Cloud Storage 2

15-719 Advanced Cloud Computing

Garth Gibson
Greg Ganger
Majd Sakr

Cloud storage options

- Provide an ”traditional” filesystem for clients’ VM
  - API/consistency tuned for expected apps
  - E.g., pNFS, **Google file system**, HDFS
- Provide a containerized (traditional) union filesystem on client
  - Union FS does pathname lookup in one namespace, then a second, third, etc taking the first instance found; allows reuse of common (lower level) file systems
  - Populate private top layers of union (the container) from repositories of FS images (pull & untar)
  - E.g., Docker
- Provide just a block store (a virtual disk) to VM
  - Let clients build needed filesystem (is this a simplification?)
  - E.g., AWS EBS, Eucalyptus VBS
- Provide an archive store (an object store) separate from VM
  - Simple put/get semantics (like UNIX SCP)
    - CRUD: Create, Read, Update, Delete API
    - HDFS “Block”: single writer, sequential write, immutable on close
  - Explicitly external for thinking about failures
    - Archive, backup & disaster recovery, simple sharing
  - E.g. AWS S3, UNIX FTP/SCP, Box/Dropbox, iCloud

Containerized Union Filesystem

- Containers customize OS in "hypervisor"
  - Base OS is shared, so startup is faster, but customizations limited
- Container file system is union of multiple FS
  - Search in specific order, so lots of file be shared between containers
  - Can "white out" specific files; can copy-on-write into private layers
  - Use pull from image repository to build customized images
- Even with sharing, still lots of unnecessary data access during startup
  - Harter16 shows Docker "hello world" millions of pulls, little of it read

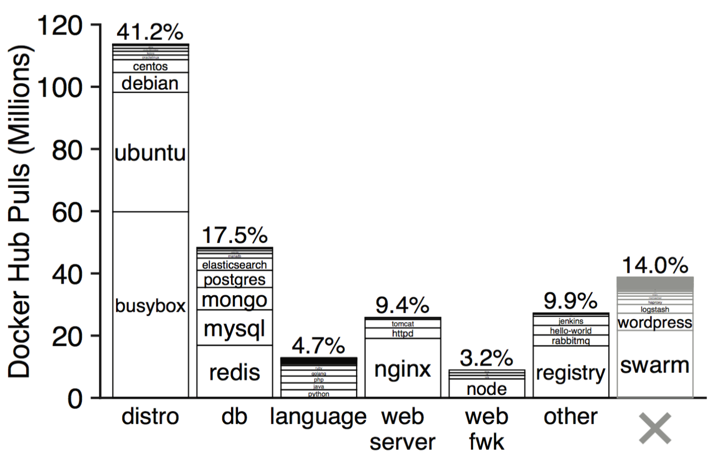


Figure 4: **Docker Hub Pulls.** *Each bar represents the number of pulls to the Docker Hub library, broken down by category and image. The far-right gray bar represents pulls to images in the library that are not run by HelloBench.*

[Harter, FAST16]

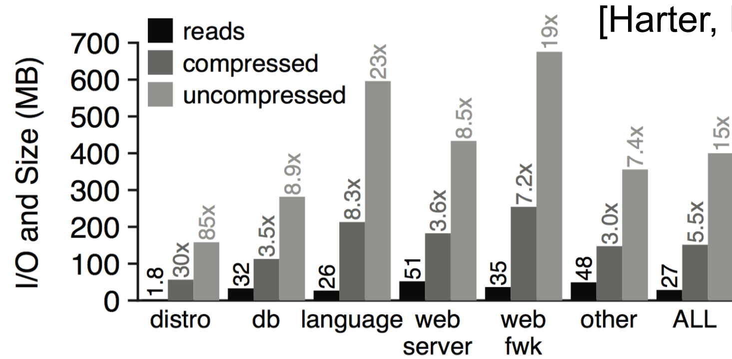


Figure 6: **Data Sizes (By Category).** *Averages are shown for each category. The size bars are labeled with amplification factors, indicating the amount of transferred data relative to the amount of useful data (i.e., the data read).*

AWS EBS or OpenStack Cinder

- Allows users to create virtual disks (VDs)
  - 1GB to 1TB in size
- VDs behave like block devices
  - Can be formatted; can be used like a HD; inherently not shared
- VDs are replicated for availability
  - Can also be snapshotted and stored in object store (S3, etc)

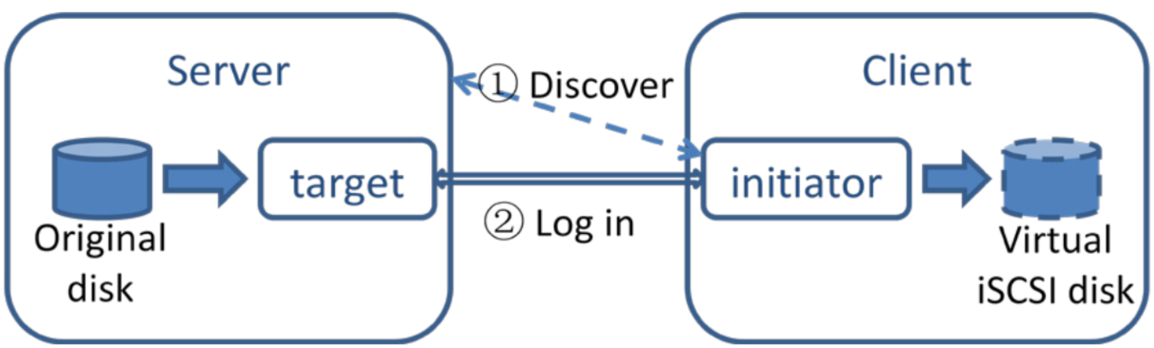


Figure 1. iSCSI target and initiator. Gao09

Details & implementation

- Clients access VDs with low-level protocol
  - E.g., iSCSI, not NFS
  - Reason may be non-technical
    - early VM providers were not distributed file system vendors
- VDs often implemented as files in disk
  - Allows for expansion/shrinking (“thin provisioning”)
  - But can result in performance interference

Software Defined Networking (SDN)

- Before SDN, lots of switches separately configured by admin
  - Many vendors, unique protocols and capabilities
  - System wide results hard to predict and control
- After SDN, all switches configured by SDN controller
  - Individual switches have little policy (just simple forwarding rules)
    - Individual switches cheaper (this alone facilitates deployment)
  - All exceptions to simple rules go to SDN controller
  - SDN controller sees all, so “maybe” plans global optimal forwarding rules
  - SDN controller not on data path most of time, so can be centralized
    - Maybe expensive but cost is amortized

Software Defined Storage (SDS)

- By analogy to SDN, separate execution path from planning path
- Homogenize all device & SW with capabilities abstractions & API
  - E.g. disk size, disk speed, RAID reliability, RAID speed, cache coherence
  - Can commoditize device costs (driving down prices)
- Homogenize all workloads with requirements abstraction & API
  - E.g. speed, reliability, access pattern, utility of old data
- Develop a placement scheduler to match requirement to capability
  - Maybe simple "good enough" compatibility plus cost of service
- Maybe apply continual evaluation and optimizing reconfiguration
  - An appealing idea that suffers from interference >> benefit too often
- Really a "Quality of Storage" abstraction

Background on Quality of Service (QoS)

- Quality of service (QoS) generalizes Quality of Storage (QoSt)
  - Service level objectives (SLO): goals, requirements, priorities
    - Eg. Cloud service available > 99.999% of year
  - Service level agreements (SLA): contract with failure penalties
    - E.g 25% refund if availability (99.99-99.999)%, 50% refund if (99.9%-99.99)%, money back otherwise
    - Usually restricts customer too
      - Eg. Reconfiguration downtime not counted, service load bounded to <= 6000 requests per minute
      - If bound is exceeded, a non-penalty outage of N minutes may occur in the next hour
- Monetary/contractual side makes continual optimization feasible
  - CFO funds initial design, adding/removing funds with frequency of penalties

Quality of Storage (QoSt)

- History of QoSt is series of attempts on SLO/SLA APIs
  - SNMP – peek into appliance internals to see what it is doing
  - SMIS – object model of system; sub classes specializing device function
  - TOSCA/OSLC – IBM's new JSON/XML specification language
- Problem with SLO/SLA is unpredictability of complex systems
  - Models of complex systems not accurate much of the time
  - Optimal matching only possible in simplified view of system
  - Leads to over-promising and under-delivering at technology level
  - CFO doesn't care that much because contract terms and alternative suppliers "optimization" works anyway (provided agility to change)

One aggressive demonstration of QoSt

- IOFlow: a Software-Defined Storage Architecture.

  Eno Thereska, Hitesh Ballani, Greg O'Shea, Thomas Karagiannis,

  Antony Rowstron, Tom Talpey, Richard Black, Timothy Zhu.

  SOSP 2013, Farmington PA, Nov 2013.

  - SDN "forwarding rules" replaced with "request queue ordering"
  - Flows are abstraction of SLO, service binding, data & requests
    - Used for bandwidth allocation & sharing, content checking, prioritization for latency

Next day plan

- Scheduling

Microsoft Research

Microsoft

IOFlow: a Software-Defined Storage Architecture

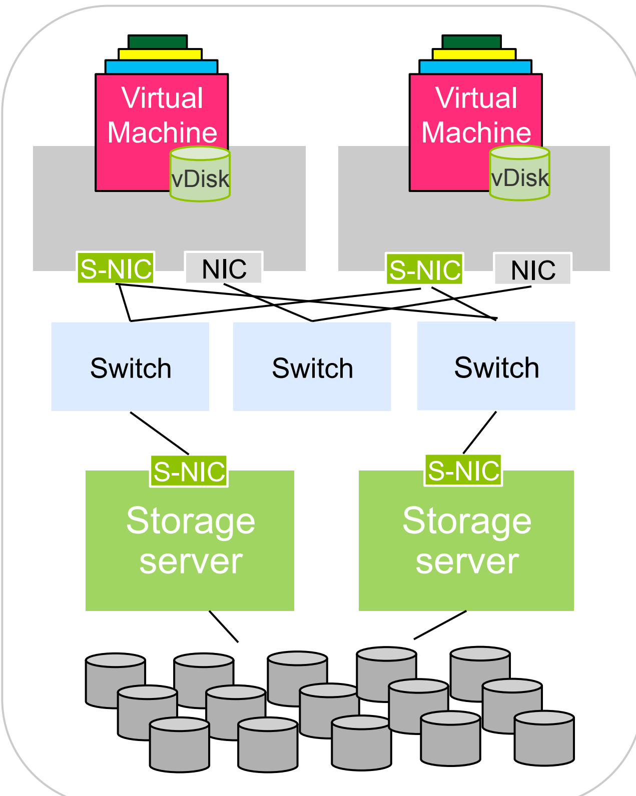
Eno Thereska, Hitesh Ballani, Greg O’Shea, Thomas Karagiannis,

Antony Rowstron, Tom Talpey, Richard Black, Timothy Zhu

Microsoft Research

“IOFlow: A Software-Defined Storage Architecture. Eno Thereska, Hitesh Ballani, Greg O'Shea, Thomas Karagiannis, Antony Rowstron, Tom Talpey, and Timothy Zhu. In SOSP'13, Farmington, PA, USA. November 3-6, 2013. “

Background: Enterprise data centers



- General purpose applications
- Application runs on several VMs
- Separate network for VM-to-VM traffic and VM-to-Storage traffic
- Storage is virtualized
- Resources are shared

Motivation

Want: predictable application behaviour and performance

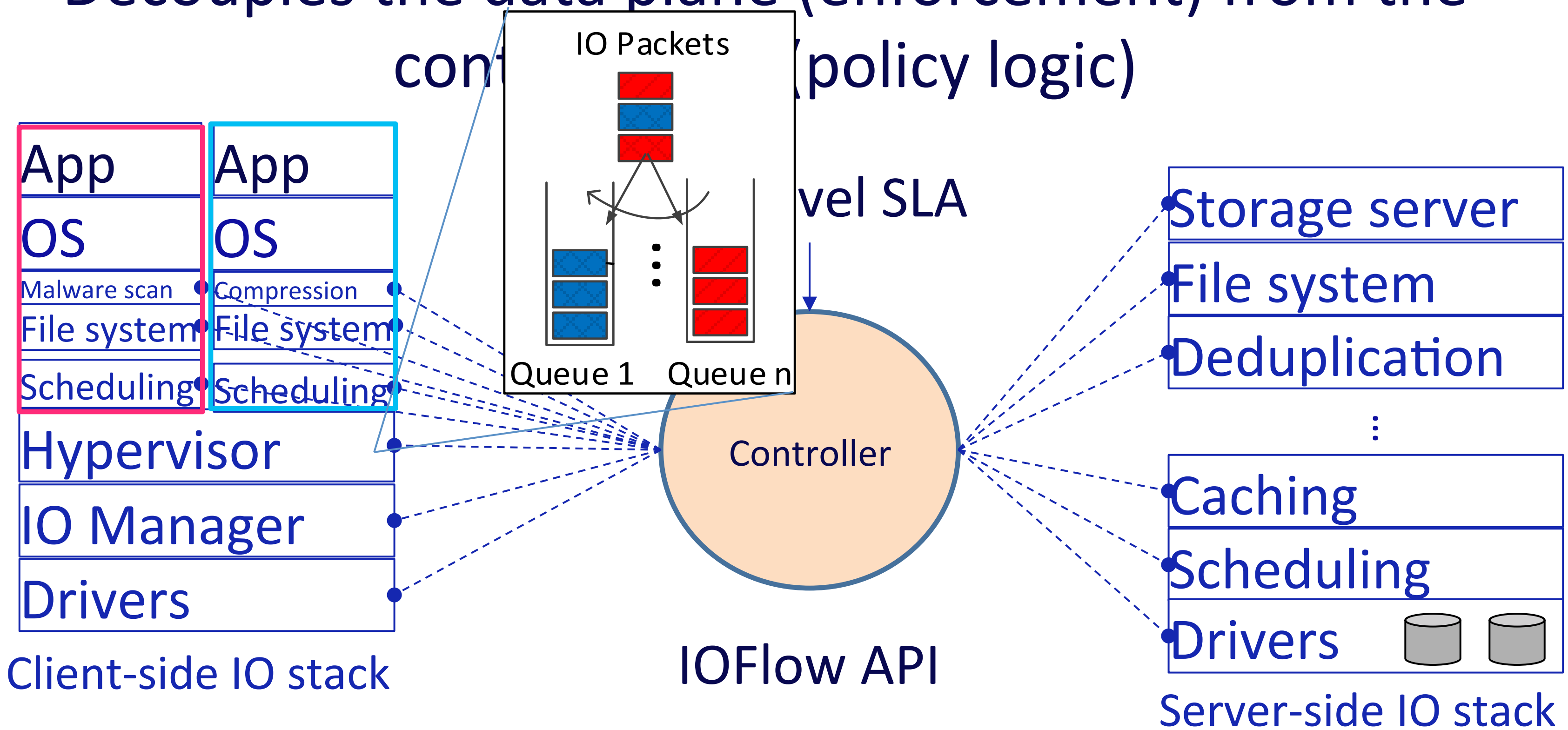
Need system to provide end-to-end SLAs, e.g.,

- Guaranteed storage bandwidth B
- Guaranteed high IOPS and priority
- Per-application control over decisions along IOs' path

It is hard to provide such SLAs today

IOFlow architecture

Decouples the data plane (enforcement) from the con (policy logic)



Storage flows

Storage “Flow” refers to all IO requests to which an SLA applies

<{VMs}, {File Operations}, {Files}, {Shares}> ---> SLA

source set — destination sets

- Aggregate, per-operation and per-file SLAs, e.g.,

  <{VM 1-100}, write, *, \\share\db-log}>---> high priority

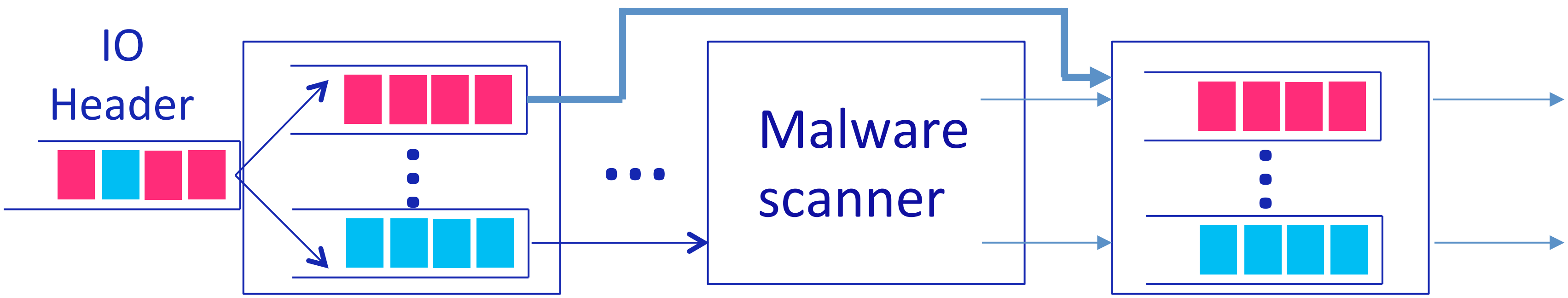
  <{VM 1-100}, *, *, \\share\db-data}> ---> min 100,000 IOPS

- Non-performance SLAs, e.g., path routing

  <VM 1, *, *, \\share\dataset>---> bypass malware scanner

IOFlow API: programming data plane queues

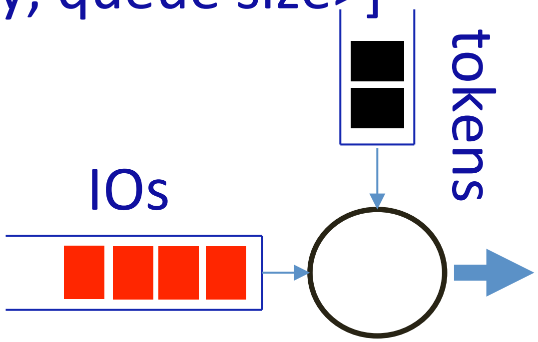
1. Classification [IO Header -> *Queue*]
2. Queue servicing [Queue -> *<token rate, priority, queue size>*]
3. Routing [Queue -> *Next-hop*]



Rate limiting for congestion control

Queue servicing [Queue -> <token rate, priority, queue size>]

- Important for performance SLAs
- Today: no storage congestion control



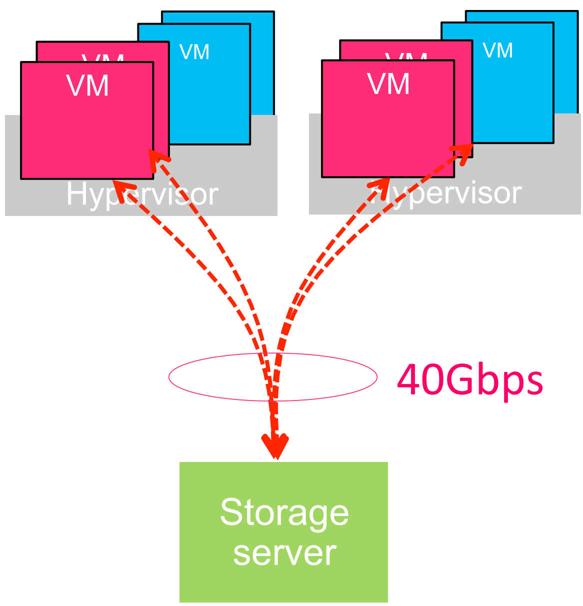
Challenging for storage: e.g., how to rate limit two VMs, one reading, one writing to get equal storage bandwidth?

Rate limiting based on cost

- Controller constructs empirical cost models based on device type and workload characteristics
  - RAM, SSDs, disks: read/write ratio, request size
- Cost models assigned to each queue
  - *ConfigureTokenBucket [Queue -> cost model]*
- Large request sizes split for pre-emption

Distributed, dynamic enforcement

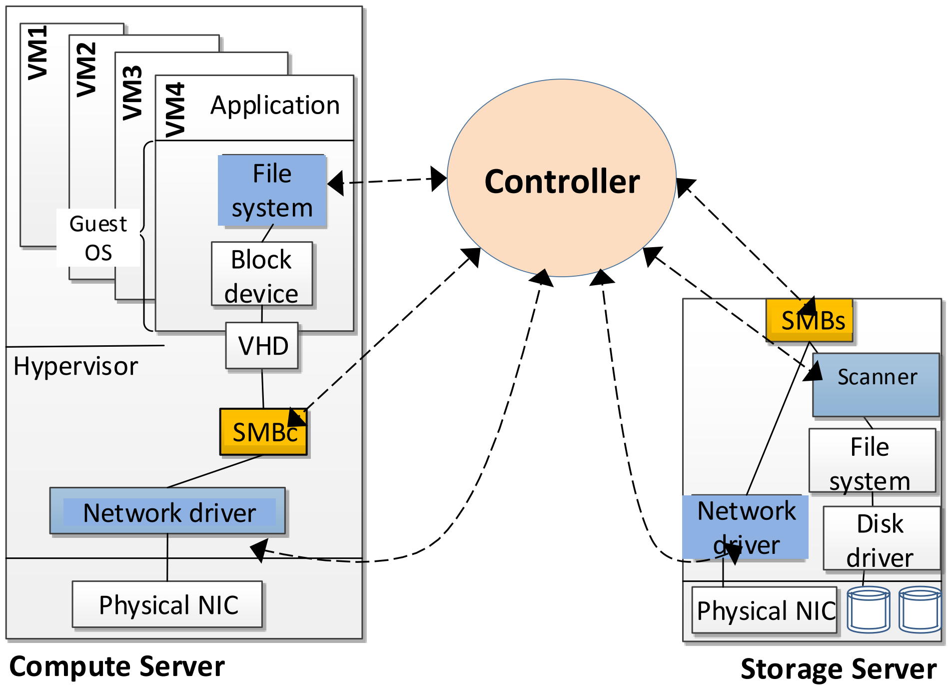
<{Red VMs 1-4}, *, * //share/dataset> --> Bandwidth 40 Gbps



- SLA needs per-VM enforcement
- Need to control the aggregate rate of VMs 1-4 that reside on different physical machines

- Static partitioning of bandwidth is sub-optimal

IOFlow implementation



2 key layers for VM-to-Storage performance SLAs

4 other layers
. Scanner driver (routing)
. User-level (routing)

. Network driver
. Guest OS file system

Implemented as filter drivers on top of layers

Summary of contributions

- Defined and built storage control plane
- Controllable queues in data plane
- Interface between control and data plane (IOFlow API)
- Built centralized control applications that demonstrate power of architecture
- *Ongoing work: applying to public cloud scenarios*

Related work (1)

- Software-defined Networking (SDN)
  - [Casado et al. SIGCOMM'07], [Yan et al. NSDI'07], [Koponen et al. OSDI'10], [Qazi et al. SIGCOMM'13], and more in associated workshops.
  - OpenFlow [McKeown et al. SIGCOMM Comp. Comm.Review'08]
  - Languages and compilers [Ferguson et al. SIGCOMM'13], [Monsanto et al. NSDI'13]
- SEDA [Welsh et al. SOSP'01] and Click [Kohler et al. ACM ToCS'00]

Related work (2)

- Flow name resolution
  - Label IOs [Sambasivan et al. NSDI'11], [Mesnier et al. SOSP'11], etc
- Tenant performance isolation
  - For storage [Wachs et al. FAST'07], [Gulati et al. OSDI'10], [Shue et al. OSDI'12], etc.
  - For networks [Ballani et al. SIGCOMM'11], [Popa et al. SIGCOMM'12]
  - Distributed rate limiting [Raghavan et al. SIGCOMM'07]