mClock: Handling Throughput Variability for Hypervisor IO Scheduling

Ajay Gulati, VMware Inc.

Arif Merchant, Google Inc. *(work done while at HP Labs)*

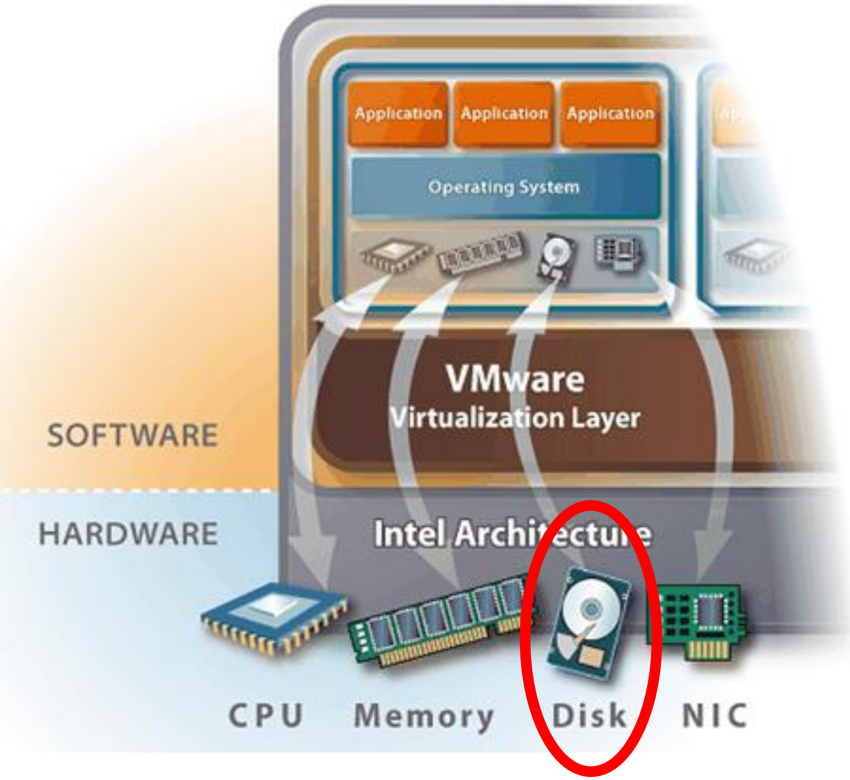
Peter J. Varman, Rice University

USENIX / ACM OSDI

October 6, 2010

Resource Management—State of the Art

- **Hypervisor multiplexes hardware resources between VMs**



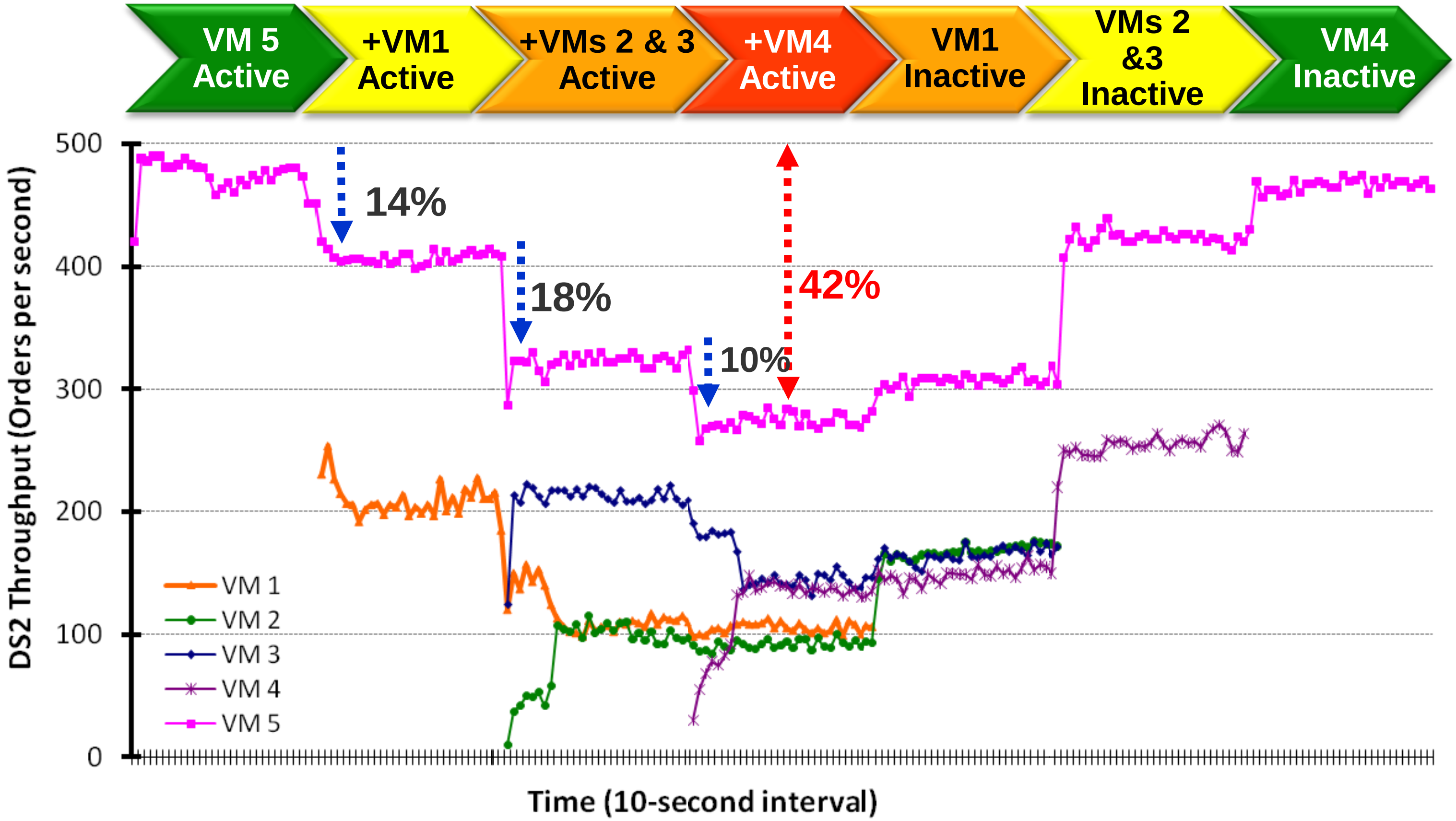
Three Controls

- **Reservation: minimum guarantee**
- **Limits: maximum allowed**
- **Shares: proportional allocation**
- ***Supported for CPU, Memory in ESX since 2003***

How about IO resource allocation?

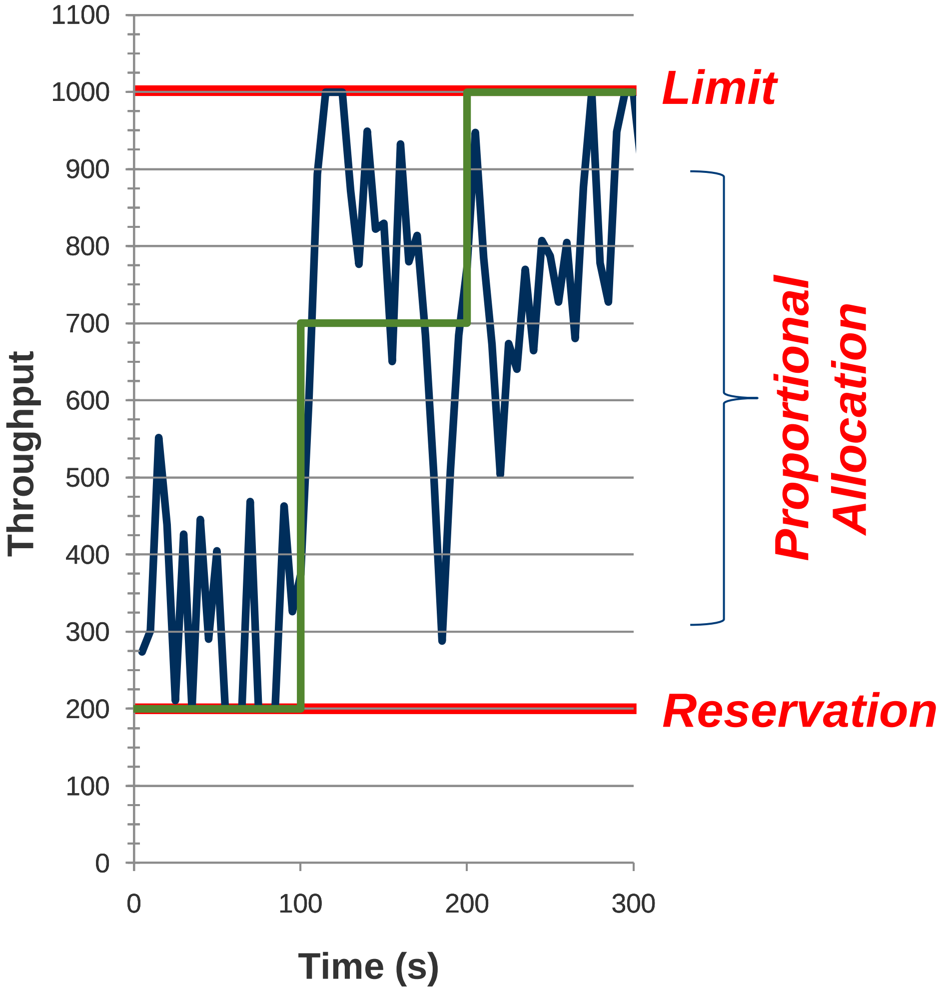
Variable IOPS Capacity Seen by VMs

- **Contention for I/O resources can arbitrarily lower a VM's allocation**



Each VM is running DVDStore on MS SQL Server

Why is Storage IO Allocation Hard?



- ***Storage workload characteristics are variable***
- **Available throughput changes with time**
- ***Must adjust allocation dynamically***
- **Distributed shared access**

Outline

- Problem Description & Motivation
- **Related Work**
- mClock Algorithm
- Experimental Results
- Conclusions & Future Work

Shoulders of Giants

A lot of fair-queuing, reservation control work precedes us

- **Proportional Share Algorithms**
  WFQ, virtual-clock, SFQ, Self-clocked, WF^{2}Q, SFQ(D), DRR, Argon, Aqua, Stonehenge
- **Algorithms with support for latency-sensitive applications**
  BVT, SMART, Lottery scheduling
- **Reservation-based Algorithms**
  Rialto, CPU & Memory management in ESX, Hierarchical CPU scheduling

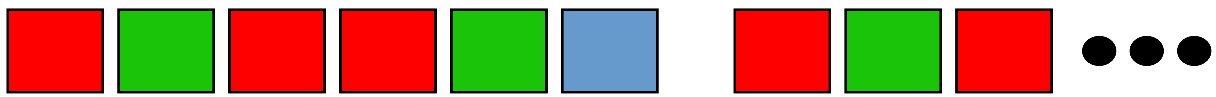
- **Novel features of mClock**
  - **Supports all controls in a single algorithm**
  - **Handles variable & unknown capacity**
  - **Easy to implement**

Outline

- Problem Description & Motivation
- Related Work
- **mClock Algorithm**
- Experimental Results
- Conclusions & Future Work

Typical Proportional-Share Scheduling

- **Each application has a weight w_i**
- **Each request is assigned a *tag***
- **Tags are spaced $1/w_i$ apart ➔ service allocated in proportion to w_i**
- **Example: 3 VMs A, B, C with weights 1/2, 1/3, 1/6**



A	**2**	4	**4**	**6**	8	8	**8**	10	**10**
B	3	**3**	6	6	**6**	9	9	**9**	12
C	6	6	6	6	6	**6**	12	12	12

time →

- **How to synchronize idle applications?**
- **Global virtual time (gvt) : gets updated on every request completion**

$$s^r = Max(s^{r-1} + 1/w_i, gvt)$$

gvt = minimum start tag in the system

mClock Algorithm

Three key ideas:

- **Real-time tags**
  - Needed for tracking reservations & limits
  - Virtual time loses track of actual allocation vs. time
- **Separate tags for reservation, shares & limit**
- **Dynamic tag selection and synchronization**
  - Need to decide which tag to use
  - Need to synchronize tags after idleness

mClock Algorithm: Multiple Tags

- **Three *real-time* tags**
  - Reservation tag : R — Reservation = r_i
  - Shares tag : P — Shares = w_i
  - Limit tag : L — Limit = l_i

$$R^r = Max(R^{r-1} + 1/r_i, currentTime)$$

$$P^r = Max(P^{r-1} + 1/w_i, currentTime)$$

$$L^r = Max(L^{r-1} + 1/l_i, currentTime)$$

mClock Algorithm: Tag selection

Two phases of Scheduling:

if (smallest reservation tag < current time) **// constraint-based**

Schedule smallest *eligible* reservation tag

else **// weight-based, reservations are met**

Schedule smallest *eligible* shares tag

Subtract $1/r_k$ from reservation tags of VM k.

A VM is eligible if (limit tag < current time)

Synchronization on request arrival from VM v_i:

if (v_i was idle)

Make minimum P tag = current time

Shift all P tags accordingly to maintain the relative ordering

mClock: Storage-specific Issues

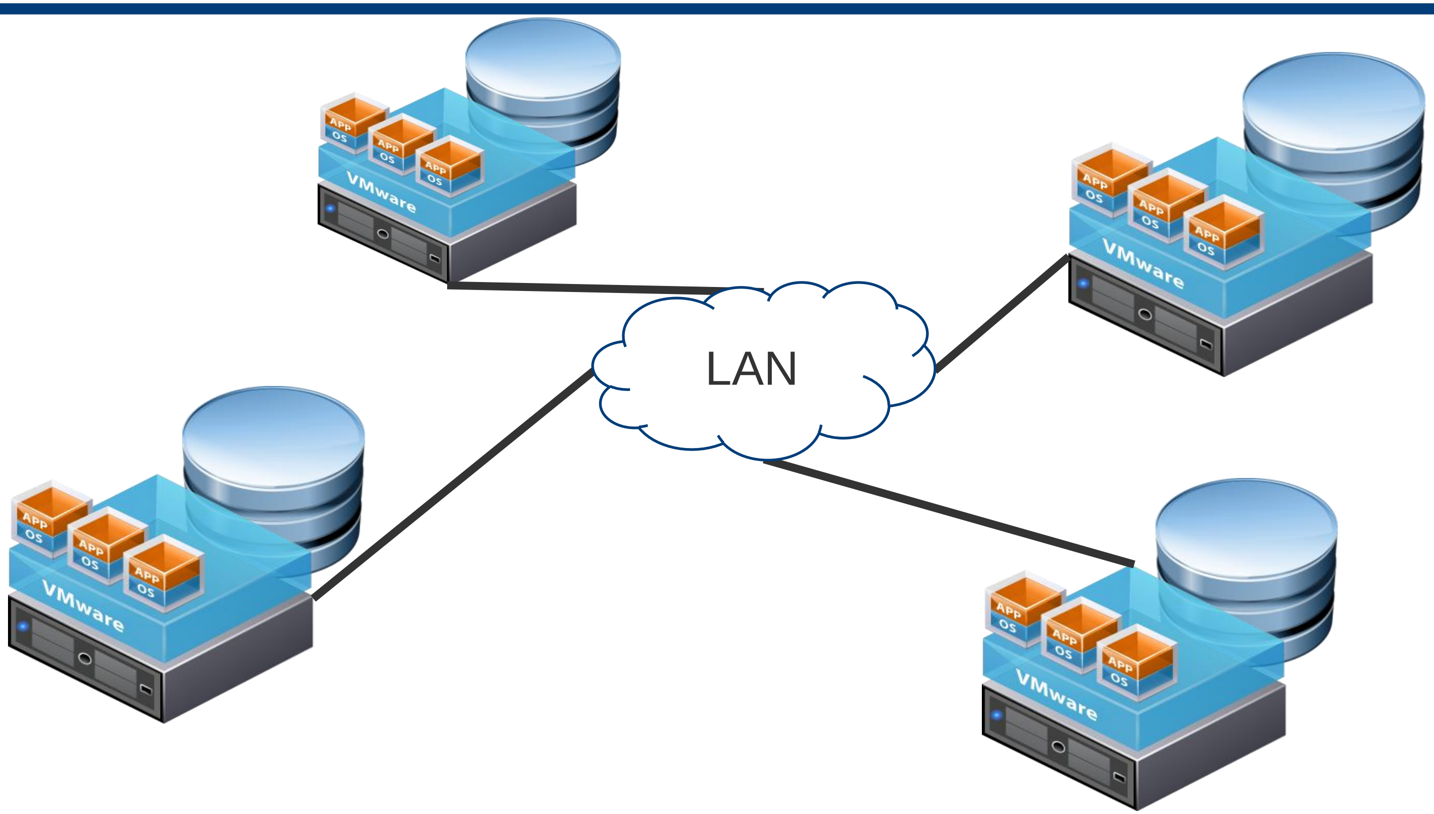
- **Burst Handling**
  - Allow VMs to gain idle-credit by pushing back *P* tags by σ
  - Key property: reservations are not impacted

$$P^r = Max(P^{r-1} + 1/w_i, t - \sigma / w_i)$$

- **IO size**
  - IO cost increases sub-linearly with request size
  - Scale the number of requests based on size

- **Request Location**
  - mClock schedules a bounded batch from a VM if addresses within 2 - 4 MB

dmClock: Clustered Storage Architectures



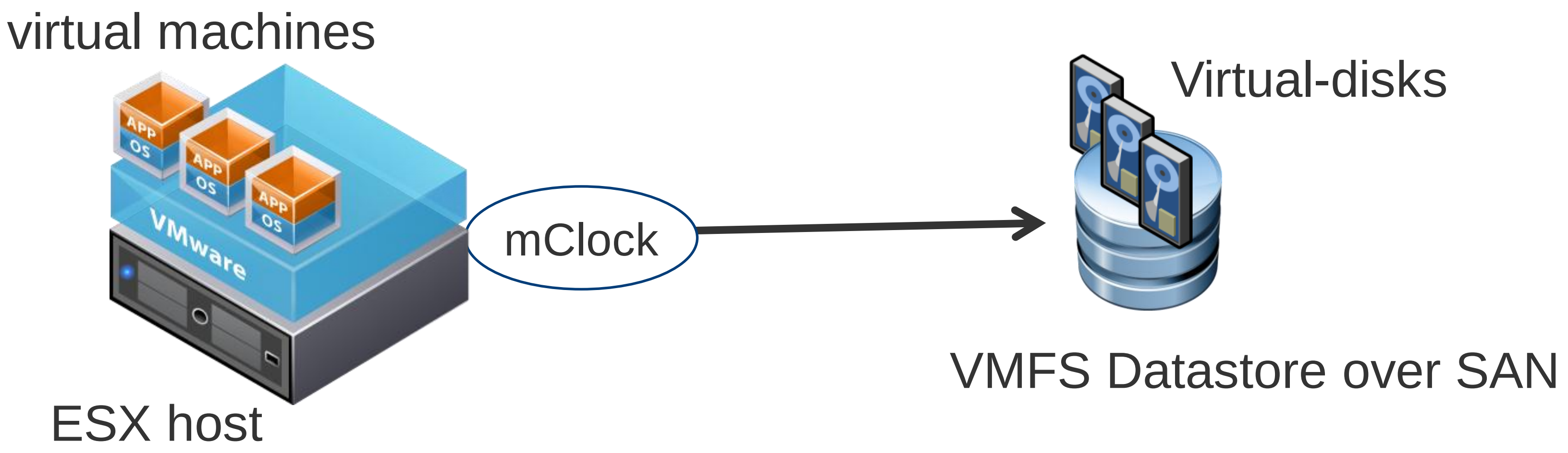
- A LUN is striped across local storage devices
- Host forwards VMs traffic, with certain tags
- dmClock enforces R, L, S controls (details in paper)

Outline

- Problem Description & Motivation
- Related Work
- mClock Algorithm
- **Experimental Results**
- Conclusions & Future Work

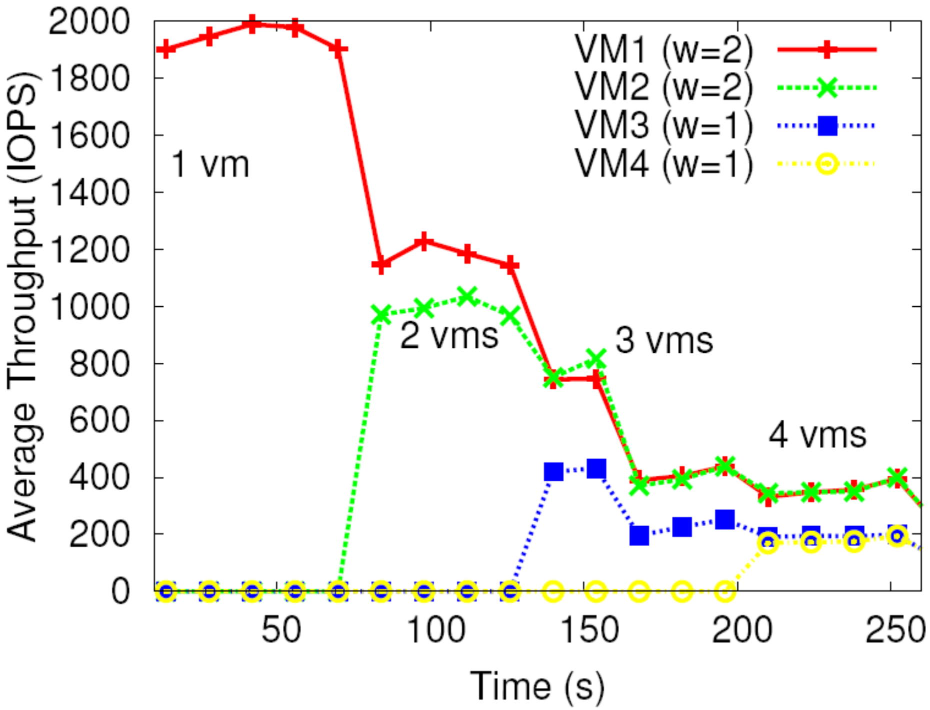
Experimental Setup

- **Dell PowerEdge 2950 server running VMware ESX hypervisor**
  - 3 to 6 virtual machines (VMs) – mix of Windows, Linux OSes
  - Data stores on EMC CLARiiON storage array – 10 disk Raid 0, Raid 5 groups
- **Workloads**
  - Iometer configurations and a Linux based micro-benchmark
  - Filebench: OLTP

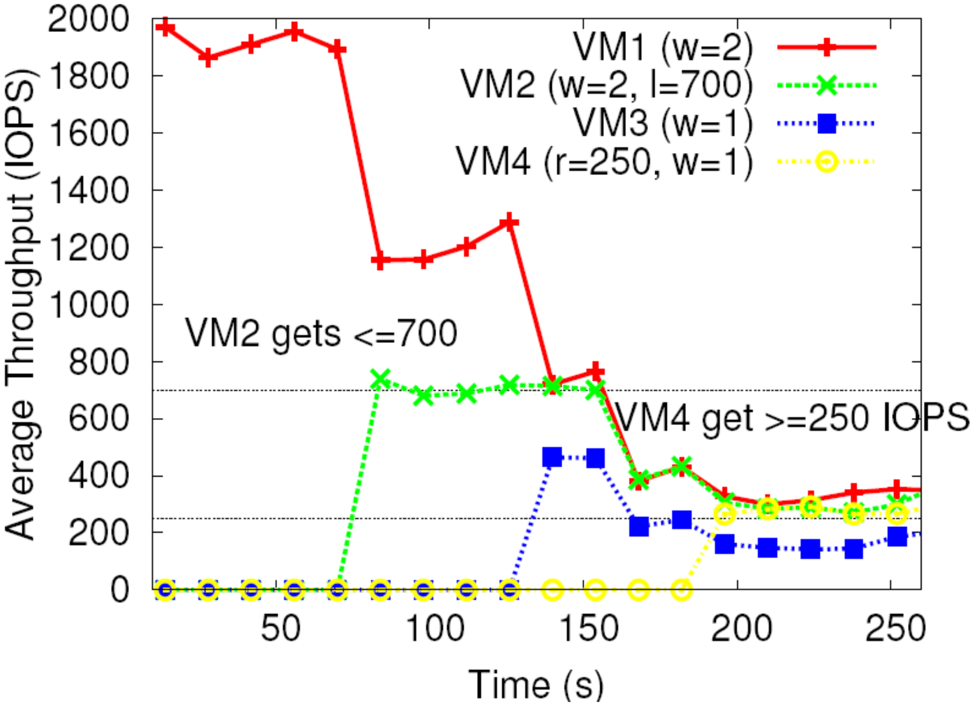


mClock: Reservation & Limits Enforcement

- 4 VMs, Shares in ratio 2:2:1:1
- VM2 has a limit of 700 IOPS, VM4 has reservation of 250 IOPS
- VMs are started every 60 sec



SFQ(D)



mClock

Enforces reservations, Limits

mClock: Burst Handling

- **Recall idle VM gets benefit when next there is spare capacity**
- **2 VMs**

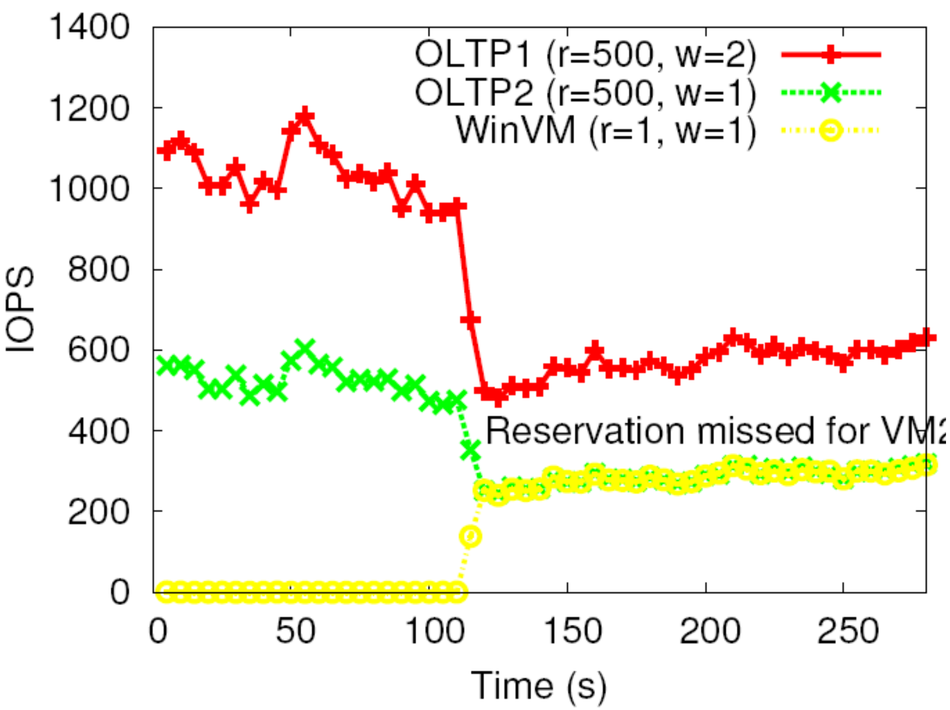
VM	R, L,S	Workload
VM1	0,Unlimited, 1	Bursty:128 IOs every 400ms, 80% random
VM2	0, Unlimited, 1	16 KB reads, 20% random,32 OIOs

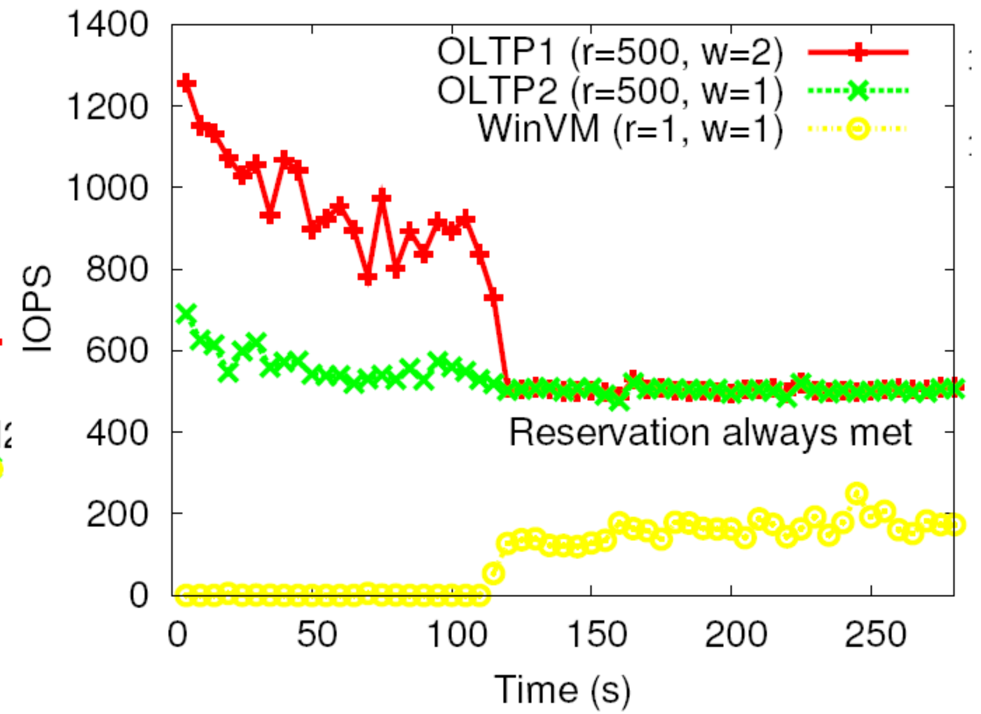
- **Results with idle credit of 1 and 64**

	σ = 1		σ = 64	
VM	IOPS	Latency	IOPS	Latency
VM1	312	**49 ms**	316	**30.8 ms**
VM2	2420	13.2 ms	2460	12.9 ms

mClock: Filebench workloads

- VM1, VM2 running Filebench OLTP workload
- Windows VM3 running Iometer started at *t* = 115 sec





SFQ(D),

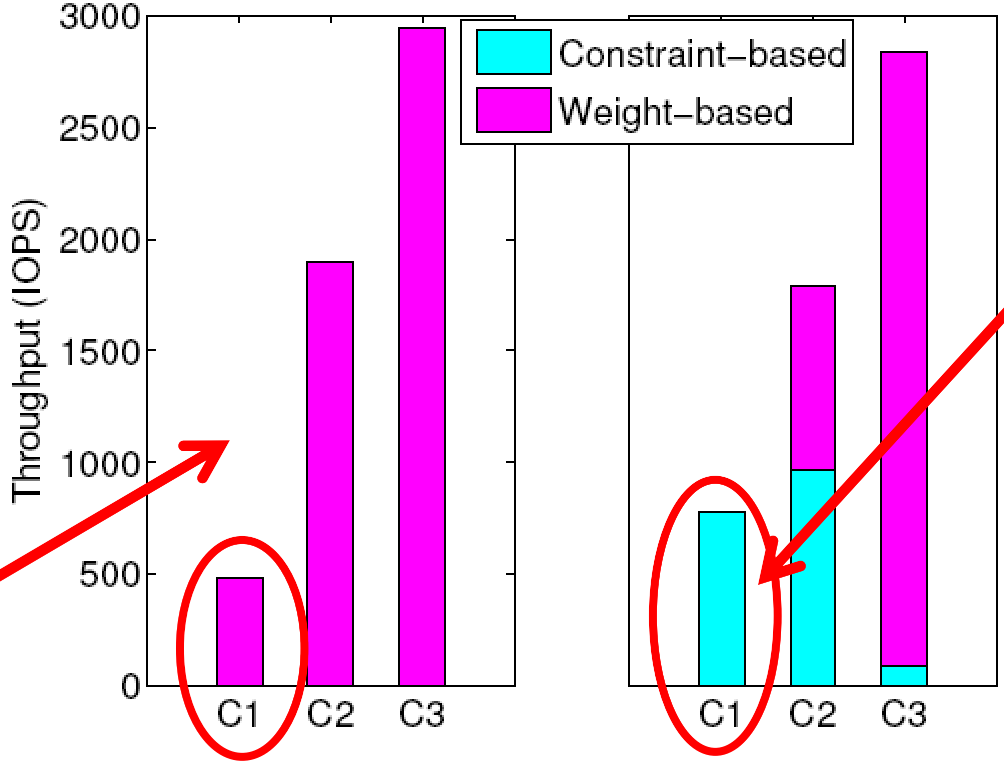
OLTP2 misses reservation

mClock,

Reservations are met

dmClock Result

- 3 Servers, 3 Clients (VMs) with shares in ratio 1:4:6
- Clients accessing servers in a uniform manner
- No Reservations to reservations of [800,1000,100]



Shares are respected

Reservations are enforced along with shares

vmware

Outline

- Problem Description & Motivation
- Related Work
- mClock Algorithm
- Experimental Results
- **Conclusions & Future Work**

Conclusions and Future Work

- **Storage IO allocation is hard**
- **mClock contributions**
  - Supports reservation, limit and shares in one place
  - Handles variable IO performance seen by hypervisor
  - Can be used for other resources such as CPU, memory & Network IO allocation as well

- **Future work**
  - Better estimation of reservation capacity in terms of IOPS
  - Add priority control along with RLS
  - *Mechanisms to set R, L,S and other controls to meet application-level goals*

Can we abstract out such controls into application's SLAs –

i.e. An upper bound on latency, lower bound on IOPS

Backup Slides

mClock: Reservation & Limits Enforcement

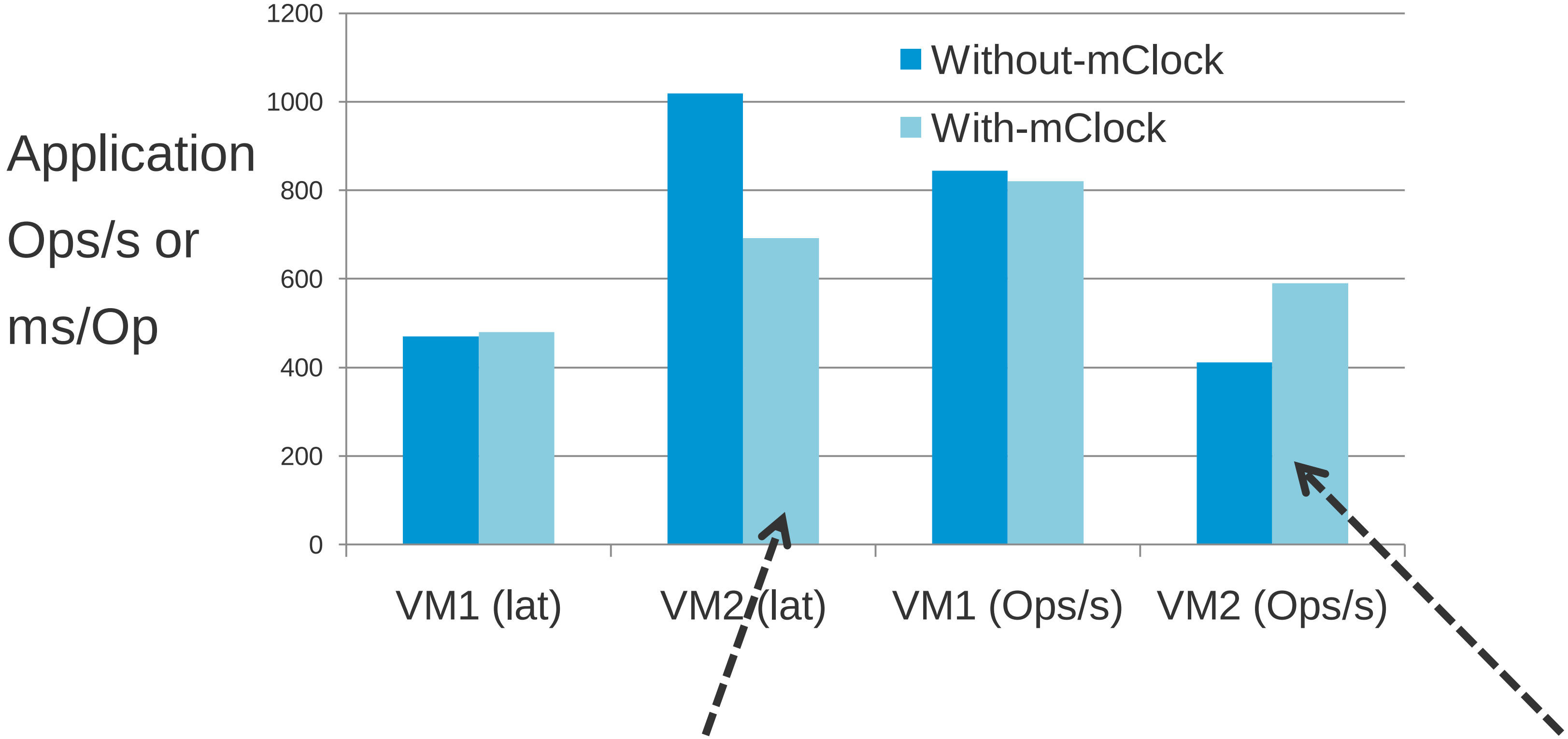
- **4 VMs, Shares in ratio 2:2:1:1**
- **VM2 has a limit of 700 IOPS, VM4 has reservation of 250 IOPS**
- **VMs are started every 60 sec**
- **VM workloads:**

VM	size, read%, random%	r_i	l_i	w_i
VM1	4K, 75%, 100%	0	MAX	2
VM2	8K, 90%, 80%	0	700	2
VM3	16K, 75%, 20%	0	MAX	1
VM4	8K, 50%,60%	250	MAX	1

Table 3: VM workloads characteristics and parameters

mClock: Filebench Application Performance

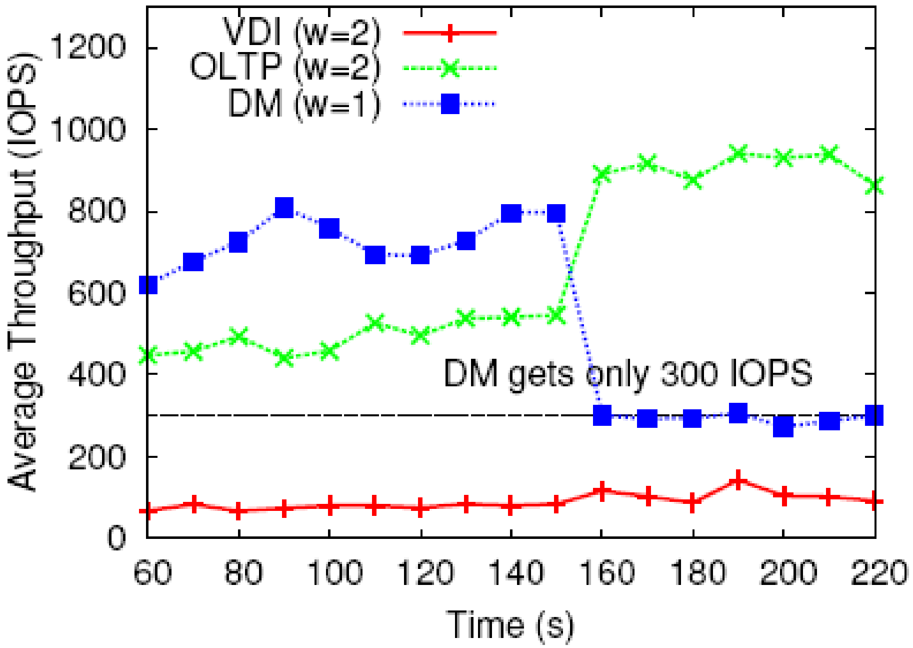
- **VM1, VM2 running Filebench OLTP workload**
- **Windows VM3 running Iometer started at t=115s**



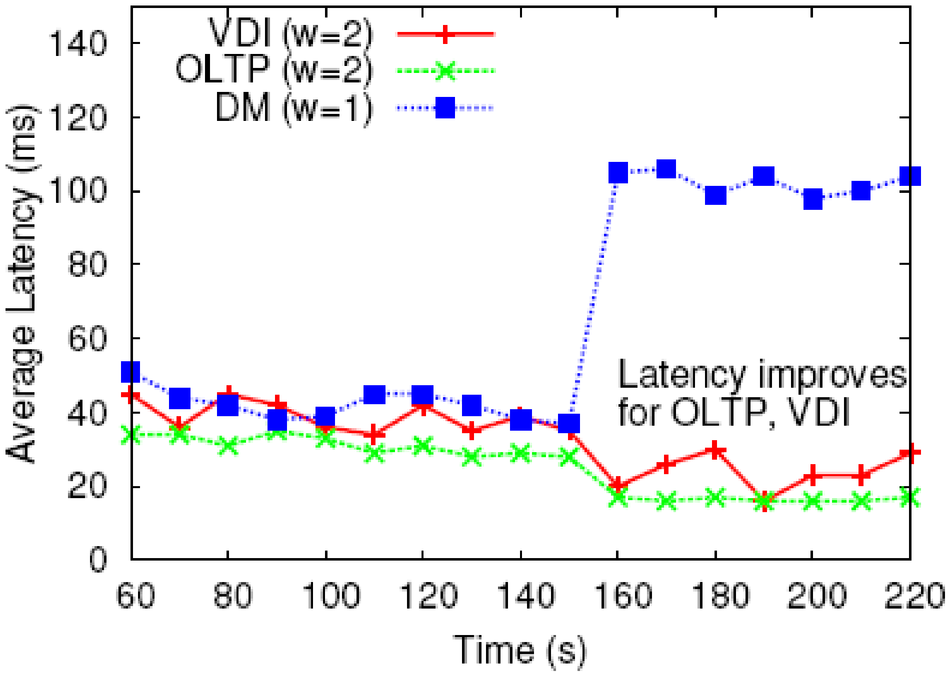
With mClock VM2's latency is lower; application Ops/s are higher

mClock: Limit Enforcement

VM	Shares	Limit	Workload
VDI	200	Unlimited	Bursty (128 IOs/s)
OLTP	200	Unlimited	8 KB, random, 75% reads, 16 OIOs
DM	100	300 (at t=140)	32 KB, seq reads, 32 OIOs



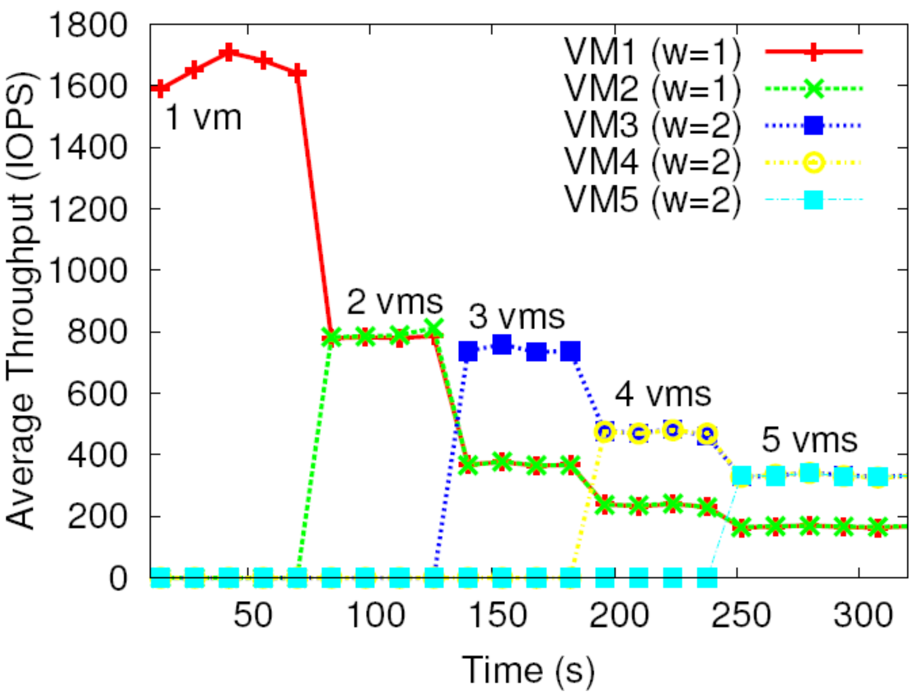
Throughput



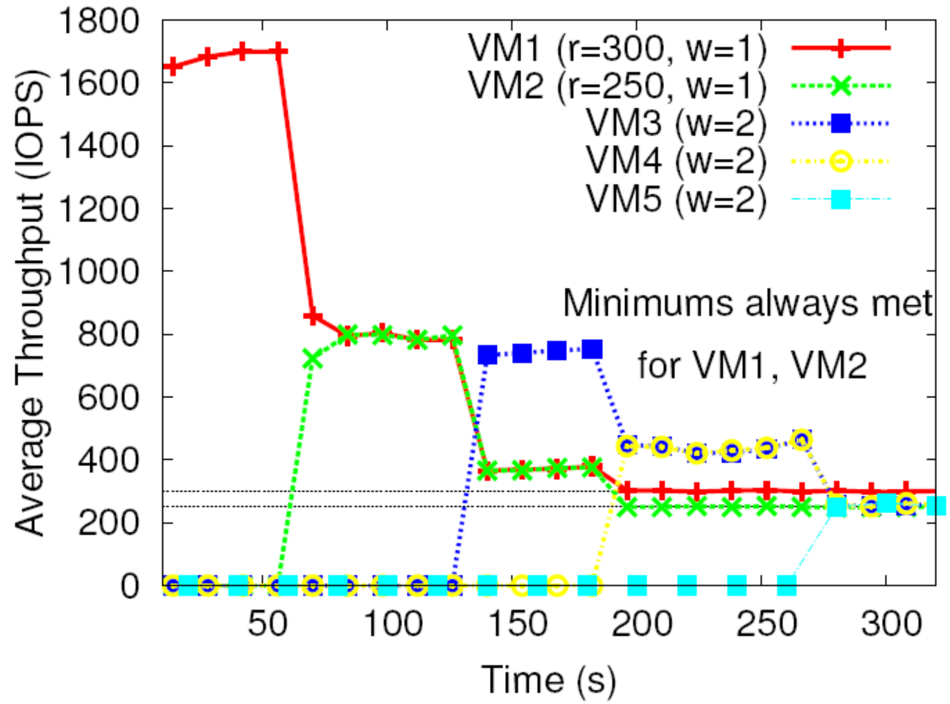
Latency

mClock: Reservation Enforcement

- **5 VMs, Shares in ratio 1:1:2:2:2**
- **VM1 and VM2 have reservation of 250 and 300 IOPS**
- **VMs are started every 60 sec**



SFQ(D)



mClock

Meets reservations

Scheduling Goals

- **Support - *Reservation, Limit (in IOPS), Shares (no units)***
- **An example:**

VM	Reservation	Shares	Limit
A	250	100	Unlimited
B	250	200	Unlimited
C	0	300	1000