bursts throttled after a few seconds
• OLTP (TPCC) + Decision support (SAP)
• Uncontrolled: SAP workload misses target by 10x
• Maestro: Targets met in 80% of intervals
Related work

• Relative throughput sharing – no latency control
  ✓ Proportional sharing: SFQ(D), WFQ, pClock
  ✓ Throughput reservations: Zygaria, AQvA, mClock
• Latency-based congestion control: PARDA
• Latency targets
  ✓ Façade, Triage, SLED
  ✓ Proportional share + latency: SARC-Avatar
• Performance Insulation (fixed fraction of solo performance)
  ✓ Argon, Fahrrad
• Adaptive feedback controllers: many in diverse areas
Summary

• Maestro: differentiated performance controller for disk arrays
  ✓ Latency and throughput targets
  ✓ Priorities apply when not all targets can be met
  ✓ Easy, intuitive specification of requirements

• Accurate, stable, robust performance in tests
  ✓ Evaluated on high-end array with synthetic & application loads
  ✓ OLTP/Decision support workloads hit targets 80% of time where uncontrolled system missed by 10X

• Released in a commercial product
Backup slides
Some open problems

• What if workload is not controllable?
  • If workload offers too low load, or is erratic, may be impossible to meet target
  • Need to ensure that it doesn’t badly affect other workloads
  • Optimizer has tendency to give more and more resource to ill-performing workload

• What if performance targets are unachievable?
  • Optimizer assumes that targets are reasonable
  • If user specifies impossible target, controller should complain

• Better performance predictions
  • Current predictor (regression-based) is only so-so
  • Better predictor would allow faster response to changes