PARDA: Proportional Allocation of Resources for Distributed Storage Access

Ajay Gulati, Irfan Ahmad, Carl Waldspurger

Resource Management Team
VMware Inc.

USENIX FAST 09 Conference – February 26, 2009
The Problem

- online store
- Microsoft Exchange
- Data Mining (low priority)
- Storage Array LUN
The Problem

What you see

- Online store
- Microsoft Exchange
- Data Mining (low priority)
- Storage Array LUN
The Problem

What you see

- Online store
- Microsoft Exchange (low priority)
- Data Mining

What you want to see

- Online store
- Microsoft Exchange (low priority)
- Data Mining

Storage Array LUN
Distributed Storage Access

Setup
- VMs running across multiple hosts
- Hosts share LUNs using a cluster filesystem
- No central control on IO path

Issues
- Hosts adversely affect each other
- Difficult to respect per-VM allocations
  - Proportional shares (aka “weights”)
  - Specify relative priority

**Goal:** Provide isolation while maximizing array utilization
Host-local Scheduling Inadequate

Local SFQ schedulers respect VM shares BUT not across hosts
Host-local Scheduling Inadequate

- Local SFQ schedulers respect VM shares
  BUT not across hosts

- Hosts get equal IOPS
  ⇒ IOPS dependent on VM placement!
Outline

- Problem Context
- **Analogy to Network Flow Control**
- Control Mechanism
- Experimental Evaluation
- Conclusions and Future Work
## Analogy to Network Flow Control

<table>
<thead>
<tr>
<th>Networks</th>
<th>Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network is a black box</td>
<td>Array is a black box</td>
</tr>
<tr>
<td>Network congestion detected using RTT, packet loss</td>
<td>Estimate array congestion using IO latency</td>
</tr>
<tr>
<td>TCP adapts window size</td>
<td>Packet loss very unlikely</td>
</tr>
<tr>
<td>TCP ensures fair sharing</td>
<td>Adapt number of pending IO requests (a.k.a. window size)</td>
</tr>
<tr>
<td>FAST TCP* proportional sharing</td>
<td>Adapt window size based on shares/weights</td>
</tr>
</tbody>
</table>

Outline

- Problem Context
- Analogy to Network Flow Control
- Control Mechanism
- Experimental Evaluation
- Conclusions and Future Work
Queue lengths varied dynamically
Per-Host Control Algorithm

\[ w(t + 1) = (1 - \gamma)w(t) + \gamma \left( \frac{L}{L(t)} w(t) + \beta \right) \]

- Adjust window size \( w(t) \) using cluster-wide average latency \( L(t) \)
  - \( L \) : latency threshold, operating point for IO latency
  - \( \beta \) : proportional to aggregate VM shares for host
  - \( \gamma \) : smoothing parameter between 0 and 1
- Motivated by FAST TCP mechanism
Control Algorithm Features

\[ w(t + 1) = (1 - \gamma)w(t) + \gamma \left( \frac{L}{L(t)} w(t) + \beta \right) \]

- Maintain high utilization at the array
  - Overall array queue proportional to Throughput x \( L \)
- Ability to allocate queue size in proportion to hosts’ shares
  - At equilibrium, host window sizes are proportional to \( \beta \)
- Ability to control overall latency of a cluster
  - Cluster operates close to \( L \) or below
Unit of Allocation

- Two main units exist in literature
  - Bandwidth (MB/s)
  - Throughput (IOPS)

- Both have problems
  - Using bandwidth may hurt workloads with large IO sizes
  - Using IOPS may hurt VMs with sequential IOs

- PARDA: allocate queue slots at array
  - Carves out array queue among VMs
  - Workloads can recycle queue slots faster or slower
  - Maintains high efficiency
Storage-Specific Issues

Issues
- Throughput varies by 10x depending on workload characteristics
- IO sizes may vary by 1000x (512B – 512KB)
- Array complexity: caching, different paths for read vs. write
- Hosts may observe different latencies

PARDA Solutions
- Latency normalization for large IO sizes
- Compute cluster-wide average latency using a shared file
Handling Bursts—Two Time Scales

PARDA Slots Assignment

All VMs active

VM 1
VM 2
VM 3
VM 4
Handling Bursts—Two Time Scales

- Workload variation over short time periods
  - Handled by existing local SFQ scheduler
  - No strict partitioning of host-level queue
Handling Bursts—Two Time Scales

- Workload variation over short time periods
  - Handled by existing local SFQ scheduler
  - No strict partitioning of host-level queue

- VM idleness over longer term
  - Re-compute $\beta$ per host based on VM activity
  - Effectively scale VM shares based on its utilization
  - Utilization computed as
    \[ \frac{\text{(# outstanding IOs)}}{\text{(VM window size)}} \]
Outline

- Problem Context
- Analogy to Network Flow Control
- Control Mechanism
- **Experimental Evaluation**
- Conclusions and Future Work
Experimental Setup

- **VMware Cluster**
  - 1-8 hosts running ESX hypervisor
  - Each host: 2 Intel Xeon dual cores, 8 GB RAM

- **FC-SAN attached Storage**
  - EMC CLARiiON CX3-40 storage array
  - Similar results on NetApp FAS6030

- **Two volumes used**
  - 200 GB, 10 disks, RAID-0 (striped)
  - 400 GB, 5-disk, RAID-5
PARDA: Proportional Sharing

Setup:
• 4 Hosts, shares – 1 : 1 : 2 : 2
• Latency threshold \( L = 25 \text{ms} \)
• Workload – 16KB, 100% reads, 100% random IO

Window sizes are in proportion to shares
Latency close to 25 ms
IOPS match shares but affected by other factors

Aggregate IOPS with/without PARDA (3400 vs 3360)
PARDA: Dynamic Load Adaptation

**Setup:**
- 6 Hosts, equal shares, uniform workload
- Latency threshold $L = 30$ ms
- Three VMs are stopped at 145, 220 and 310 sec

**PARDA adapts to load**
- Latency close to 30 ms
- IOPS increase with increase in window size

Aggregate IOPS with/without PARDA (3090 vs 3155)
With PARDA:
Shares are respected
*independent of VM placement*

Setup:
Latency threshold $\mathcal{L} = 25$ms

![Diagram showing VM shares and host shares with PARDA](image)
**PARDA: Handling Bursts**

**Setup:**
- One VM idles from 140 sec to 310 sec

**Result:**
- PARDA is able to adjust $\beta$ values at host
- No undue advantage given to VMs sharing the host with idle VM
PARDA: SQL Workloads

Setup:
- 2 Hosts, 2 Windows VMs running SQL server (250 GB data disk, 50 GB log disk)
- Shares 1:4
- Latency threshold $L = 15$ ms

Without PARDA:
- No Fairness Across hosts
- Both hosts get ~600 IOPS

With PARDA:
- Shares are respected across hosts
- Host 1,2 with shares 1:4 get 6952 and 12356 OPM
PARDA: Burst Handling

Result:
- PARDA adjusts $\beta$ values at host with the change in pending IO count
- VM2 with high shares is unable to fill its window
- IOPS and OPM are differentiated based on $\beta$ values
Evaluation Recap

- Effectiveness of PARDA mechanism
  - Fairness
  - Load or throughput variations
  - Burst handling
  - End-to-end control
  - Enterprise workloads

- Evaluation of control parameters (without PARDA)
  - Latency variation with workload
  - Latency variation with overall load
  - Queue length as control mechanism for fairness
Outline

- Problem Context
- Analogy to Network Flow Control
- Control Mechanism
- Experimental Evaluation
- Conclusions and Future Work
Conclusions and Future Work

> PARDA contributions

- Efficient distributed IO resource management – without any support from storage array
- Fair end-to-end VM allocation, proportional to VM shares
- Control on overall cluster latency

> Future work

- Latency threshold estimation or adaptation?
- Detect and handle uncontrolled (non-PARDA) hosts
- NFS adaptation
Questions ...

{agulati, irfan, carl}@vmware.com

Storage in Virtualization BoF tonight @7:30 pm