IOFlow
A Software-Defined Storage Architecture

Jakub Łuczyński

October 28, 2015
So, what is the problem?

- how to enforce high level flow policies
- we want to dictate storage IO
  - performance (bandwidth/tenant guarantee)
  - routing (additional processing middlebox)
Scope

- Enterprise Data Center
- Compute Servers and Storage Servers
- virtualized (VMs, VHDs)
- small/medium Data Centers
  - \( \sim 10 \) SS, \( \sim 100 \) CS, 8-16 VMs/CS
  - \( O(thousand) \) VMs
- scaling deferred to future work
- assumed performance bottleneck in storage
{[set of VMs], [set of storage shares]} \rightarrow Policy

<table>
<thead>
<tr>
<th></th>
<th>Policy</th>
<th>Where to enforce?</th>
<th>What to enforce?</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>{p, X} \rightarrow B</td>
<td>C(p) Or S(X)</td>
<td>Static rate limit</td>
</tr>
<tr>
<td>P2</td>
<td>{p, X} \rightarrow Min B</td>
<td>C(p) Or S(X)</td>
<td>Dynamic rate limit</td>
</tr>
<tr>
<td>P3</td>
<td>{p, X} \rightarrow Sanitize</td>
<td>C(p) Or S(X)</td>
<td>Static routing</td>
</tr>
<tr>
<td>P4</td>
<td>{p, X} \rightarrow Priority</td>
<td>C(p) &amp; S(X)</td>
<td>Static priority</td>
</tr>
<tr>
<td>P5</td>
<td>{[p, q, r], [X, Y]} \rightarrow B</td>
<td>C(p), C(q) &amp; C(r) Or S(X) &amp; S(Y)</td>
<td>Dynamic VM Or Server rate limits</td>
</tr>
</tbody>
</table>

**Figure:** C(p) - Compute Server, S(X) - Storage Server
Challenges

- Differentiated treatment
- Flow name resolution
- Distributed enforcement
- Dynamic enforcement
- Admission control

Summarizing

- Traffic differentiation and global visibility at control plane.
- This motivates controller base design
Design

One IO path (VM-to-Storage)

Hypervisor

VM

Compute Server

Storage Server

Guest OS

VMn

VM1

Application

TCP

File system

IP

Block device

VNIC

VHD

SMBc

Network driver

Physical NIC

Controller

SMBs

Network driver

Disk driver

Physical NIC

IO Packets

Queue 1

Queue n

Stage control API
- centralized controller
- control apps
  - translate policies to state configurations (queueing rules)
- stages - layers implementing API (7 calls)
- stages in IOFlow prototype
  - SMBc at Hypervisor
  - SMBs at Storage Server
  - network drivers
- SMB over RDMA (client-server)
traffic differentiation through queues
maps each IO request to queue
queue control
  how fast requests are serviced (service)
  next stage IO requests are routed to (routing)
Controller

- discovers and interacts with the stages
- maintain topology graph
- exposes those to control applications
IOFlow’s API for stage

<table>
<thead>
<tr>
<th>A0</th>
<th><code>getQueueInfo()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>returns kind of IO header stage uses for queuing,</td>
</tr>
<tr>
<td></td>
<td>the queue properties that are configurable,</td>
</tr>
<tr>
<td></td>
<td>and possible next-hop stages</td>
</tr>
<tr>
<td>A1</td>
<td><code>getQueueStats(Queue-id q)</code></td>
</tr>
<tr>
<td></td>
<td>returns queue statistics</td>
</tr>
<tr>
<td>A2</td>
<td><code>createQueueRule(IO Header i, Queue-id q)</code></td>
</tr>
<tr>
<td></td>
<td>creates queuing rule $i \rightarrow q$</td>
</tr>
<tr>
<td>A3</td>
<td><code>removeQueueRule(IO Header i, Queue-id q)</code></td>
</tr>
<tr>
<td>A4</td>
<td><code>configureQueueService(Queue-id q,</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;token rate,priority, queue size&gt;</code>)</td>
</tr>
<tr>
<td>A5</td>
<td><code>configureQueueRouting(Queue-id q,</code></td>
</tr>
<tr>
<td></td>
<td>Next-hop stage $s$)</td>
</tr>
<tr>
<td>A6</td>
<td><code>configureTokenBucket(Queue-id q,</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;benchmark-results&gt;</code>)</td>
</tr>
</tbody>
</table>
Example

Single policy:
- \{VM 4, Share X\} → Bandwidth B

Calls at SMBc stage:

1: `getQueueInfo();`  returns “File IO”
2: `createQueueRule(<VM 4, //server X/\*\>, Q1)`
3: `createQueueRule(<\*, \*\>, Q0)`
4: `configureQueueService(Q1, <B, 0, 1000>)`
5: `configureQueueService(Q0, <C-B, 0, 1000>)`
Design goals

- stages
  - fast, cause minimal performance degradation

- control plane
  - flexible
  - responsive,
  - accurate,
  - resilient,
  - scalable

- not to require any application or VM changes
Stages

1. IO Header <VM1, //server X/file F> → Queue Q1
2. IO Header <VM2, //server Y/*> → Queue Q2
3. IO Header <VM3, *> → Queue Q4
4. <*, *> → Queue Q3

Figure: example SMBc quering rule

1. IO Header <SID S1, H:/File F> → Queue Q1
2. IO Header <SID S2, H:/File G> → Queue Q1
3. IO Header <SID S2, H:/Directory A/*> → Queue Q2
4. <*, *> → Queue Q3

Figure: example SMBs quering rule

- queuing rules checked in order they were created
- default rule checked last
- if no match exists, the request is blocked and controller is requested for matching rule
Stages

- Queues and queuing rules are soft state.
- Throttle or treat IO requests preferentially.
- Queues are token-bucket abstraction.
- Properties (configureQueueService) <token rate, priority, queue size>.
- Peculiarities of IO requests:
  - Different operation types (create, read, write).
  - Processing at storage-backend is variable.
  - \( f(\text{operation type, tokens}) \) may not be linear.
  - Other factors: data locality, device type..
Some Solutions

- controller discovery component
  - benchmarks the storage devices
  - done periodically
  - configureTokenBucket()

- split requests
  - reduces performance uncertainty
- present information to control applications (discovery)
- converts policies into stage-specific configurations
- flow name resolution
- soft state
- keeps track of version number
- request ordering (not changed, only delayed)
2 control applications implemented

- performance control (P2, P4, P5)
- routing control (P3)
Let’s take P5 as an example:

\[ \{[p, q, r], [X, Y]\} \rightarrow B \]

- admission control

![Diagram showing VMs, stages, shares, controller, and traffic](image)

- Queue for IOs from VM r to shares X and Y. Drain Rate \( R_r \)

- Controller collects traffic stats and returns per-VM rate

\[ R_p + R_q + R_r = B \]
Algorithm 4.1 Controller-based distributed rate limiting

Require: $N$ VMs with aggregate guarantee $B$; $D$: set of VM demands sorted in ascending order ($D_i > 0$); VM $i$’s IOs are queued at $q_i$

Ensure: Assign rate $R_i$ to VM $i$ s.t. $\sum R_i = B$

1: $leftB = B$ // bandwidth left to distribute
2: for $i$ in $[0, N-1]$ do
3: if $D_i \leq \frac{leftB}{N-i}$ then
4: $R_i = D_i$ // VM demand is less than fair share
5: else
6: $R_i = \frac{leftB}{N-i}$ // VM demand is more than fair share
7: $leftB -= R_i$

8: {share any left bandwidth and configure queues}
9: for $i$ in $[0, N-1]$ do
10: $R_i += leftB / N$
11: configureQueueService($q_i, < R_i, 0, 1000 >$)
Performance Control

- dynamic enforcement
- demand estimation (periodically)
- VMs with no IOs are estimated to low value (not 0)

- priority
  - requires support from all stages
  - IOFlow tolerate layers not being stages (ex SSD)

90 requests to utilize 95% throughput
Routing Control

- malware scanning middlebox
- scanning stage is optional (untrusted VMs)
Transient Controller Failure

- no impact on system correctness
- can lead to temporary degraded performance
- default policy in case the controller is unreachable
  ex. best effort queue, no scan etc
Implementation

- Windows-based IO stack
- drivers (C, 22 kLOC)
  - 2 storage drivers (SMBc SMBs)
  - network driver
  - malware scanning device driver
  - optional guest os driver (ioctl trick)
- controller (C#, 3 kLOC), single server
Evaluation

Figure: 4 tenants using 120 VMs in total across 10 hypervisors with policies. When IOFlow is disabled tenant policies B are not met. With IOFlow enabled tenant policies are met (“actual” ≥ B). Furthermore, extra capacity is assigned in proportion to the tenant minimum bandwidth.
Evaluation

Figure: Performance overheads of IOFlow when compared to unmodified storage stack. Error bars show minimum and maximum values from 5 runs. Note that y-axis is different for each graph.
Figure: Average flow creation latency. Error bars show minimum and maximum values from 5 runs.
Figure: Memory and network overheads associated with creating flows and getting flow statistics.