Microsoft Research

Microsoft

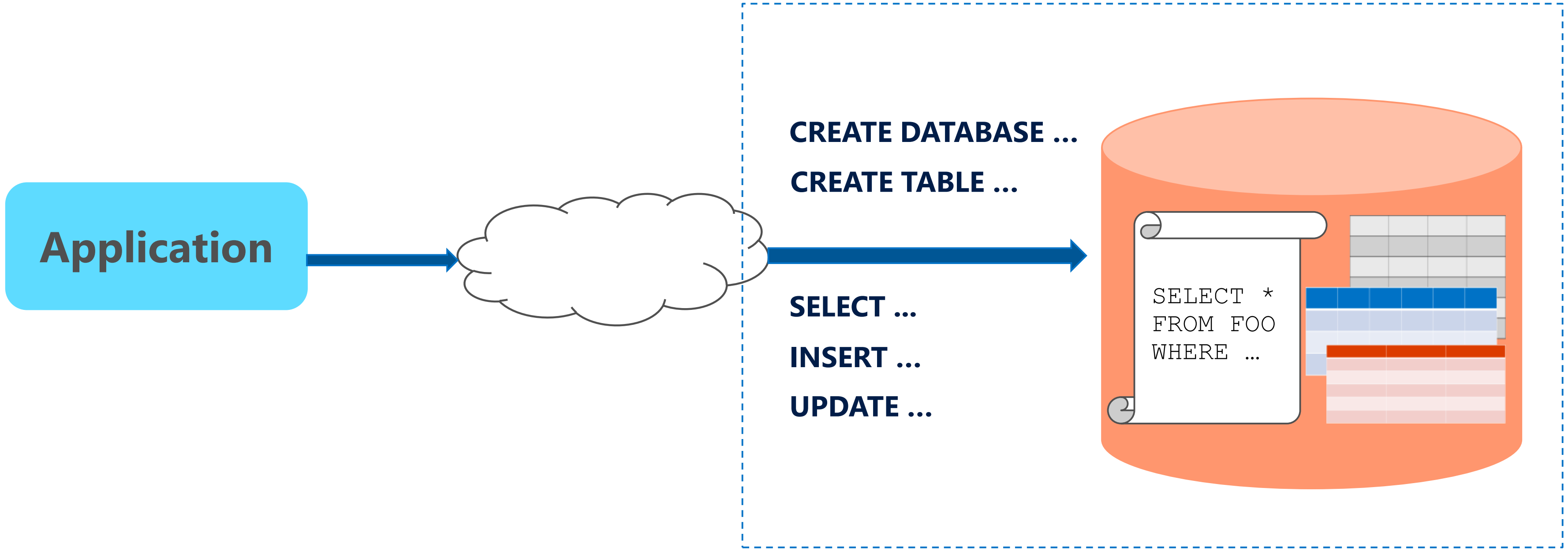
Performance Isolation in Multi-Tenant Relational Database-as-a-Service

Sudipto Das (Microsoft Research)

Joint work with
Vivek Narasayya, Manoj Syamala, Surajit Chaudhuri, Feng Li, Hyunjung Park, Ishai Menache
Microsoft Research

Peter Carlin, Mike Habben, George Reynya
Microsoft Azure SQL DB

Relational Database-as-a-Service (DaaS)



Tenants provision a logical database
Familiar relational data model, SQL API
Easy to provision, pay-as-you-go
High availability, managed backups, geo-distribution, disaster recovery

Microsoft Azure SQL Database

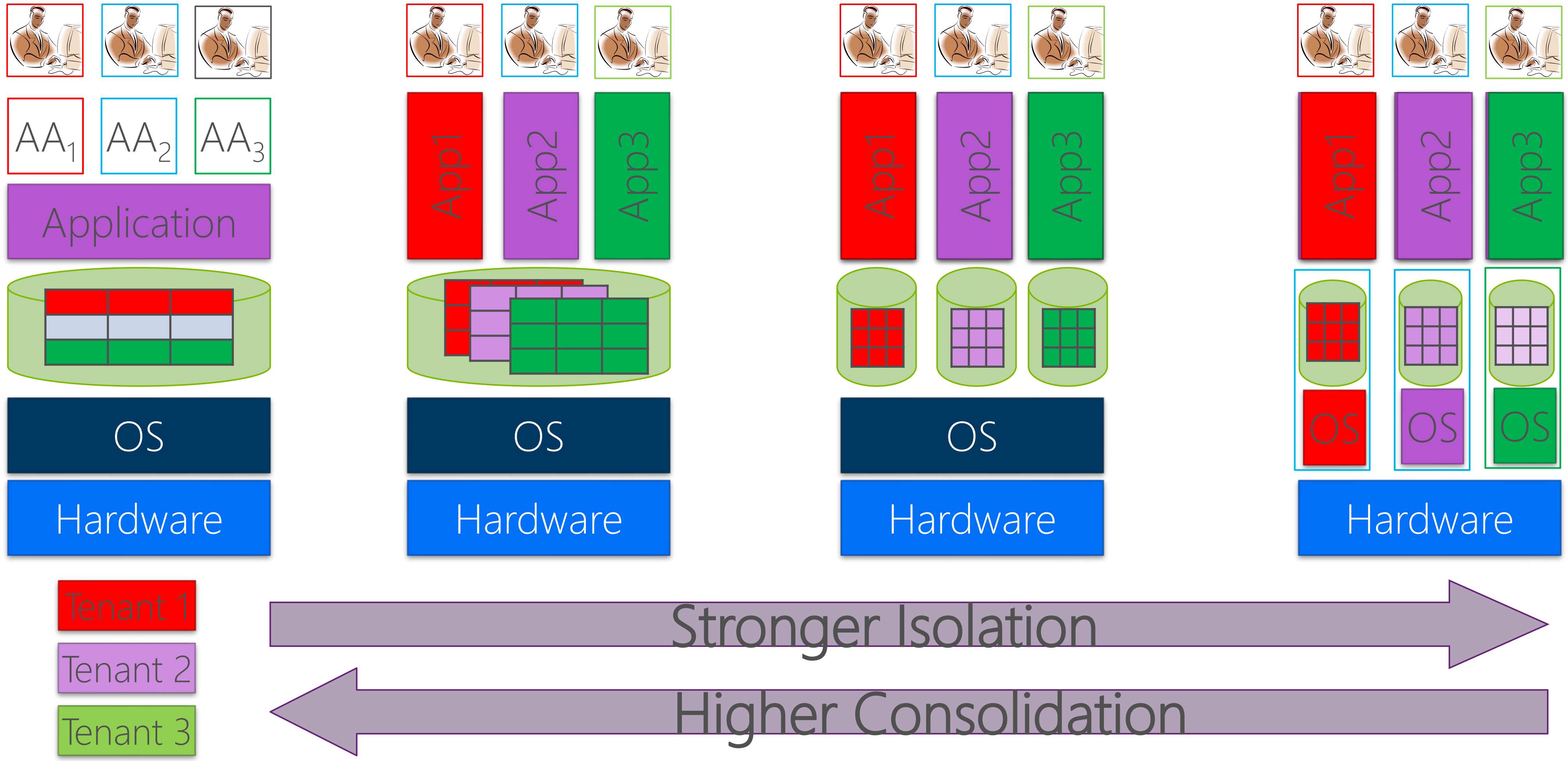
- Formerly known as SQL Azure
- Enterprise-grade Relational Database-as-a-Service

Multi-tenancy in a DaaS

- Multiple tenant databases co-located on a server
- Static resource partitioning is expensive
  - A core per tenant or disk per tenant leads to low consolidation factors
  - Huge demand for databases that cost tens of dollars a *month*
- Low resource utilization with static allocation
  - Many databases often require fraction of a core or a disk
  - A machine per-tenant or core per-tenant is wasteful
- Multi-tenancy is a necessity!

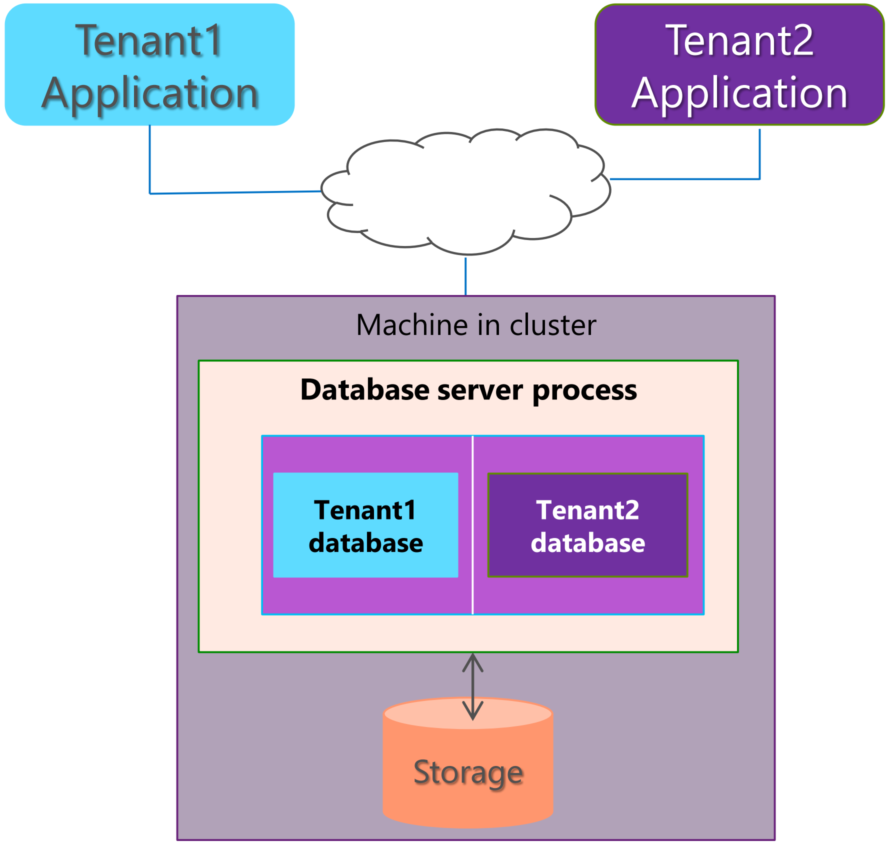
Multi-tenancy models

Resource sharing at different levels of the stack

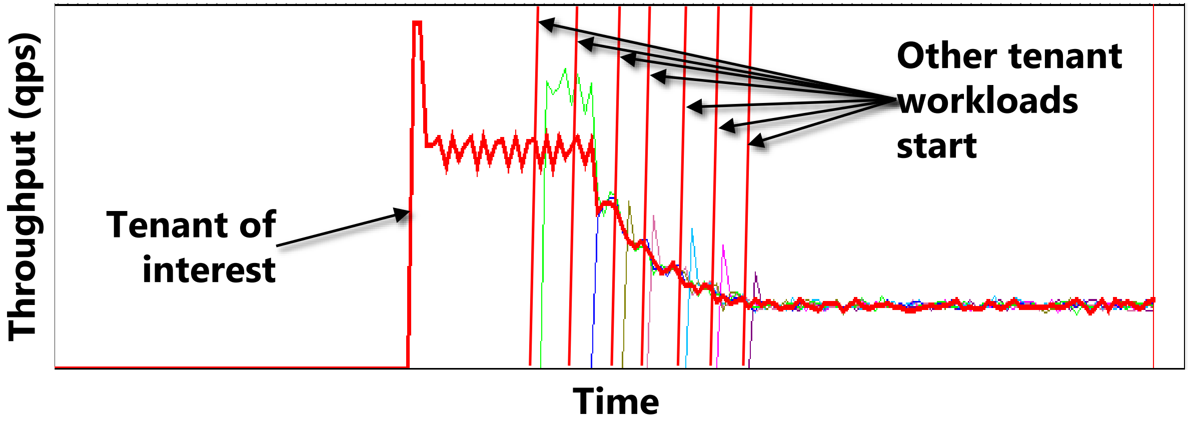


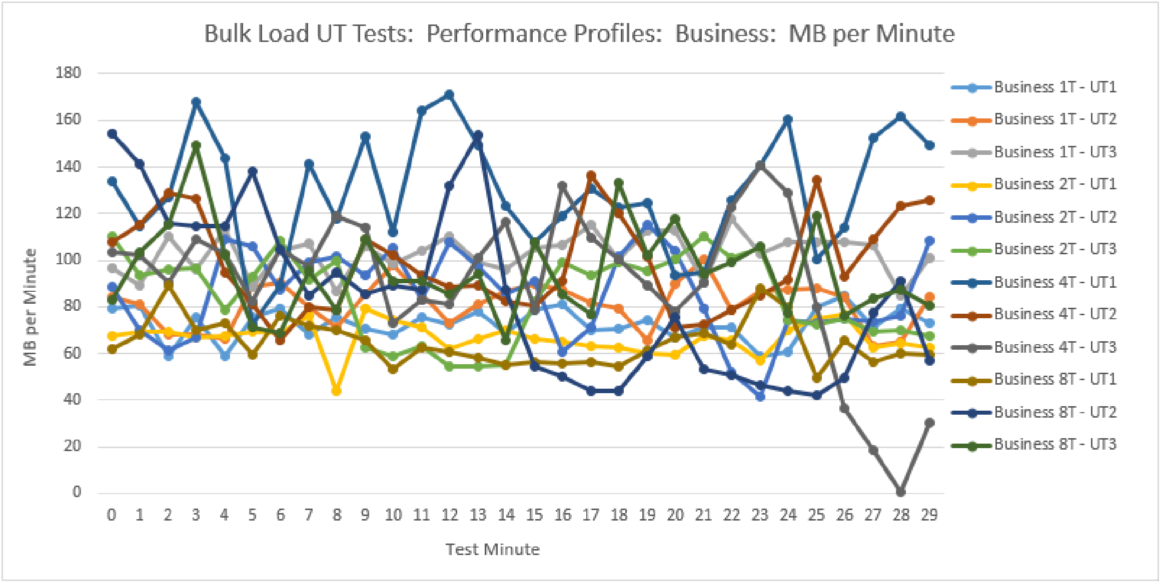
Multi-tenancy in Azure SQL Database

- Queries from a tenant share server's resources with other tenants
- CPU, Memory, I/O, network *shared* across tenants
- *Major concern*: performance of Tenant 1 *affected* by workload of Tenant 2
  - Noisy neighbor
- A major customer pain point



Impact of Noisy Neighbors





https://cbailiss.wordpress.com/2014/09/16/performance-in-new-azure-sql-database-performance-tiers/

What Should Performance Isolation Mean?

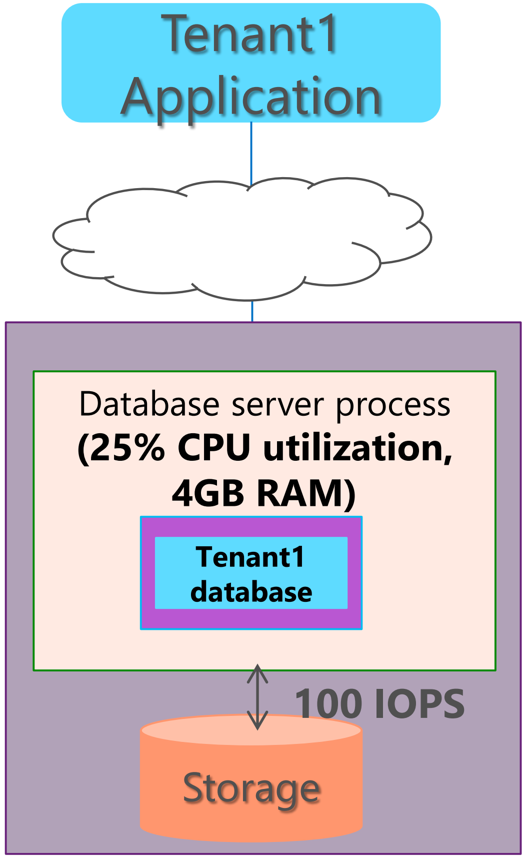
- Tenants *want* performance *unaffected* by *other* tenant workloads
  - Can we promise ***queries/sec*** or ***query latency***?
- Queries can consume *vastly different amounts of resources*

SELECT Product, SUM(Sales) as TotalSales
FROM FactSales F **JOIN** DimProduct P JOIN DimCountry C
ON F.ProdID = P.ProdID and F.CountryId = S.CountryId
WHERE Country= ~~'Honduras'~~ 'China'
GROUP BY Product

- Providers such as Microsoft Azure SQL Database aims to support most existing apps with *rich support for SQL*
  - *Even ad-hoc queries*

SQLVM Project at MSR

[Narasayya et al., CIDR '13], http://bit.ly/sqlvm



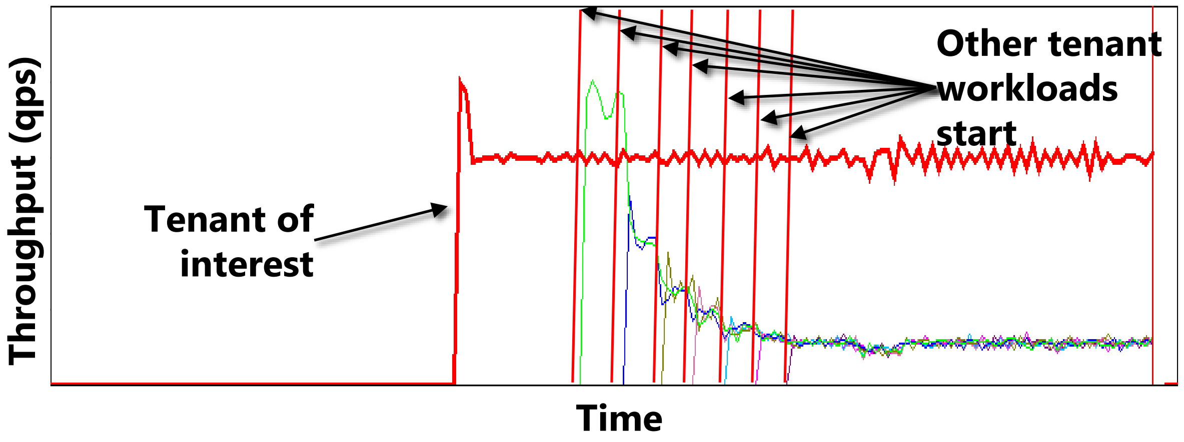
- Tenant is promised *minimum reservation* of DBMS resources
  - *Logical* "resource container" inside DBMS process
  - CPU utilization, IOPS, Memory, …
- Resource governance
  - Fine-grained dynamic resource scheduling mechanisms for CPU, I/O, memory
  - Targeted towards *requirements of multi-tenancy*
- Metering (auditing)
  - Monitor actual and promised resources for tenant
  - Determine *violations*

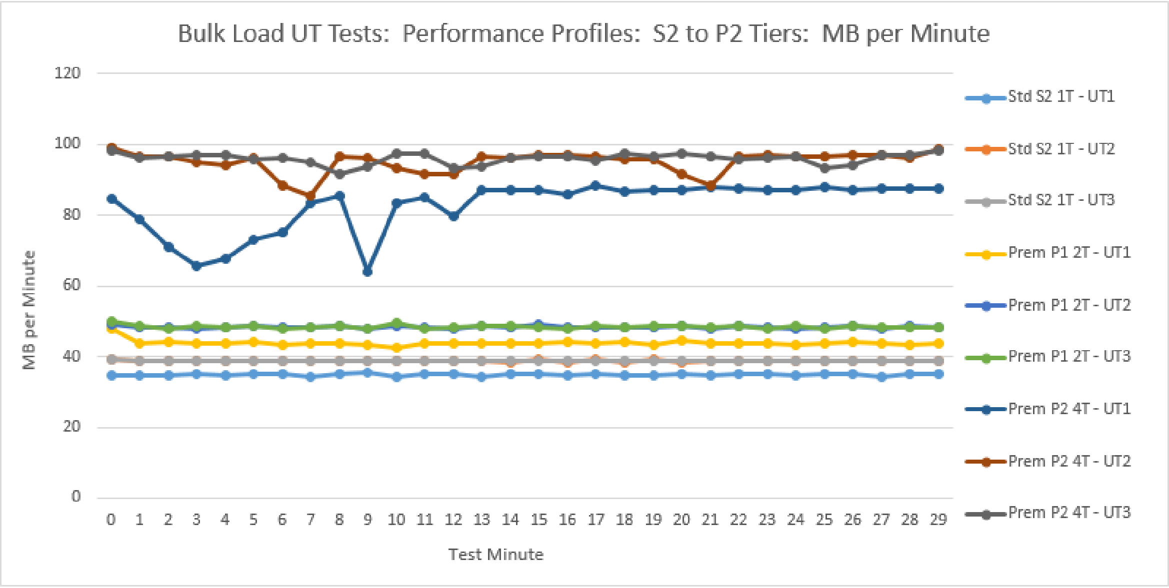
Key Benefits of SQLVM approach

- High degree of isolation from resource demands of co-located tenants
  - E.g. 99th percentile latency unaffected despite many noisy neighbors
- High degrees of consolidation
  - Enables 100s to 1000s of tenant databases on a single node
- Accountability due to metering logic independent of resource governance mechanisms
- Basis for service provider to overbook resources

SQLVM’s Impact

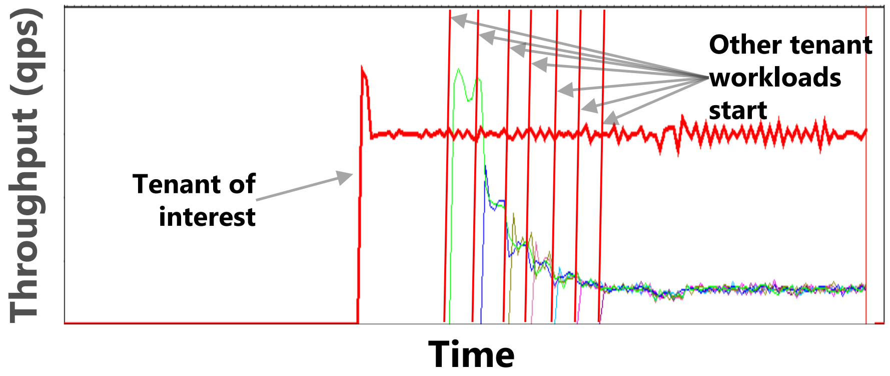
Impact of Performance Isolation

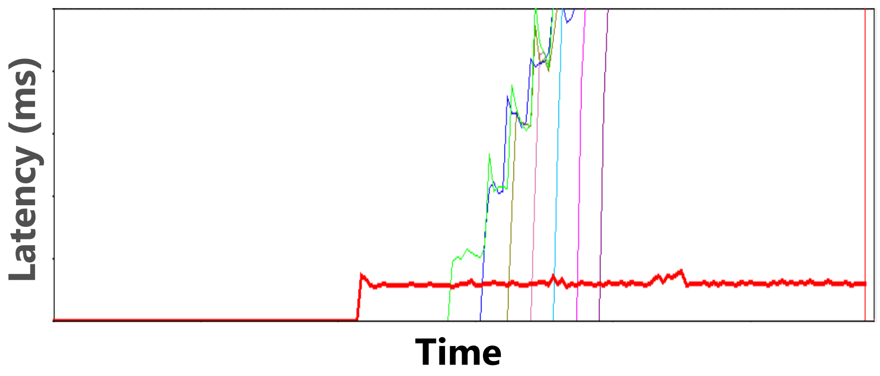


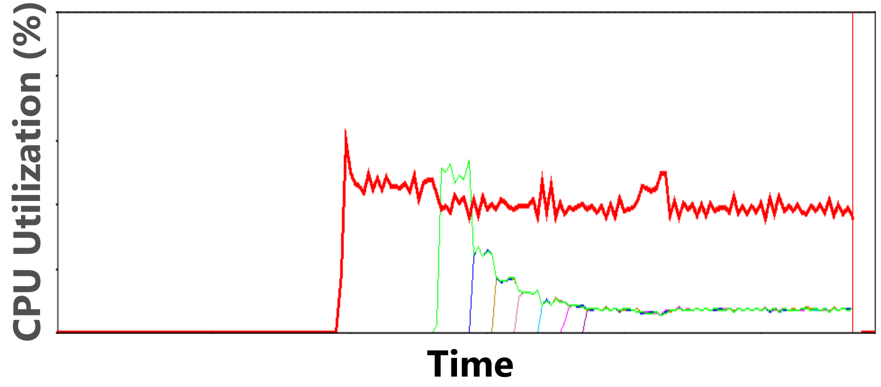


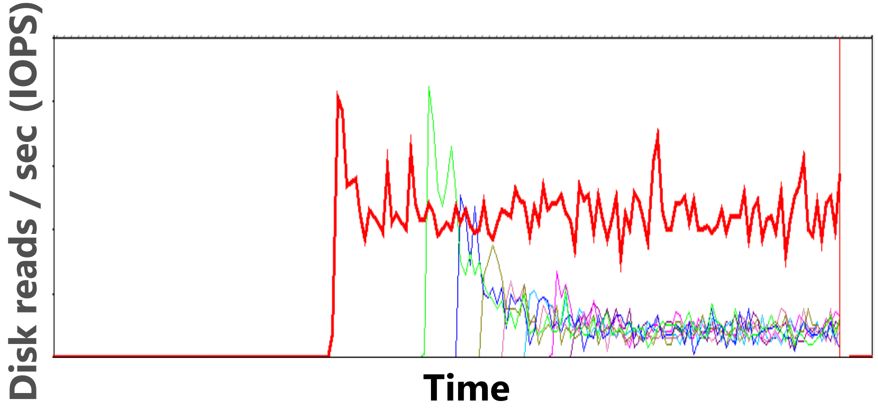
https://cbailiss.wordpress.com/2014/09/16/performance-in-new-azure-sql-database-performance-tiers/

Performance Isolation with SQLVM

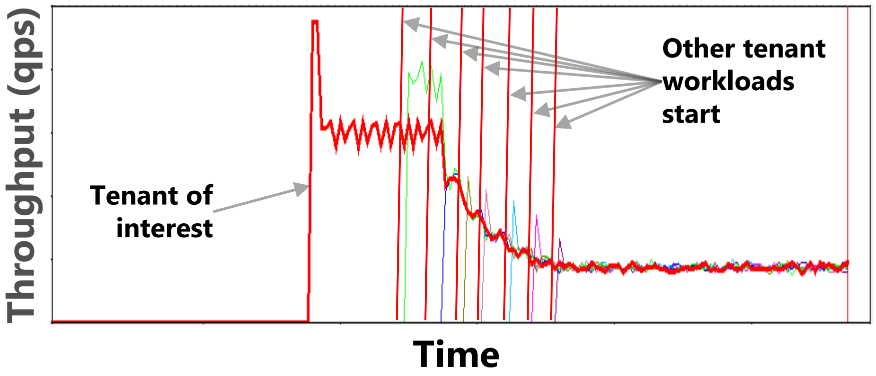


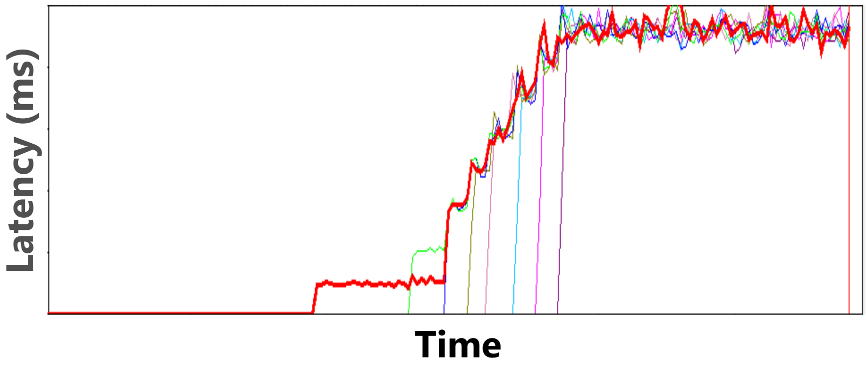


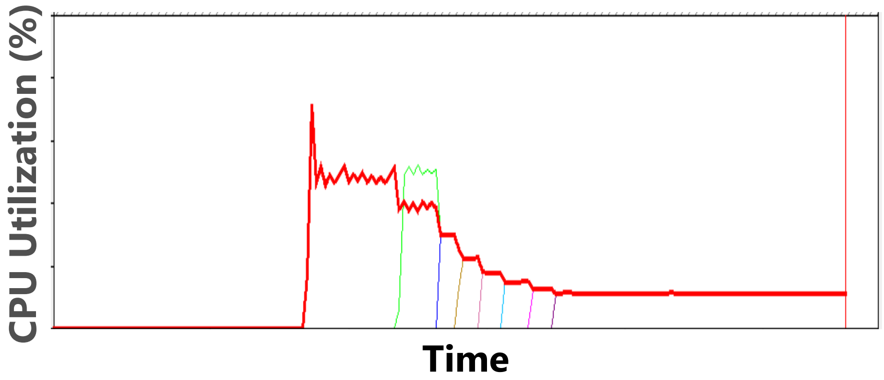


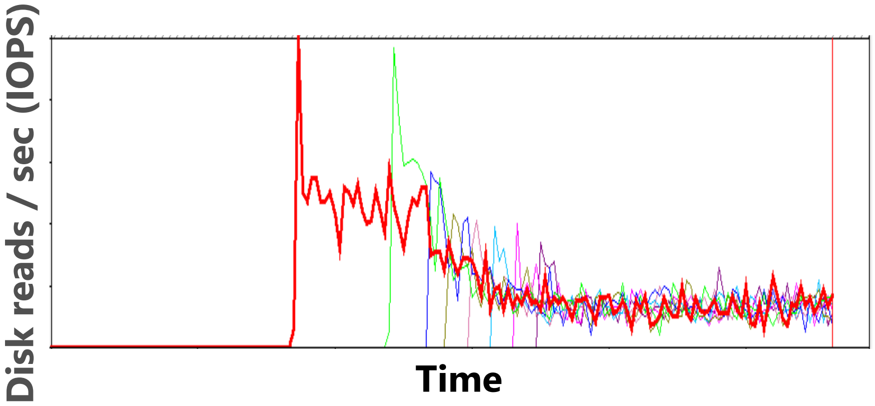


Without Performance Isolation









Business Impact – "Performance Levels"

- Forms the basis for Azure DB's Service Tiers and Performance Levels
  - Generally available since September 2014
- Resource containers to offer performance isolation without requiring static allocation
  - CPU, I/O, memory, transaction log, …
  - CPU, I/O governance, and many more ideas contributed by the SQLVM Project @ MSR
- Supports wide range of tenant workload demands

Tier		DATABASE THROUGHPUT UNITS	DATABASE SIZE	POINT IN TIME RESTORE	PRICE
Basic tier ~ txns/hr	B	5	2 GB	7 Days	$0.0067/hr (~$5/mo)
Standard tier ~ txns/min	S0	10	250 GB	14 Days	$0.0202/hr (~$15/mo)
	S1	20	250 GB	14 Days	$0.0403/hr (~$30/mo)
	S2	50	250 GB	14 Days	$0.1008/hr (~$75/mo)
	S3	100	250 GB	14 Days	$0.2016/hr (~$150/mo)
Premium tier ~ txns/sec	P1	125	500 GB	35 Days	$0.625/hr (~$465/mo)
	P2	250	500 GB	35 Days	$1.25/hr (~$930/mo)
	P3	1000	500 GB	35 Days	$5/hr (~$3,720/mo)

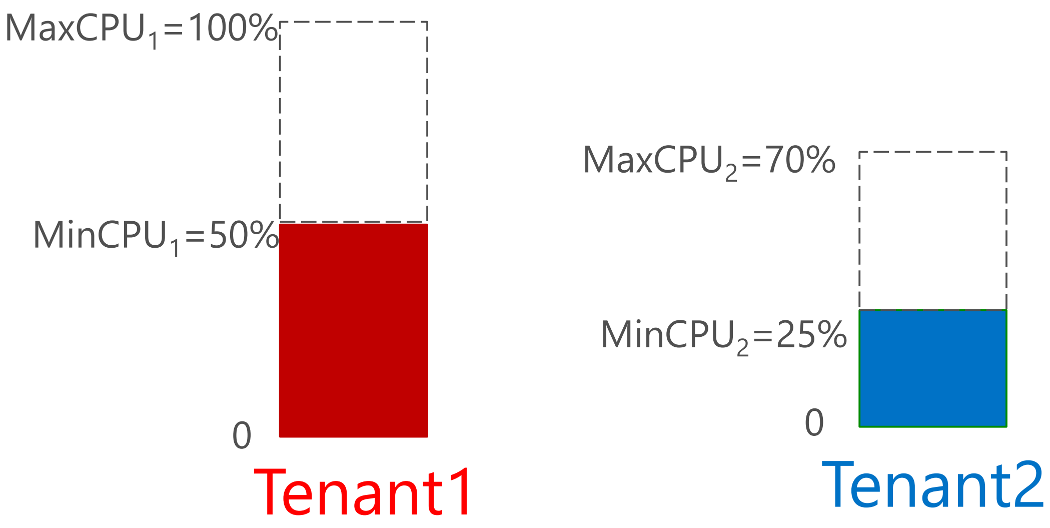
Presentation Outline

- CPU Governance
- Governing other critical resources
  - I/O Governance
  - Memory Governance
- Future directions

CPU Governance

Technical details available in Das et al., VLDB 2014:
"CPU Sharing Techniques in Multi-Tenant
Relational Database-as-a-Service"

CPU Reservations in SQLVM



Reservation of ***CPU utilization at the server***

- Client-facing abstractions may vary

- Reservations guaranteed *without any knowledge of workload*
- *Low latency* for short queries (e.g., logins)
- Non-preemptive scheduling in database kernel
- Scale to hundreds of reservations for co-located tenants
- *Flexible* enough to support *provider-enforced policies*
  - Service-level differentiation, provider's revenue vs. tenant fairness

SQL Server CPU Scheduler 101

- User-mode non-preemptive scheduler
- One scheduler per logical CPU core
- Queries compile to one of more threads
- Once allocated the CPU, threads use a quantum
- Of all threads ready to run, SQL scheduler makes at most one thread runnable per core

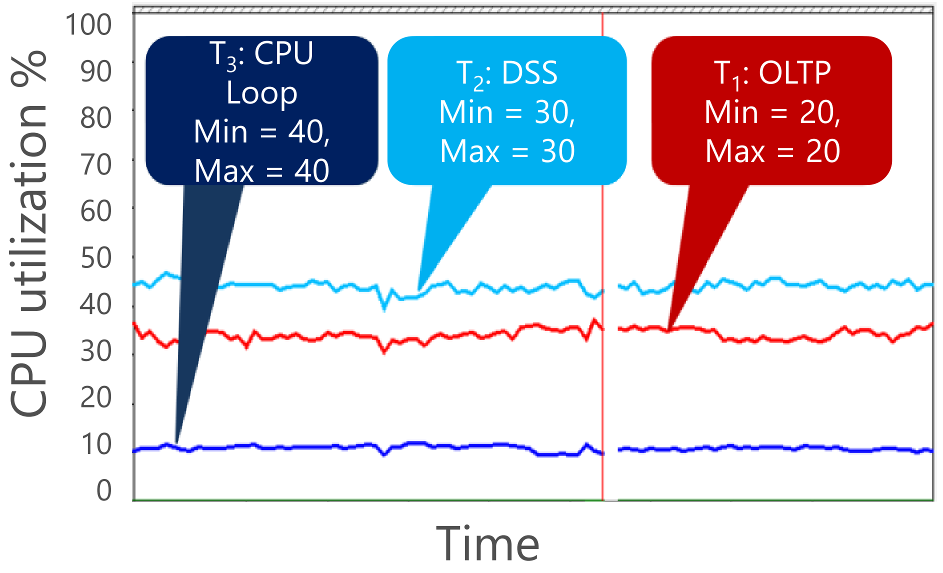
Proportional Sharing is not enough

Variety of database workloads
➔ Highly-variable quantum lengths

T_1: Dell DVD benchmark (OLTP)
Min=20%, Max=20%

T_2: TPC-H (Data warehousing)
Min=30%, Max=30%

T_3: Very short CPU bursts (CPU Loop)
Min=40%, Max=40%



Sharing scheduling opportunities in proportion of MinCPU

Largest Deficit First (LDF) Scheduler

Deficit = Difference between target and actual utilization

- At every context switch, schedule tenant with ***largest deficit (d_i)***

$$d_i = \frac{MinCPU_i - CurCPU_i}{MinCPU_i}$$

Target Utilization

Actual current utilization

Proportional sharing

Feedback

- *Key idea:* Leverage *feedback* from CPU utilization
  - Resilient to quantum length variation
  - Captures tenant utilization across all cores

LDF in action

$MinCPU_1 = MaxCPU_1 = 50\%$

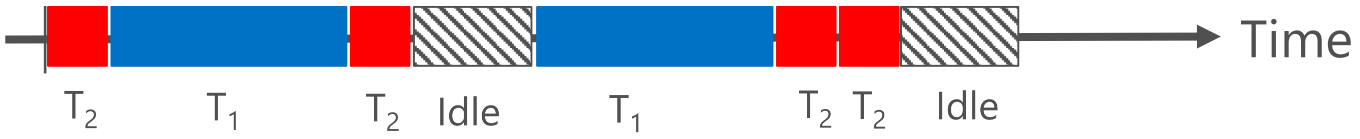
$MinCPU_2 = MaxCPU_2 = 25\%$

T_1's quantum **4X** that of T_2

$$d_i = \frac{MinCPU_i - CurCPU_i}{MinCPU_i}$$

d_1	$\frac{(50-0)}{50} = 1$	$\frac{(50-0)}{50} = 1$	$\frac{(50-80)}{50} = -0.6$	$\frac{(50-66.6)}{50} = -0.33$	$\frac{(50-50)}{50} = 0$
d_2	$\frac{(25-0)}{25} = 1$	$\frac{(25-100)}{25} = -3$	$\frac{(25-20)}{25} = 1$	$\frac{(25-33.3)}{25} = -0.33$	$\frac{(25-25)}{25} = 0$

. . .



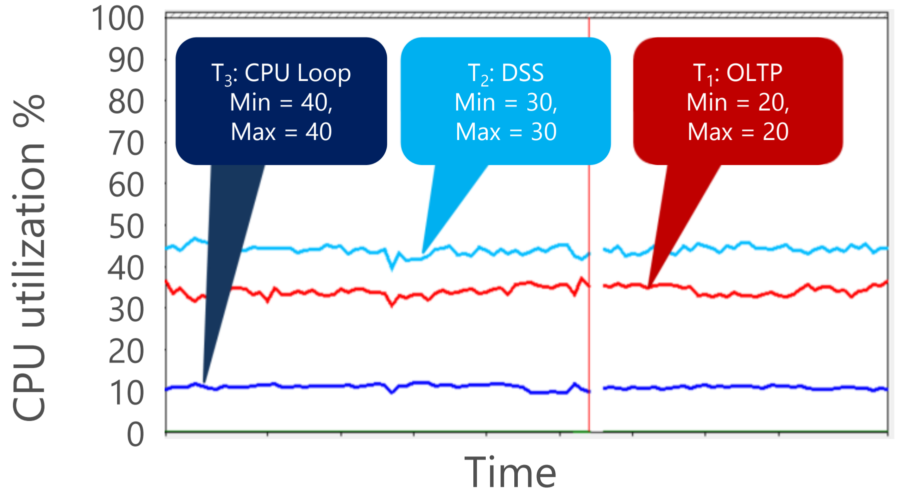
- T_2 get 2X more scheduling opportunities than T_1
- *Guarantees minimum CPU* reservations when demand does not exceed capacity
- *Sharing at a fine time granularity results in better latency response*

Overcoming quantum length variations

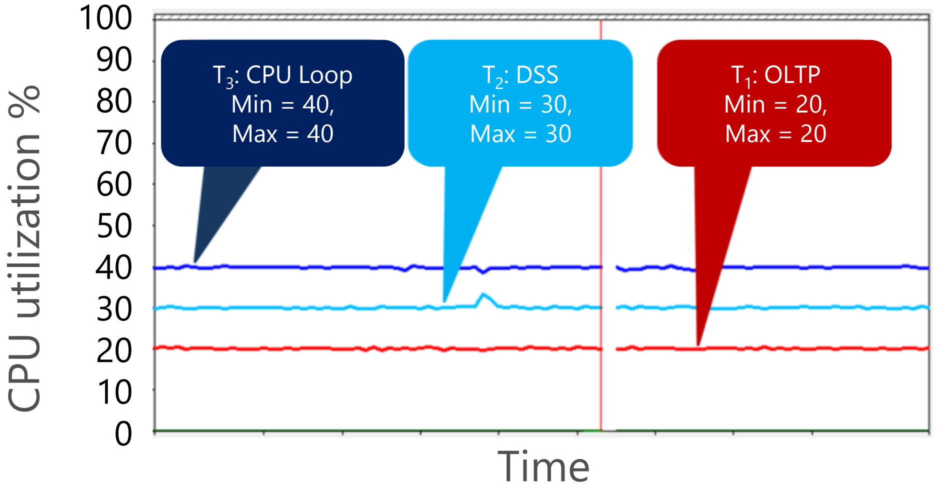
T_1: Dell DVD benchmark (OLTP)

T_2: TPC-H (Data warehousing)

T_3: CPU intensive (very short queries)



Proportional sharing of scheduling opportunities



Largest Deficit First

Properties of LDF

- *Guarantees minimum CPU* reservations when demand does not exceed capacity
- *Global reservations* across multiple cores and sockets
  - Allows one scheduler to catch up for another
- *Dynamic priority work-conserving scheduler*
- *Additional policies by adapting the definition of deficit*

Establishing Accountability

- Differentiate low utilization due to *insufficient demand* from *provider not adequately allocating resources*
  - Factor out idle time without heavy-weight synchronization
- *Intuition*: violation possible by *delaying* T_i's allocation
- $Delay_i$ = T_i's delay as percentage of *metering interval*

$$CPU_i^{Eff} = \frac{CPU_i}{CPU_i + Delay_i}$$

- *Numerator:* CPU used; *Denominator:* active time
- *Violation if and only if* $\boldsymbol{CPU_i^{Eff} < MinCPU_i}$

Evaluation

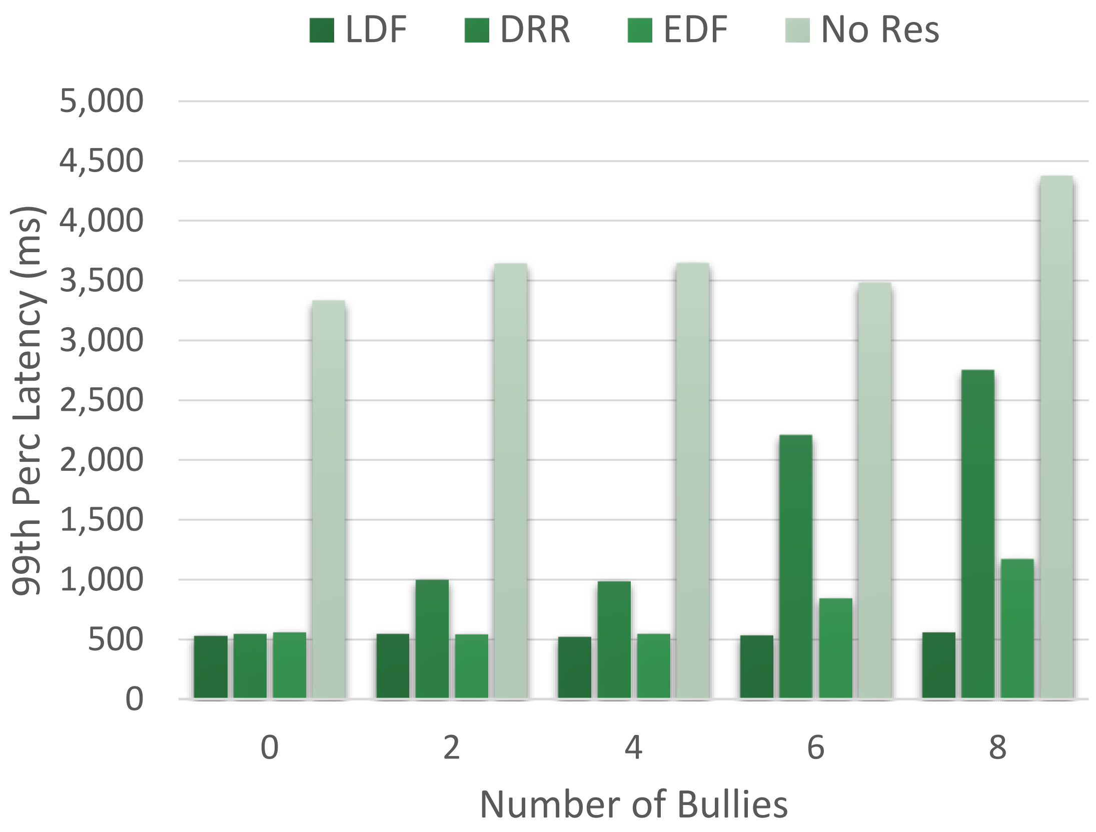
- Detailed evaluation using TPC-C, TPC-H, Dell DVD Store, and a CPU-IO micro benchmark workloads
- Highlights:
  - *Meets reservations* when no overbooking
  - Provides *excellent performance isolation*
    - Negligible effect on other tenant's 99^{th} percentile latency
- More details in the VLDB 2014 paper

Other approaches

- *Deficit Round Robin (DRR)* [Shreedhar & Varghese, 1996]
  - Use the *same deficit formula* as LDF
  - *Round robin scheduling* instead of LDF's greedy approach
- *Earliest Deadline First (EDF)* [Liu & Layland, 1973]
  - Adaptation of a variant used in Xen's Atropos scheduler can be adapted to our setting [Cherkasova et al., 2007]
  - Use the absolute deficit $(MinCPU_i - CurCPU_i)$
  - *Different deficit formula*, but *greedy similar* to LDF

Excellent Performance Isolation

- *Eight tenants* with CPU reservations (MIN=MAX)
  - T1: 5%, T_2-T_4:8%, T_5-T_7: 10%, *T_8: 25%*; **85% capacity reservation**
  - All tenants executing *CPU-IO benchmark*; server running at ~95% utilization
- Up to *eight* ***bully*** *workloads:* generate high demand for CPU, no reservations

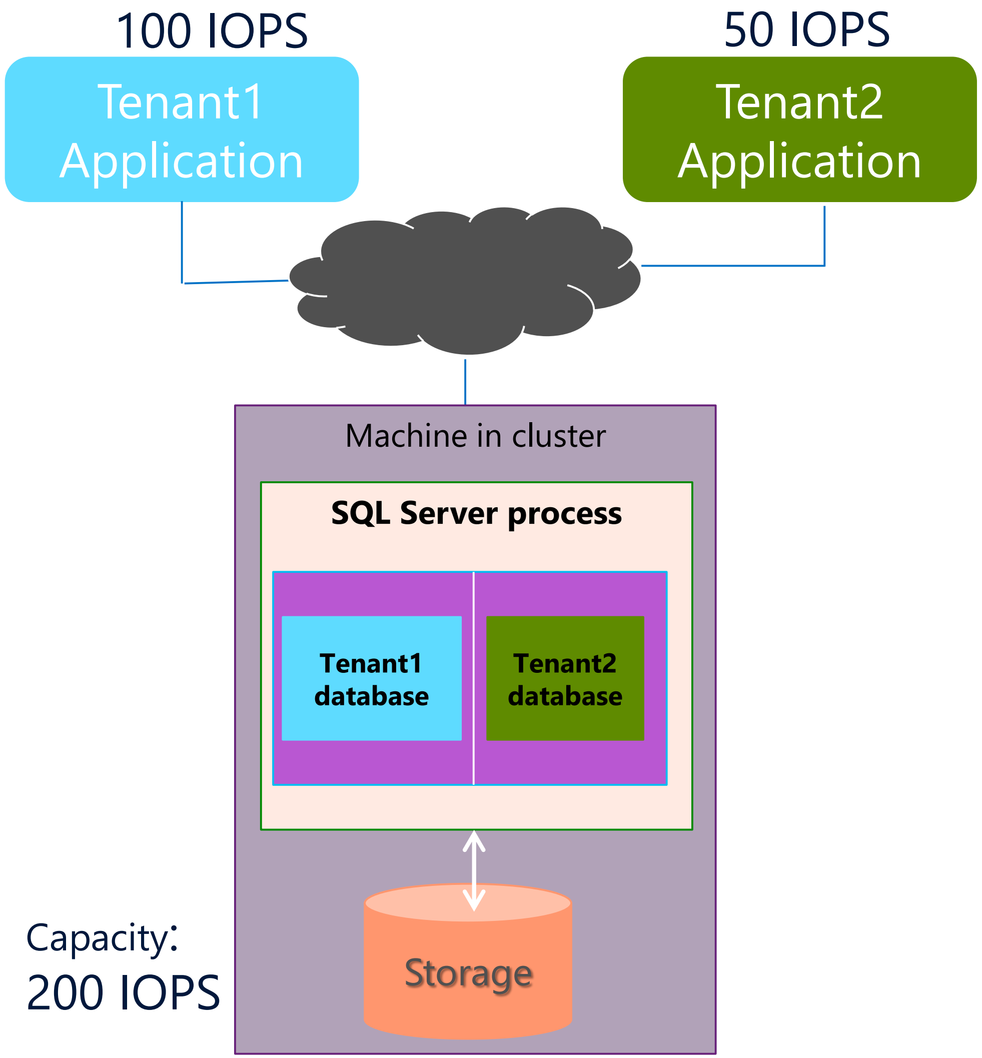


T_8's latency

Other Resources

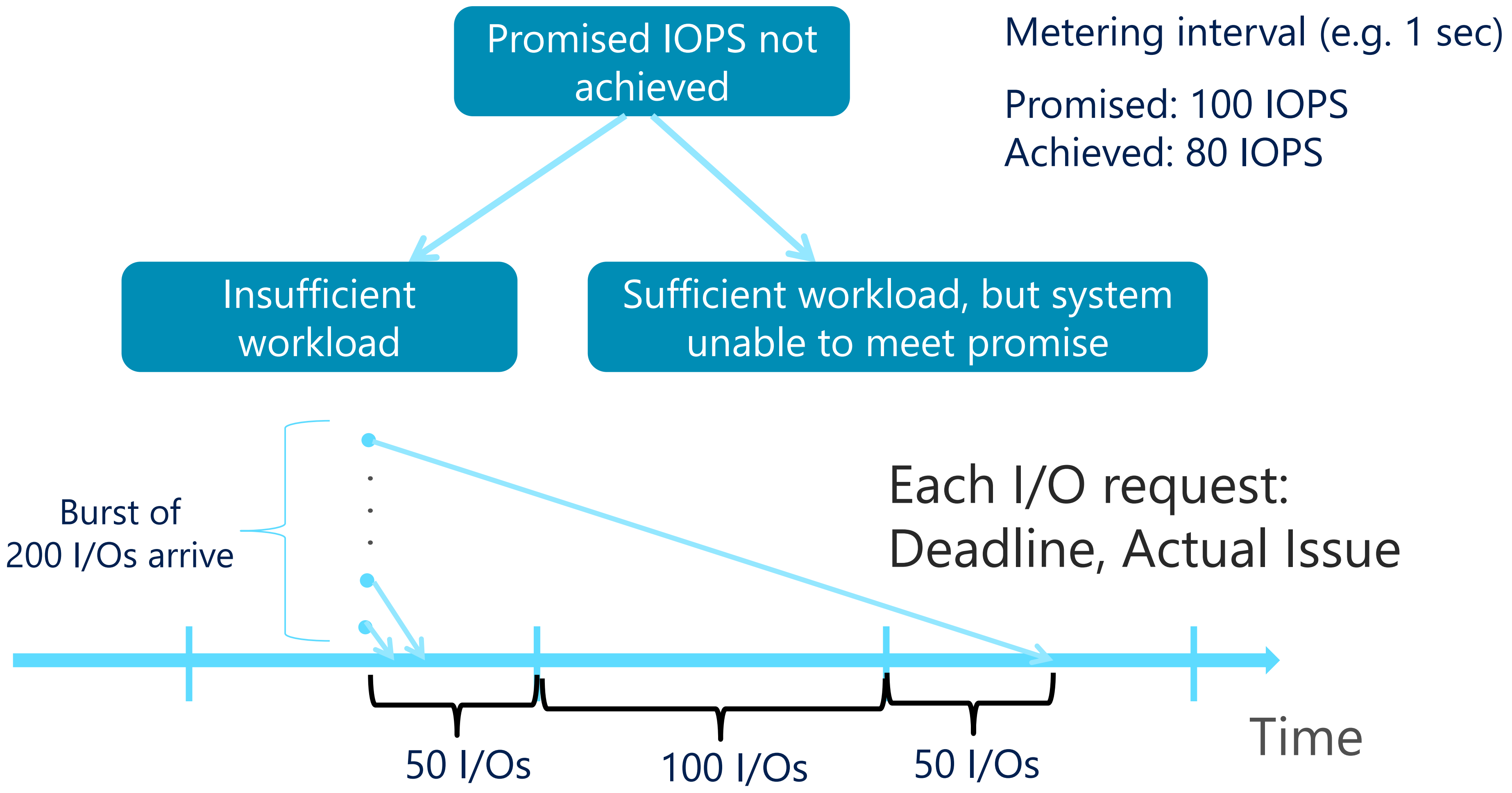
I/O: Details in Narasayya et al., CIDR 2013 Paper
Bufferpool memory: Details in upcoming paper
Narasayya et al., VLDB 2015

I/O Governance

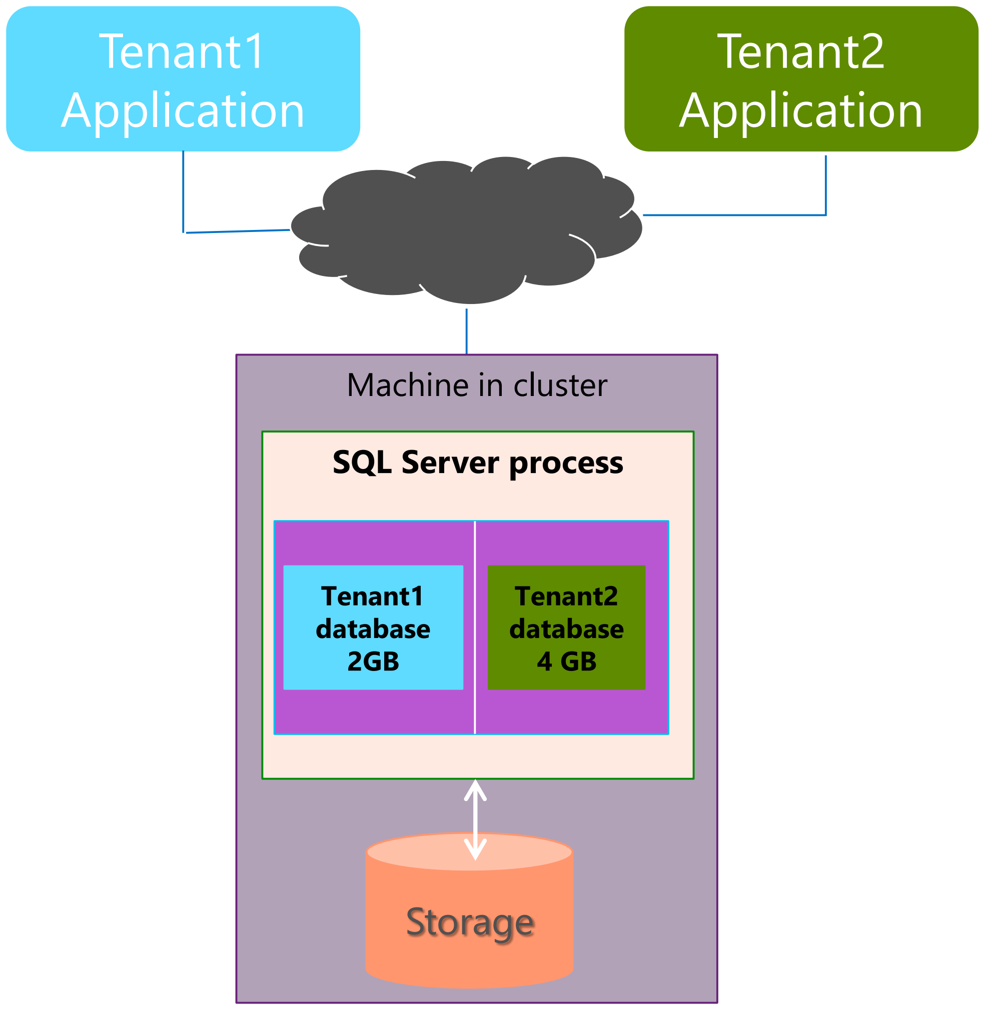


- Challenges
  - Bursty I/O patterns
  - Coordinating tenant I/Os across cores
  - Capturing I/Os issued indirectly on tenant's behalf
- Key idea: Shape I/O traffic
  - 50 IOPS ⇒ one I/O every 20 msec
  - I/O request tagged with deadline
  - Issue I/Os whose deadline has arrived

Establishing Accountability



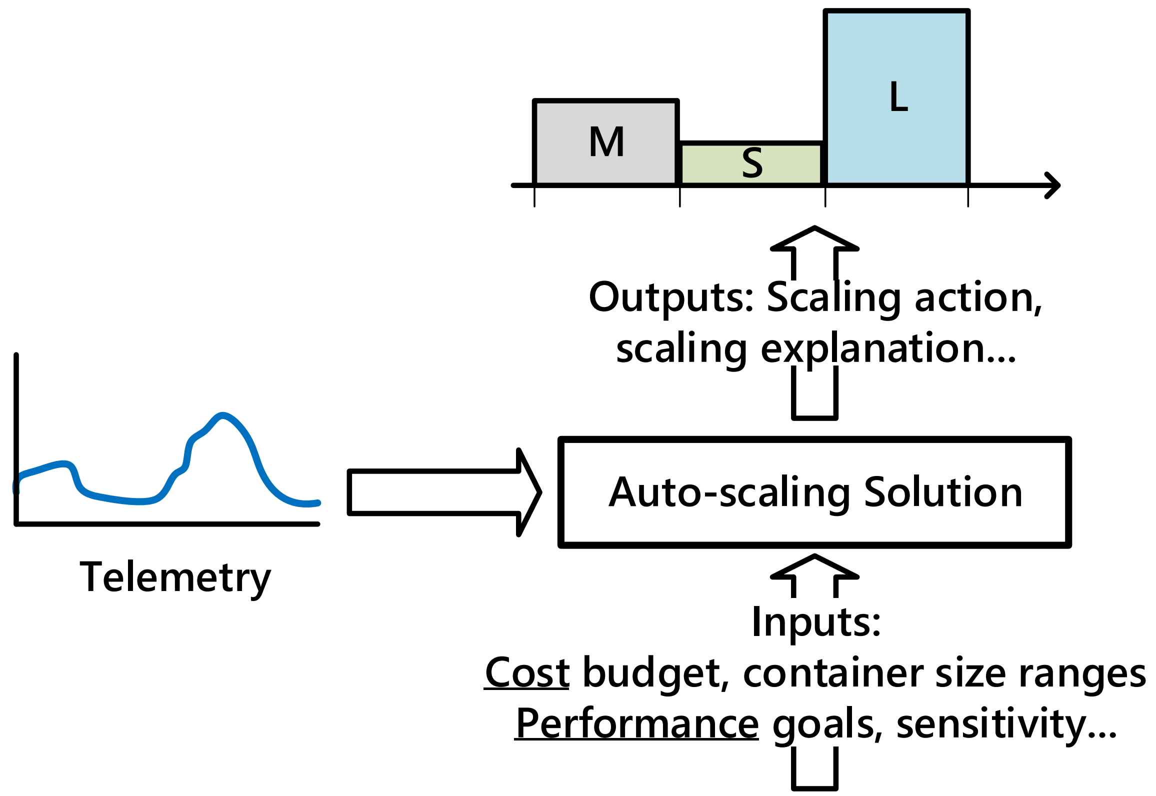
Buffer pool Memory



- Bufferpool caches "hot" database pages
  - Crucial for application's performance
- Memory reservation
  - Min: 2GB, Max: 4GB
  - No static memory allocation
- **Accountability**: Page hit ratio as if the reserved memory was statically allocated
- LRU-k based policies need to be reservation-aware
  - Ideas adapted from online caching

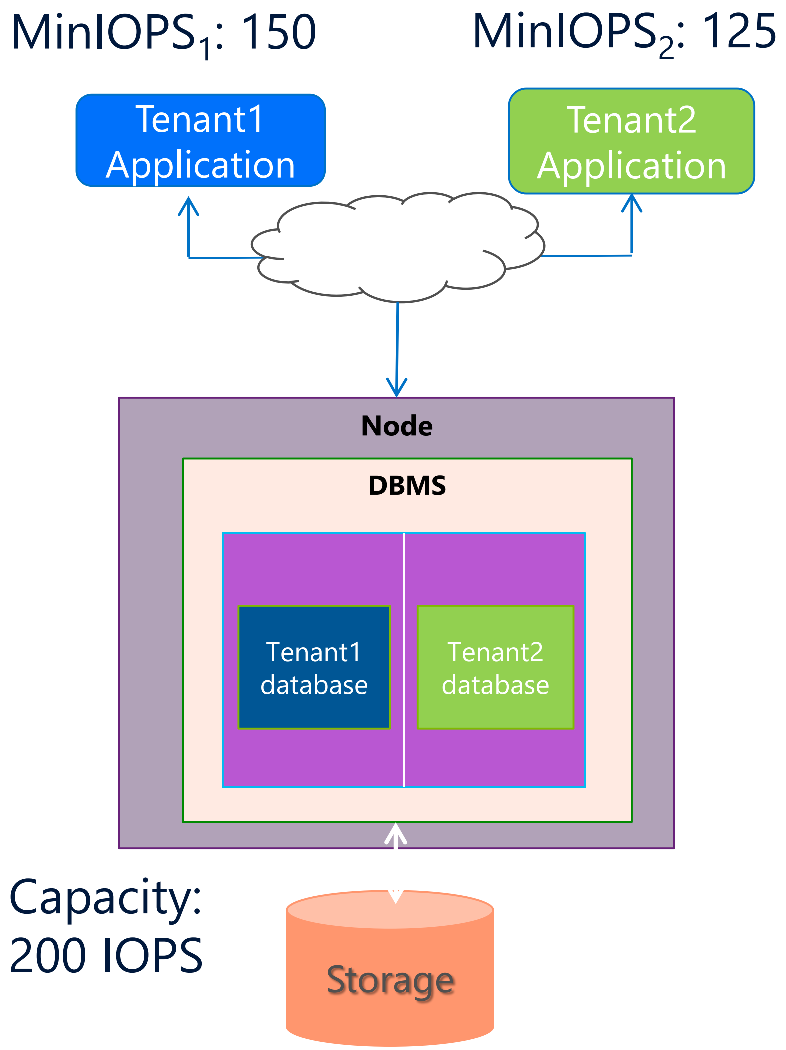
Future Directions

Automatic Dynamic Resource Provisioning



- Automatically and dynamically scale a database's performance level on tenant's behalf
- Challenges:
  - For database workloads, there is complex interplay of resources, performance and price
  - How much resources does the workload need?
    - Resource demand cannot be measured
    - What is the abstraction exposed to tenants?

Overbooking Resources



- Summation of reservations exceeds capacity
  - Similar to overbooking in airlines
- Tenant promises may be violated
  - Penalty if violation
- Questions
  - How much to overbook?
  - Tenant placement/movement
- Objectives
  - Minimize penalty, fairness

Concluding Remarks

- *Multi-tenancy* is essential in relational database-as-a-service
- Microsoft Azure DB supports *performance service tiers* without requiring static resource allocation
  - New resource governance and metering mechanisms developed in the *SQLVM Project @ Microsoft Research*
- Building block for higher-level performance SLAs in a shared cloud infrastructure
- More information: http://bit.ly/sqlvm

DMX Group @ MSR

- Data Platforms
  - Service Intelligence
  - Hyder
  - Auto-admin

- Data Explorations
  - Structured data search
  - Synonym mining
  - Data cleaning

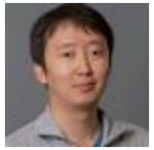













Questions?

More information:
http://bit.ly/sqlvm

Microsoft