Virtualizing Disk Performance

Presented by: Pinglei Guo

https://github.com/at15/

Virtualizing Disk Performance

Tim Kaldewey, Theodore M. Wong†, Richard Golding†, Anna Povzner, Scott Brandt, Carlos Maltzahn
Computer Science Department, University of California, Santa Cruz
†IBM Almaden Research Center
{kalt, tmwong, golding, apovzner, scott, carlosm}@cs.ucsc.edu

Abstract

Large- and small-scale storage systems frequently serve a mixture of workloads, an increasing number of which require some form of performance guarantee. Providing guaranteed disk performance—the equivalent of a "virtual disk"—is challenging because disk requests are non-preemptible and their execution times are stateful, partially non-deterministic, and can vary by orders of magnitude. Guaranteeing throughput, the standard measure of disk performance, requires worst-case I/O time assumptions orders of magnitude greater than average I/O times, with correspondingly low performance and poor control of the resource allocation. We show that disk time utilization—analogous to CPU utilization in CPU scheduling and the only fully provisionable aspect of disk performance—yields greater control, more efficient use of disk resources, and better isolation between request streams than bandwidth or I/O rate when used as the basis for disk reservation and scheduling.

1 Introduction

Significant research has been conducted on managing and guaranteeing CPU performance. Although many applications—e.g., multimedia, transaction processing, real-time data capture, and virtual machines—require I/O performance guarantees, relatively little successful research has been conducted on guaranteeing storage performance for mixed workloads.

As traditional real-time applications such as multimedia become ubiquitous, managing storage performance is critical for storage systems concurrently executing a mix of workloads. In small dedicated devices (e.g., an iPod), performance can be managed via offline or design-time resource provisioning. For other devices, including desktop machines and, especially, large shared storage servers, some other mechanism is needed.

Our goal is to provide a "virtual disk" in a shared storage system: applications should be able to reserve disk performance and the system should guarantee that performance, as shown in Figure 1. Doing so requires a uniform representation of disk performance that can be guaranteed regardless of what other applications are accessing the disk or how they behave.

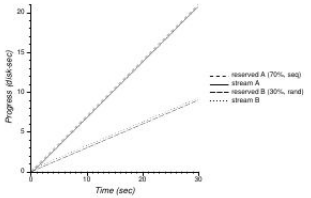


Figure 1: Progress of two I/O streams; one random and one sequential. Each stream uses its own virtual disk, both of which are hosted by a single physical disk. The virtual disk of the sequential stream has reserved 70% of the physical disk time and the virtual disk of the random one has reserved 30%. Other combinations (not shown) are similar.

Disk throughput has traditionally been used to describe storage system performance. Although throughput is a natural way to represent application requirements, guaranteeing throughput is difficult for four reasons. First, disk requests are stateful: the time required for a request depends heavily upon the location of the immediately preceding request. Second, disk request times are partially non-deterministic: disks are intelligent devices that perform bad block remapping, re-calibration, and other operations that can affect I/O times in ways that are impossible to determine *a priori*. Third, disk requests are non-preemptible: once issued to the disk, a request must be allowed to finish. Fourth, and perhaps most significant, the impact of the previous three is made critical by the huge difference between best- and worst-case request times: a best-case request can take a fraction of a millisecond, while a worst-case request may take tens of milliseconds. Thus throughput depends

Virtualizing Disk Performance

2008 UCSC & IBM

Real-Time and Embedded Technology and Application Symposium

TL;DR

- Shared storage is the trend
- Disk time utilization is workload independent
- Replace throughput based scheduler with utilization based in Zygaria
- AWS is still using throughput based for accounting, why?

Catlog

- Introduction
- Design and Implementation
- Evaluation
- Questions

Introduction - State of the art (around 2008)

- 2004 AWS launched
- 2007 AWS reach 18, 000 users
- 2009 Above the Clouds: A Berkeley View of Cloud Computing
- 2010 SOCC (Symposium on Cloud Computing)

Cloud（shared computing resource) is booming

Introduction - Problem of shared computing resource

- People just want to pay for what they used
  - How to measure the amount of usage, what’s the basis
- Mixed workload
  - Different requirement / usage of underlying hardware
  - Isolation from one
- Users are picky
  - Your users also have their users, they need guarantee
  - You want to make every user happy (nice day dream)

Introduction - Virtualizing Disk (Previously)

- Share ?
  - Capacity
  - Performance
- Basis? Throughput based (is not a good measurement)
  - Depend on Workload heavily
  - Sequential requests: 10 microsecond
  - Random requests : 15 millisecond = 15, 000 microsecond **1,500** times
- Guarantee?
  - Based on worse case -> low efficiency 1% of max performance

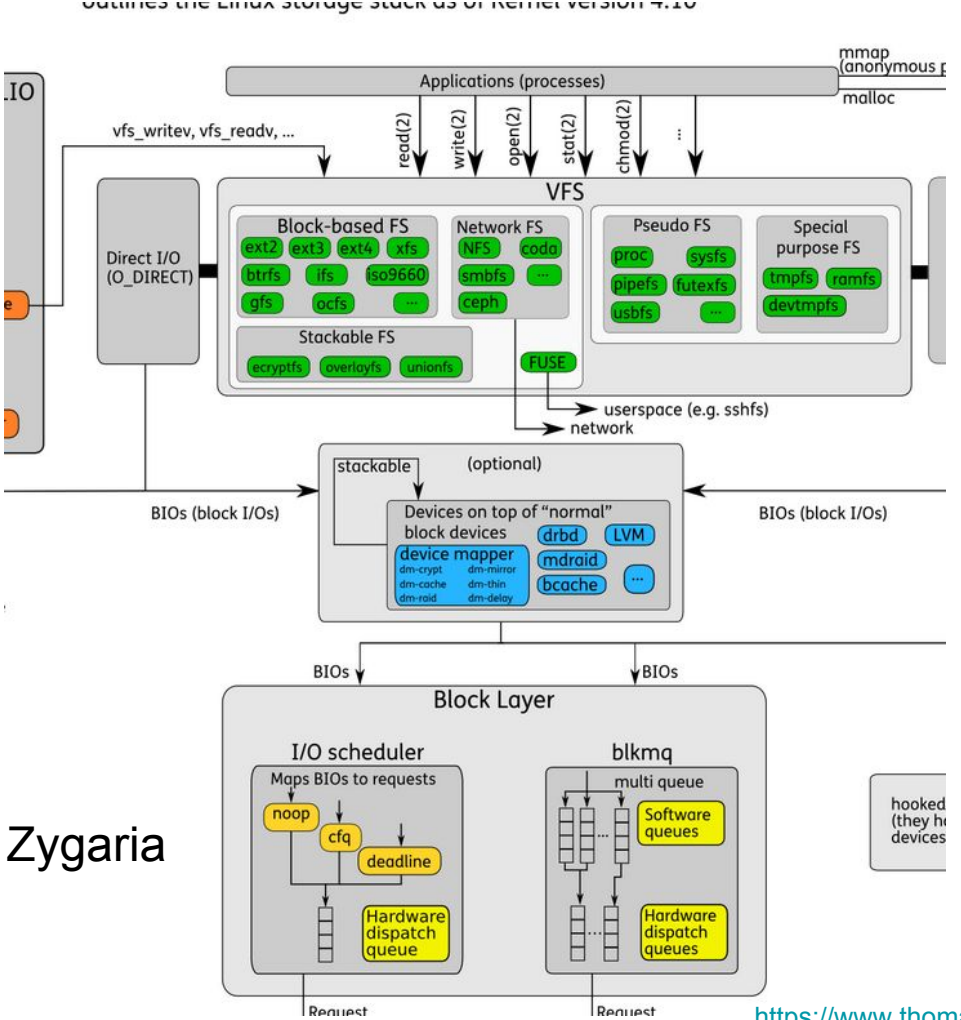
Introduction - Visualizing Disk (in this paper)

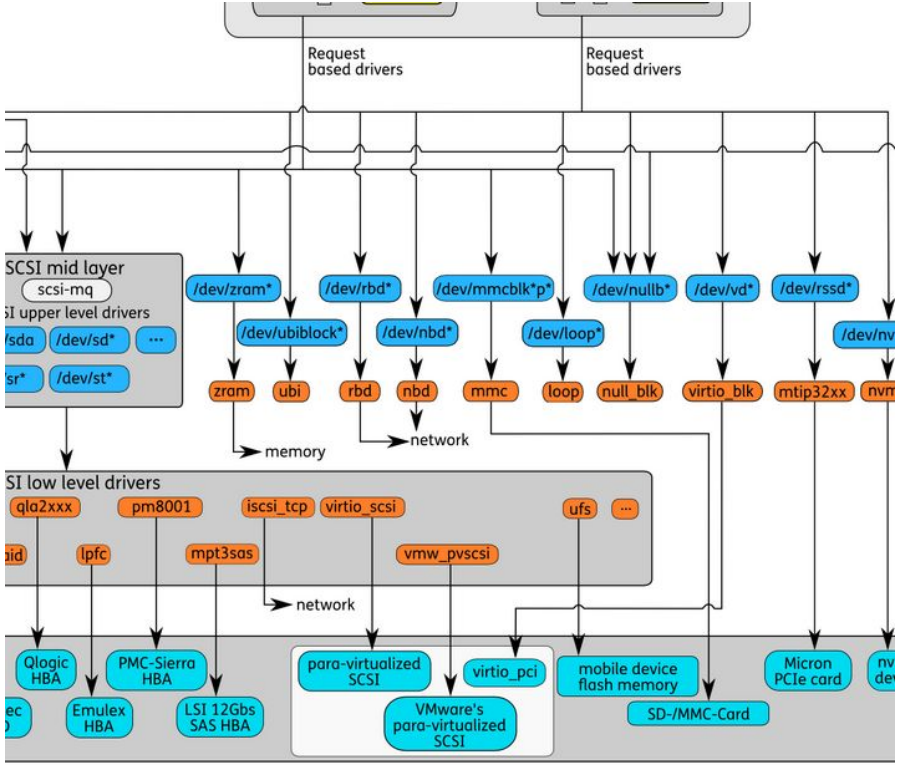
Disk Time Utilization

‘The only workload independent way of expressing and managing disk performance’

Design and Implementation

- Based on *Zygaria: Storage performance as a managed resource*
- Token Bucket
  - Reserve: guaranteed share of disk time
  - Limit: caps the use of unclaimed resources
- EDF (Earliest deadline first)
- Estimation using fixed value and categorize
  - Sequential: < 100 sectors from previous request, 300 microseconds
  - Random: 20 milliseconds





Zygaria

https://www.thomas-krenn.com/de/wikiDE/images/e/e0/Linux-storage-stack-diagram_v4.10.png

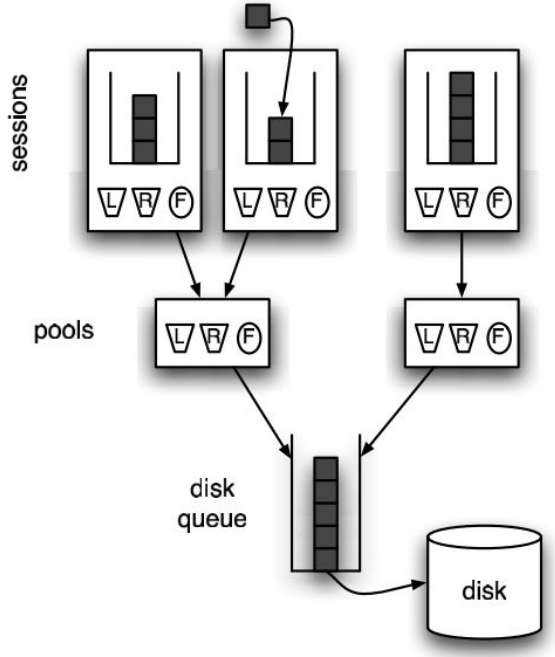


Figure 2. Token bucket hierarchy and I/O queue structure for sessions, pools, and the underlying device. (R) and (L) are reserve and limit token buckets; (F) is a moving average estimator used for fair sharing.

Zygaria (throughput-based)

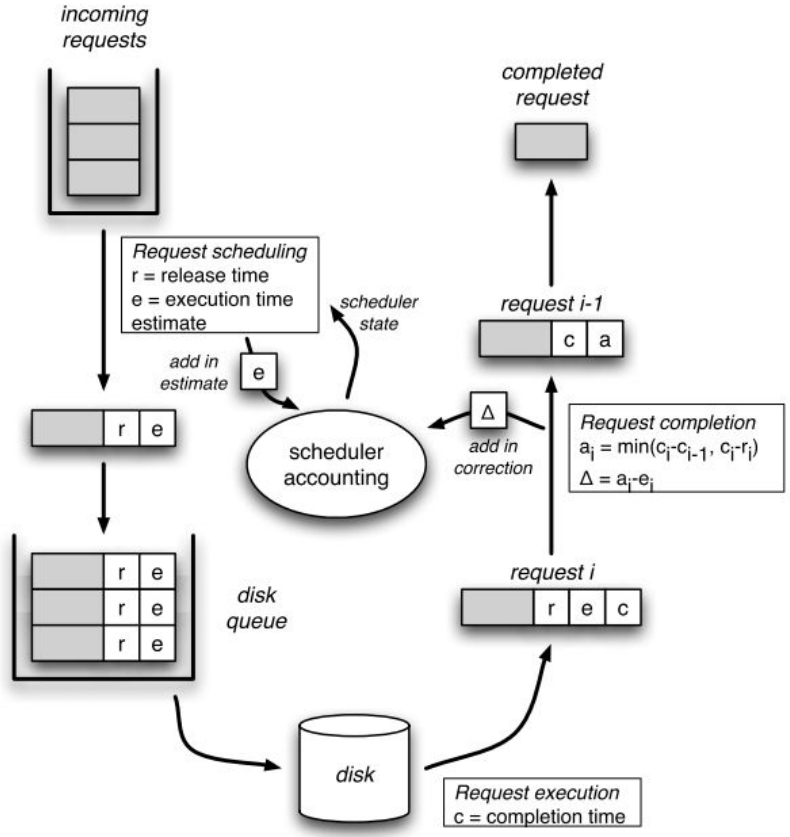
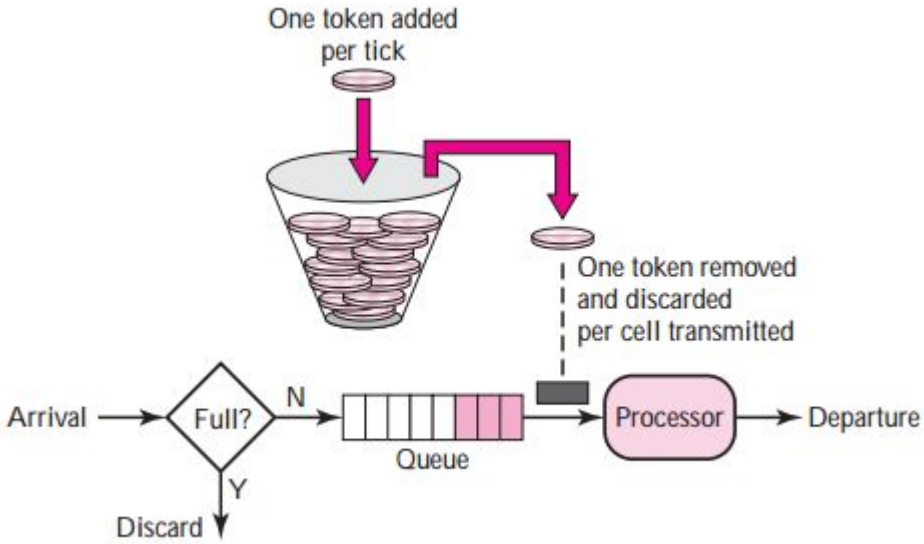


Figure 3: Flow of I/O requests through an I/O scheduler.

Zygaria++? (utilization- based)

Implementation - Token bucket



- Commonly used for rate limit in network and API
- Can be implemented as a counter
- When stable, speed is same as the ticker
- Burst when bucket is full (P9 L7）

You can’t spend more money than your parents give you

http://gateoverflow.in/100232/token-bucket-gate-2016-question

Implementation - Scheduling : EDF

Earliest deadline first

- Widely used by students
- Read the paper that is due tomorrow first
- Performance boost when it comes to deadline
- See the Horizon paper for more info

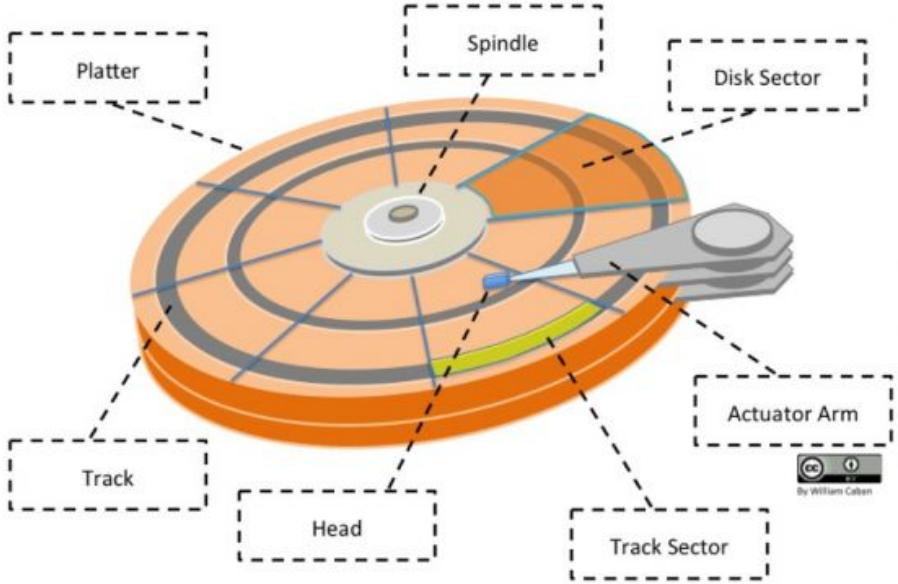
Implementation - Estimate time

Previously

- Special device driver
- Machine learning
- Like what people did for branch prediction and data prefetching

Zygaria++

- Categorize based on sectors distance (100 sectors)
- 300 microseconds for sequential request
- 20 milliseconds for random request
- Correction based on feedback



PS: Sector size 512 -> 4096 since 2010

https://en.wikipedia.org/wiki/Disk_sector

https://blogs.cisco.com/datacenter/decoding-ucs-invicta-part-1

Evaluation - Setup

Two virtual disk, one random, one sequential

Pharos, random / sequential I/O streams with two arrival pattern

- Open-loop: Go, Go, Go
- Closed loop: Go, check, Go, check, Go, Go, Go, fire in the hole …..

Extension (not directly related to paper)

- The workload generator is also a feedback control system like the scheduler
- Cascading failure in distributed systems
  - https://github.com/Netflix/Hystrix

Evaluation - Three parts

- Compare with throughput based Zygaria
  - Zygaria++ is better
- Effect of different estimate time (tolerance of model error)
  - *As long as there is significant difference*
- Dynamic workload
  - Lagging like most feed-back control system

Evaluation - Time utilization v.s. Throughput

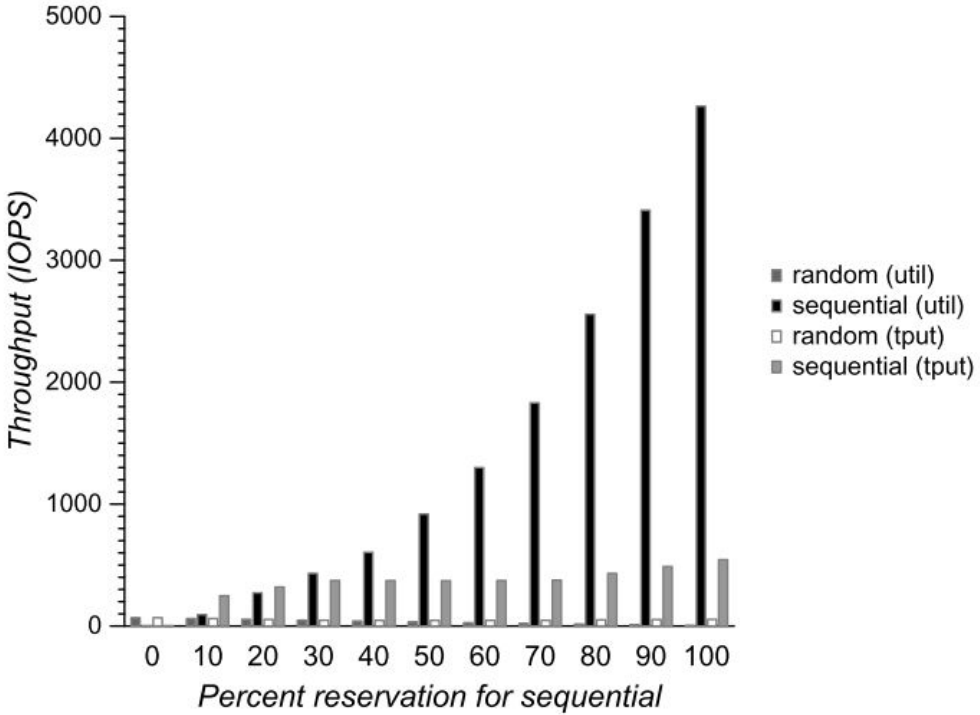


Figure 6: Efficiency comparison

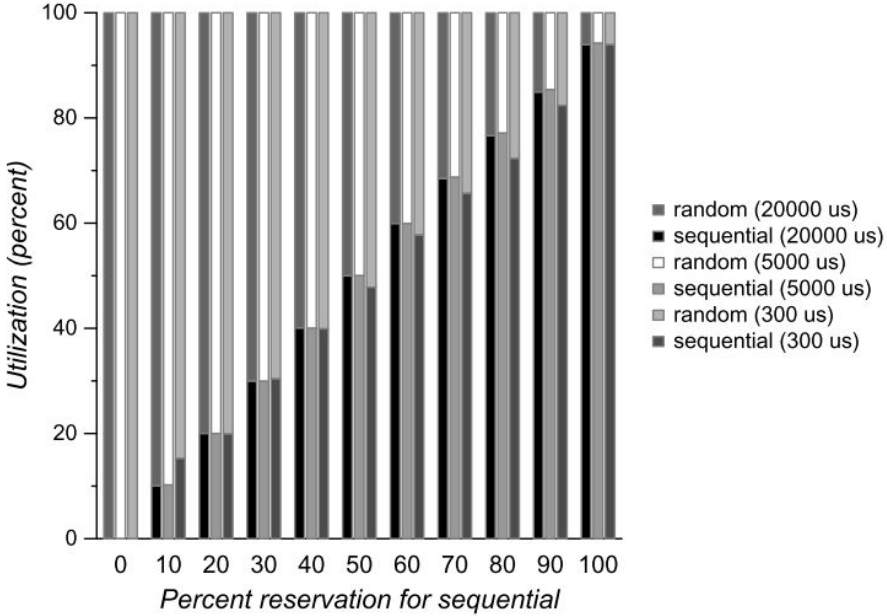
Throughput based

Sequential	<=	Reservation
Reservation c	<=	Maximum guaranteeable
Max guarantee	:=	Worst case

Evaluation - Tolerance model error

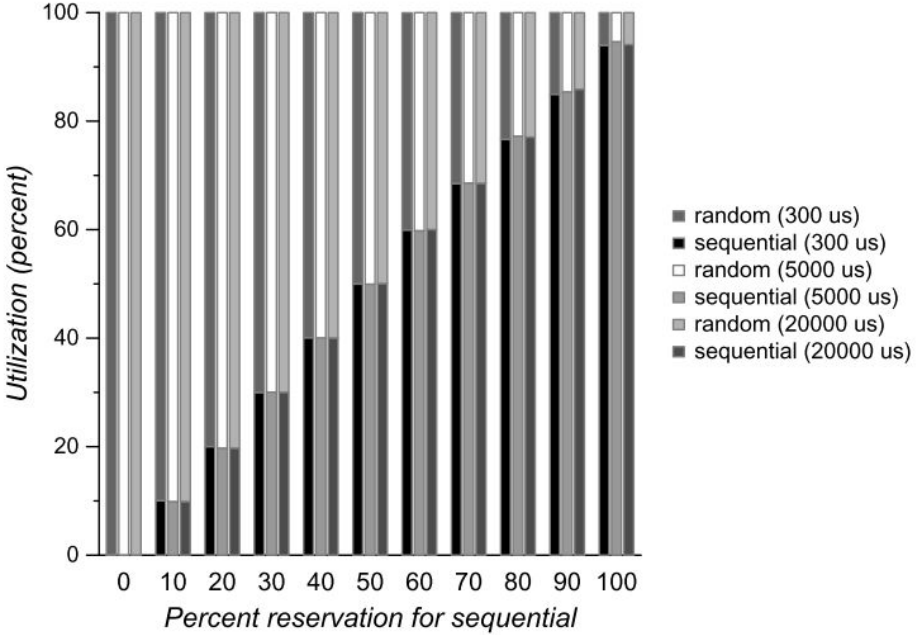
Origianl
Random: 20,000 us
Sequential: 300 us

Figure 7(a) Change Random estimation



Random: 300 us -> 5,000 us -> 20,000 us

Figure 8(a) Change Sequential estimation



Sequential: 20, 000 us -> 5,000 us -> 300 us

As long as there is a significant difference between the sequential and random estimates, the scheduler will behave correctly

Evaluation - Dynamic workload

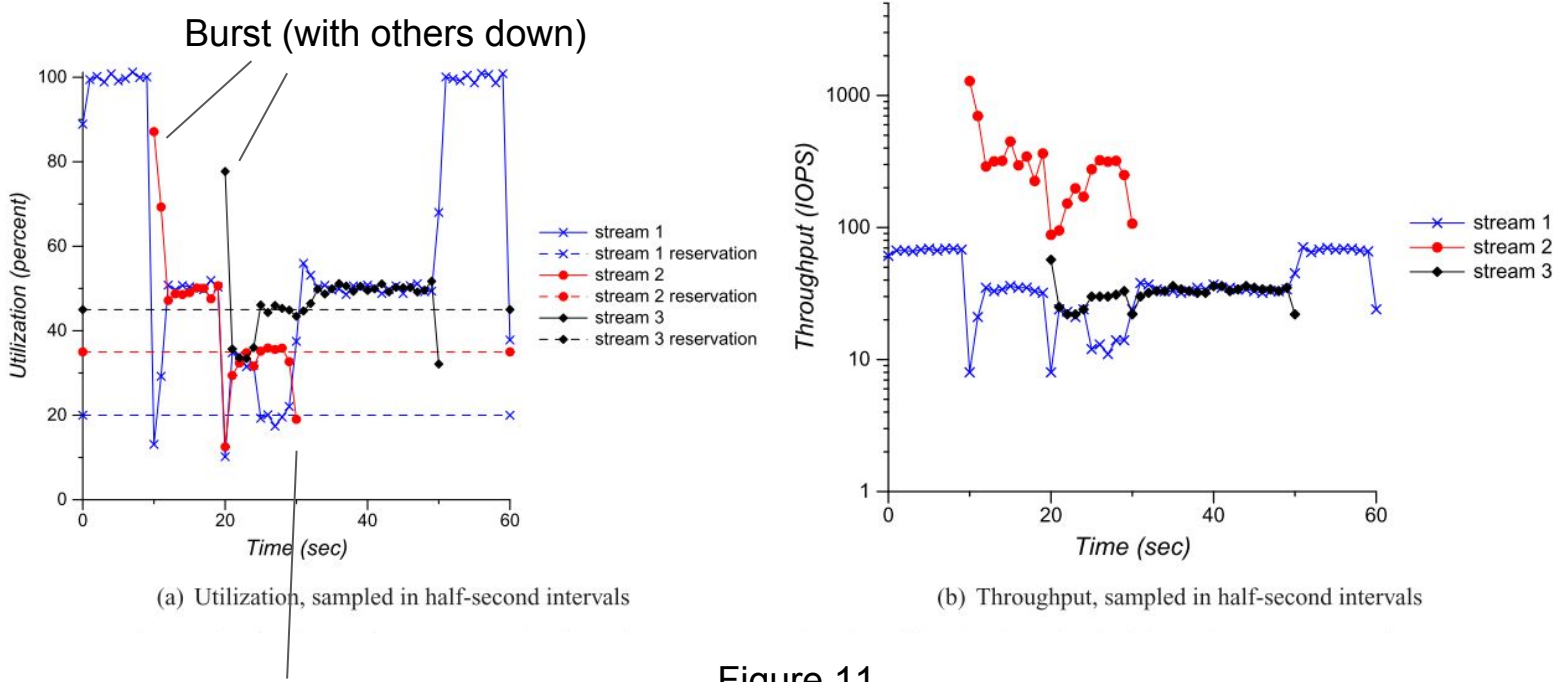


Figure 11

Questions

- Reservation vs. Priority
  - *People need priority because the have poor scheduler*
- Only compared to Zygaria, what about other systems, old Zygaria already beat all the peers?
- Mixed workload on a single virtual disk instead of one workload for one virtual disk
- The simple estimation may not work for new HDD, not to mention SSD, NVM
- Why AWS EBS (Elastic Block Store) is still charging by number of requests, and it only shows you MAX IOPS
  - EBS is not on same server with EC2, will network ever become the bottleneck?
  - Only suggest RAID0, don’t suggest use RAID for redundancy

Volume Type	EBS Provisioned IOPS SSD (io1)	EBS General Purpose SSD (gp2)*	Throughput Optimized HDD (st1)	Cold HDD (sc1)
Short Description	Highest performance SSD volume designed for latency-sensitive transactional workloads	General Purpose SSD volume that balances price performance for a wide variety of transactional workloads	Low cost HDD volume designed for frequently accessed, throughput intensive workloads	Lowest cost HDD volume designed for less frequently accessed workloads
Use Cases	I/O-intensive NoSQL and relational databases	Boot volumes, low-latency interactive apps, dev & test	Big data, data warehouses, log processing	Colder data requiring fewer scans per day
API Name	io1	gp2	st1	sc1
Volume Size	4 GB - 16 TB	1 GB - 16 TB	500 GB - 16 TB	500 GB - 16 TB
Max IOPS**/Volume	20,000	10,000	500	250
Max Throughput/Volume	320 MB/s	160 MB/s	500 MB/s	250 MB/s
Max IOPS/Instance	75,000	75,000	75,000	75,000
Max Throughput/Instance	1,750 MB/s	1,750 MB/s	1,750 MB/s	1,750 MB/s
Price	$0.125/GB-month $0.065/provisioned IOPS	$0.10/GB-month	$0.045/GB-month	$0.025/GB-month

https://aws.amazon.com/ebs/details/

Take Away

- Disk time utilization might be a more effective basis than throughput
- Provide guarantee when you know how to measure (and charge your users)
- Simple solution works (don’t blindly apply machine learning to everything, machine needs a break)
- Scheduler is everywhere
- Earliest deadline first (research plan due today)

Thank You

https://github.com/at15/