Performance Isolation and Fairness for Multi-Tenant Cloud Storage

David Shue*, Michael Freedman*, and Anees Shaikh†

*Princeton  †IBM Research
Setting: Shared Storage in the Cloud

- yelp
- zynga
- foursquare
- TypePad
- Amazon Web Services
Setting: Shared Storage in the Cloud

- Yelp
- Zynga
- Foursquare
- TypePad
- Amazon Web Services
- S3
- EBS
- SQS
Setting: Shared Storage in the Cloud
Predictable Performance is Hard

Multiple co-located tenants $\Rightarrow$ resource contention
Predictable Performance is Hard

Multiple co-located tenants $\Rightarrow$ resource contention
Predictable Performance is Hard

Fair queuing @ big iron

Multiple co-located tenants ⇒ resource contention
Predictable Performance is Hard

Multiple co-located tenants $\Rightarrow$ resource contention
Distributed system $\Rightarrow$ distributed resource allocation
Predictable Performance is Hard

Multiple co-located tenants $\Rightarrow$ resource contention
Distributed system $\Rightarrow$ distributed resource allocation
Predictable Performance is Hard

Multiple co-located tenants ⇒ resource contention
Distributed system ⇒ distributed resource allocation
Predictable Performance is Hard

Multiple co-located tenants $\Rightarrow$ resource contention
Distributed system $\Rightarrow$ distributed resource allocation
Skewed object popularity $\Rightarrow$ variable per-node demand
Predictable Performance is Hard

Multiple co-located tenants ⇒ resource contention
Distributed system ⇒ distributed resource allocation
Skewed object popularity ⇒ variable per-node demand
Disparate workloads ⇒ different bottleneck resources
Tenants Want System-wide Resource Guarantees

Multiple co-located tenants ⇒ resource contention
Distributed system ⇒ distributed resource allocation
Skewed object popularity ⇒ variable per-node demand
Disparate workloads ⇒ different bottleneck resources
Tenant Want System-wide Resource Guarantees

Multiple co-located tenants \( \Rightarrow \) resource contention
Distributed system \( \Rightarrow \) distributed resource allocation
Skewed object popularity \( \Rightarrow \) variable per-node demand
Disparate workloads \( \Rightarrow \) different bottleneck resources
Pisces Provides Weighted Fair-shares

$w_z = 20\%$

Zynga

$w_y = 30\%$

Yelp

$w_t = 10\%$

TP

$w_f = 40\%$

Foursquare

Multiple co-located tenants $\Rightarrow$ resource contention

Distributed system $\Rightarrow$ distributed resource allocation

Skewed object popularity $\Rightarrow$ variable per-node demand

Disparate workloads $\Rightarrow$ different bottleneck resources
Pisces: Predictable Shared Cloud Storage

• Pisces
  - Per-tenant max-min fair shares of system-wide resources
    ~ min guarantees, high utilization
  - Arbitrary object popularity
  - Different resource bottlenecks
Pisces: Predictable Shared Cloud Storage

• Pisces
  - Per-tenant max-min fair shares of system-wide resources
    ~ min guarantees, high utilization
  - Arbitrary object popularity
  - Different resource bottlenecks

• Amazon DynamoDB
  - Per-tenant provisioned rates
    ~ rate limited, non-work conserving
  - Uniform object popularity
  - Single resource (1kB requests)
Predictable Multi-Tenant Key-Value Storage
Predictable Multi-Tenant Key-Value Storage

Tenant A

VM
VM
VM

GET 1101100

Tenant B

VM
VM
VM

Controller

PP
Predictable Multi-Tenant Key-Value Storage

Tenant A

VM  VM  VM

GET 1101100

Tenant B

VM  VM  VM

Controller

PP

RS
Predictable Multi-Tenant Key-Value Storage
Predictable Multi-Tenant Key-Value Storage
Predictable Multi-Tenant Key-Value Storage

Weight\textsubscript{A}

Tenant A

VM VM VM

GET 1101100

Weight\textsubscript{B}

Tenant B

VM VM VM

Controller

RR

WA\textsubscript{2} WB\textsubscript{2}

PP

WA

RS

FQ

WA\textsubscript{1} WB\textsubscript{1}
Strawman: Place Partitions Randomly

Weight\textsubscript{A} Tenant A

Weight\textsubscript{B} Tenant B

Controller

PP

WA W\textsubscript{A2} W\textsubscript{B2}

RS

FQ
Strawman: Place Partitions Randomly

Weight\textsubscript{A}

Tenant A

VM VM VM

Weight\textsubscript{B}

Tenant B

VM VM VM

Overloaded

Controller

PP

WA \ W\textsubscript{A2} \ W\textsubscript{B2}

RS

FQ
Pisces: Place Partitions By Fairness Constraints

Weight\textsubscript{A} Tenant A

Weight\textsubscript{B} Tenant B

Collect per-partition tenant demand

Controller
Pisces: Place Partitions By Fairness Constraints

Weight\textsubscript{A} Tenant A VM VM VM

Weight\textsubscript{B} Tenant B VM VM VM

Collect per-partition tenant demand

Controller

Bin-pack partitions
Pisces: Place Partitions By Fairness Constraints

Weight\textsubscript{A}  
Tenant A  
VM VM VM

Weight\textsubscript{B}  
Tenant B  
VM VM VM

Controller

Results in feasible partition placement
Pisces: Place Partitions By Fairness Constraints

Results in feasible partition placement
Strawman: Allocate Local Weights Evenly

\[ W_{A1} = W_{B1} \]
\[ W_{A2} = W_{B2} \]
Strawman: Allocate Local Weights Evenly

Weight_A
Tenant A
VM VM VM

Weight_B
Tenant B
VM VM VM

WA = WB

Controller

WA1 = WB1

WA2 = WB2
Strawman: Allocate Local Weights Evenly

Weight\textsubscript{A} = W\textsubscript{B1}

Weight\textsubscript{B} = W\textsubscript{B2}

Overloaded

\[ W_{A1} = W_{B1} \]
Pisces: Allocate Local Weights By Tenant Demand

Weight\textsubscript{A}

Tenant A

VM VM VM

Weight\textsubscript{B}

Tenant B

VM VM VM

Controller

W\textsubscript{A1} = W\textsubscript{B1}

W\textsubscript{A2} = W\textsubscript{B2}
Pisces: Allocate Local Weights By Tenant Demand

Weight_A
Tenant A
VM VM VM

Weight_B
Tenant B
VM VM VM

max mismatch
Compute per-tenant +/- mismatch

Controller

W_{A1} = W_{B1}

W_{A2} = W_{B2}
Pisces: Allocate Local Weights By Tenant Demand

Weight\textsubscript{A} > Weight\textsubscript{B}

Compute per-tenant +/- mismatch

Controller

W\textsubscript{A1} > W\textsubscript{B1}

W\textsubscript{A2} = W\textsubscript{B2}
Pisces: Allocate Local Weights By Tenant Demand

Weight_A
Tenant A
VM VM VM

Weight_B
Tenant B
VM VM VM

Compute per-tenant +/- mismatch

Controller

Reciprocal weight swap

W_A1 > W_B1

W_A2 < W_B2

A ← B

A → B
Strawman: Select Replicas Evenly

Weight$_A$

Tenant A

VM VM VM

GET 1101100

RR

50%

$W_{A1} > W_{B1}$

Controller

Weight$_B$

Tenant B

VM VM VM

50%

$W_{A2} < W_{B2}$
Strawman: Select Replicas Evenly

Weight\textsubscript{A}

Tenant A

VM VM VM

Get 1 1 0 1 1 0 0

Weight\textsubscript{B}

Tenant B

VM VM VM

Controller

\text{WA} \quad W_{A2} \quad W_{B2}

\text{PP}

\text{RS}

\text{FQ}

W_{A1} > W_{B1}

W_{A2} < W_{B2}
Pisces: Select Replicas By Local Weight

Weight\textsubscript{A} \quad Tenant A \quad VM \quad VM \quad VM

Weight\textsubscript{B} \quad Tenant B \quad VM \quad VM \quad VM

Controller

$W_{A1} > W_{B1}$

$W_{A2} < W_{B2}$
Pisces: Select Replicas By Local Weight

- **Tenant A**
  - VMs: 3
  - Weight: $W_A$
  - GET: 1101100
  - $W_{A1} > W_{B1}$

- **Tenant B**
  - VMs: 3
  - Weight: $W_B$
  - GET: 1101100
  - $W_{A2} < W_{B2}$

- Detect weight mismatch by request latency
- Controller
  - 50% for $W_A$
  - 50% for $W_B$
Pisces: Select Replicas By Local Weight

Tenant A

Weight$_A$

VM VM VM

Tenant B

Weight$_B$

VM VM VM

Controller

GET 110100

60%

40%

detect weight mismatch by request latency

$W_{A1} > W_{B1}$

$W_{A2} < W_{B2}$
Pisces: Select Replicas By Local Weight

Tenant A

**Weight**$_A$

VM VM VM

 Tenant B

**Weight**$_B$

VM VM VM

Controller

GET 1101100

RR

60% 40%

W$_A1$ > W$_B1$

W$_A2$ < W$_B2$

detect weight mismatch by request latency
Strawman: Queue Tenants By Single Resource

Tenant A

VM VM VM

\( W_{A1} > W_{B1} \)

Tenant B

VM VM VM

\( W_{A2} < W_{B2} \)

Controller

GET 0100111
GET 1101100

RS

PP

WA

WA2 WB2

FQ
Strawman: Queue Tenants By Single Resource

 Tenant A

 VM VM VM

 $W_{A1} > W_{B1}$

 Tenant B

 VM VM VM

 $W_{A2} < W_{B2}$

 Controller

 RR

 FQ
Strawman: Queue Tenants By Single Resource

Tenant A
- Bandwidth limited

Tenant B
- Request Limited

Controller

WS2 < WB2
Strawman: Queue Tenants By Single Resource

Tenant A
- Bandwidth limited
- bottleneck resource (out bytes) fair share

Tenant B
- Request Limited

Controller

\[ W_{A2} < W_{B2} \]
Pisces: Queue Tenants By Dominant Resource

Tenant A
VM VM VM
Bandwidth limited

Tenant B
VM VM VM
Request Limited

Controller

Track per-tenant resource vector

\( W_{A2} < W_{B2} \)
Pisces: Queue Tenants By Dominant Resource

Tenant A
- Bandwidth limited
- Dominant resource: Fair share
- Track per-tenant resource vector

Tenant B
- Request Limited
- Controller

Track per-tenant resource vector

WA2 < WB2
Pisces: Queue Tenants By Dominant Resource

Tenant A
Bandwidth limited
dominant resource fair share
Track per-tenant resource vector
out req out req

Tenant B
Request Limited

Controller

WA WA2 WB2
RS

FQ

WA2 < WB2

RR
Pisces Mechanisms Solve For Global Fairness

- **System Visibility**
  - **Timescale**
  - **Pisces**: RS, WA, FQ, PP
  - W_{A2}, W_{B2}

---

23
Pisces Mechanisms Solve For Global Fairness
Pisces Mechanisms Solve For Global Fairness

System Visibility

Controller

global

Timescale

seconds

minutes

FQ

RS

WA

PP

WA

WB2

WP
Pisces Mechanisms Solve For Global Fairness

System Visibility

local
global
Controller

Timescale

microseconds
seconds
minutes

FQ
RS
WA
PP

WA2 W_B2
Pisces Mechanisms Solve For Global Fairness

System Visibility
- Global
- Local

Controller
- RR
- SS

Timescale
- Microseconds
- Seconds
- Minutes

Weight Allocations
- WA
- RS
- FQ

Replica Selection Policies
- WA2
- WB2
- PP
Pisces Mechanisms Solve For Global Fairness
Pisces Mechanisms Solve For Global Fairness

System Visibility

local

global

Controller

RR

RR

SS

SS

Replica Selection Policies

Weight Allocations

Timescale

microseconds

seconds

minutes

PQ

WA

WP

WA

WB
Pisces Mechanisms Solve For Global Fairness

System Visibility
- local
- global

Controller
- RR
- RR
- SS
- SS

Microseconds
- FQ
- RS

Seconds
- WA

Minutes
- PP

Weight Allocations
- $W_{A2}$
- $W_{B2}$

Timescale
- Fairness and capacity constraints

Replica Selection Policies
Pisces Mechanisms Solve For Global Fairness

System Visibility

Local

Controller

Global

Replica Selection Policies

Weight Allocations

fairness and capacity constraints

Timescale

microseconds

seconds

minutes

Fairness and capacity constraints

Pisces Mechanisms Solve For Global Fairness

System Visibility

Local

Controller

Global

Replica Selection Policies

Weight Allocations

fairness and capacity constraints

Timescale

microseconds

seconds

minutes

Fairness and capacity constraints
Pisces Mechanisms Solve For Global Fairness

System Visibility

- local
  - RR
  - SS
- global
  - Controller

Replica Selection Policies

Weight Allocations

Timescale

- microseconds
- seconds
- minutes

Controller allocation: WA, WA2, WB2

Timescale: Pisces Mechanisms solve for global fairness in the context of system visibility ranging from local to global. The diagram illustrates weight allocations and replica selection policies across different timescales, aiming for fairness and capacity constraints.
Pisces Mechanisms Solve For Global Fairness

System Visibility
- local
- global

Controller
- RR
- SS

Timescale
- microseconds
- seconds
- minutes

Replica Selection Policies
- Weight Allocations
- fairness and capacity constraints

Weight Allocations
- WA
- PP
- W_{A2} W_{B2}
Pisces Mechanisms Solve For Global Fairness

System Visibility

- local
- RS
- SS
- Controller (global)

Timescale
- microseconds
- seconds
- minutes

Replica Selection Policies

Weight Allocations

fairness and capacity constraints

WA, W_{A2}, W_{B2}

PP
Pisces Mechanisms Solve For Global Fairness
Pisces Mechanisms Solve For Global Fairness

System Visibility
- global
- local

Controller
- RR
- RS
- SS

Timescale
- microseconds
- seconds
- minutes

Weight Allocations
- WA
- WA2, WB2

Replica Selection Policies

fairness and capacity constraints
Pisces Mechanisms Solve For Global Fairness

System Visibility

- global
- local

Controller

- RR
- SS

Microseconds

Seconds

Minutes

Timescale

Maximum bottleneck flow weight exchange

Weight Allocations

Replica Selection Policies

fairness and capacity constraints

WA

PP

WA₂ W₂

FQ
Pisces Mechanisms Solve For Global Fairness

System Visibility

Controller

local

RR

RR

SS

SS

global

Microseconds

Seconds

Minutes

Timescale

Maximum bottleneck flow weight exchange

FAST-TCP based replica selection

Replica Selection Policies

Fairness and capacity constraints

Weight Allocations

WA

PP

WA A2 W B2
Pisces Mechanisms Solve For Global Fairness

- Maximum bottleneck flow weight exchange
- FAST-TCP based replica selection
- DRR token-based DRFQ scheduler
- Weight Allocations
- Fairness and capacity constraints
- Replica Selection Policies

Timescale:
- microseconds
- seconds
- minutes
Evaluation
Evaluation

• Does Pisces achieve (even) system-wide fairness?
Evaluation

• Does Pisces achieve (even) system-wide fairness?
  - Is each Pisces mechanism necessary for fairness?
Evaluation

• Does Pisces achieve (even) system-wide fairness?
  - Is each Pisces mechanism necessary for fairness?
  - What is the overhead of using Pisces?
Evaluation

• Does Pisces achieve (even) system-wide fairness?
  - Is each Pisces mechanism necessary for fairness?
  - What is the overhead of using Pisces?

• Does Pisces handle mixed workloads?
Evaluation

• Does Pisces achieve (even) system-wide fairness?
  - Is each Pisces mechanism necessary for fairness?
  - What is the overhead of using Pisces?

• Does Pisces handle mixed workloads?

• Does Pisces provide weighted system-wide fairness?
Evaluation

• Does Pisces achieve (even) system-wide fairness?
  - Is each Pisces mechanism necessary for fairness?
  - What is the overhead of using Pisces?

• Does Pisces handle mixed workloads?

• Does Pisces provide weighted system-wide fairness?

• Does Pisces provide local dominant resource fairness?
Evaluation

• Does Pisces achieve (even) system-wide fairness?
  - Is each Pisces mechanism necessary for fairness?
  - What is the overhead of using Pisces?

• Does Pisces handle mixed workloads?

• Does Pisces provide weighted system-wide fairness?

• Does Pisces provide local dominant resource fairness?

• Does Pisces handle dynamic demand?
Evaluation

• Does Pisces achieve (even) system-wide fairness?
  - Is each Pisces mechanism necessary for fairness?
  - What is the overhead of using Pisces?

• Does Pisces handle mixed workloads?

• Does Pisces provide weighted system-wide fairness?

• Does Pisces provide local dominant resource fairness?

• Does Pisces handle dynamic demand?

• Does Pisces adapt to changes in object popularity?
Evaluation

• Does Pisces achieve (even) system-wide fairness?
  - Is each Pisces mechanism necessary for fairness?
  - What is the overhead of using Pisces?

• Does Pisces handle mixed workloads?

• Does Pisces provide weighted system-wide fairness?

• Does Pisces provide local dominant resource fairness?

• Does Pisces handle dynamic demand?

• Does Pisces adapt to changes in object popularity?
Pisces Achieves System-wide Per-tenant Fairness

Ideal fair share: 110 kreq/s (1kB requests)

Unmodified Membase

GET Requests (kreq/s)

Time (s)

0.57 MMR

8 Tenants - 8 Client - 8 Storage Nodes
Zipfian object popularity distribution
Min-Max Ratio: min rate/max rate (0,1]
Pisces Achieves System-wide Per-tenant Fairness

Ideal fair share: 110 kreq/s (1kB requests)

Unmodified Membase

GET Requests (kreq/s)

Time (s)

0.57 MMR

Pisces

GET Requests (kreq/s)

Time (s)

0.98 MMR

8 Tenants - 8 Client - 8 Storage Nodes

Zipfian object popularity distribution

Min-Max Ratio: min rate/max rate (0,1]
Each Pisces Mechanism Contributes to System-wide Fairness and Isolation

Unmodified Membase

0.57 MMR
Each Pisces Mechanism Contributes to System-wide Fairness and Isolation

Unmodified Membase

2x vs 1x demand
Each Pisces Mechanism Contributes to System-wide Fairness and Isolation

Unmodified Membase

FQ

2x vs 1x demand
Each Pisces Mechanism Contributes to System-wide Fairness and Isolation

Unmodified Membase

- 0.57 MMR
- 0.59 MMR
- 0.64 MMR

2x vs 1x demand

- 0.36 MMR
- 0.58 MMR
- 0.74 MMR
Each Pisces Mechanism Contributes to System-wide Fairness and Isolation

Unmodified Membase

![Graph comparing different mechanisms over time]

- **FQ**
  - 0.57 MMR
  - 0.59 MMR
  - 0.64 MMR
  - 0.93 MMR

- **FQ** + **PP** + **WA**
  - 0.64 MMR
  - 0.96 MMR

**2x vs 1x demand**

- **FQ**
  - 0.36 MMR
  - 0.58 MMR
  - 0.74 MMR
  - 0.96 MMR
Each Pisces Mechanism Contributes to System-wide Fairness and Isolation
Each Pisces Mechanism Contributes to System-wide Fairness and Isolation

Unmodified Membase

0.57 MMR

0.59 MMR

0.64 MMR 0.93 MMR

0.90 MMR 0.98 MMR

2x vs 1x demand

0.36 MMR

0.58 MMR

0.74 MMR 0.96 MMR

0.89 MMR 0.97 MMR
Pisces Imposes Low-overhead

Aggregate System Throughput

GET Requests (kreq/s)

1kB Requests

10B Requests

Unmodified Membase

Pisces

< 5%

> 19%
Pisces Achieves System-wide Weighted Fairness

4 heavy hitters  20 moderate demand  40 low demand
Pisces Achieves System-wide Weighted Fairness

4 heavy hitters  20 moderate demand  40 low demand

GET Requests (kreq/s)
Pisces Achieves System-wide Weighted Fairness

0.98 MMR
4 heavy hitters

0.89 MMR
20 moderate demand

0.91 MMR
40 low demand
Pisces Achieves System-wide Weighted Fairness

0.98 MMR
4 heavy hitters

0.89 MMR
20 moderate demand

0.91 MMR
40 low demand

GET Requests (kreq/s)

Time (s)

100x weight (4)
10x weight (20)
1x weight (40)

0.91 MMR
Pisces Achieves System-wide Weighted Fairness

0.98 MMR
4 heavy hitters

0.89 MMR
20 moderate demand

0.91 MMR
40 low demand

GET Requests (kreq/s)

Time (s)

100x weight (4)
10x weight (20)
1x weight (40)

0.56 MMR
0.91 MMR
Pisces Achieves Dominant Resource Fairness

1kB workload
bandwidth limited

10B workload
request limited
Pisces Achieves Dominant Resource Fairness

1kB workload
bandwidth limited

1kB bandwidth limited

10B bandwidth limited

10B workload
request limited

10B request limited

Pisces Achieves Dominant Resource Fairness
Pisces Achieves Dominant Resource Fairness

1kB \textit{workload} \\
bandwidth limited

10B \textit{workload} \\
request limited
Pisces Achieves Dominant Resource Fairness

1kB workload
bandwidth limited

1kB bandwidth limited

GET Requests (kreq/s)

76% of bandwidth

76% of request rate

1kB workload
request limited

10B bandwidth limited

10B request limited
Pisces Achieves Dominant Resource Fairness

1kB workload
bandwidth limited

1kB bandwidth limited

76% of bandwidth

GET Requests (kreq/s)

1kB bandwidth limited
10B request limited

10B workload
request limited

76% of request rate

24% of request rate

Time (s)

Bandwidth (Mb/s)

Time (s)

GET Requests (kreq/s)
Pisces Adapts to Dynamic Demand

Tenant Demand

- Constant
- Diurnal (2x wt)
- Bursty
Pisces Adapts to Dynamic Demand

Tenant Demand:
- Constant
- Diurnal (2x wt)
- Bursty

Graph showing GET Requests (kreq/s) over Time (s):
- Constant demand with steady GET Requests.
- Diurnal demand with a peak at midday.
- Bursty demand with intermittent spikes.

The graph illustrates how Pisces adapts to different types of demand.
Pisces Adapts to Dynamic Demand

Tenant Demand

- Constant
- Diurnal (2x wt)
- Bursty

GET Requests (kreq/s) vs. Time (s)

- Constant: Linear increase
- Diurnal: Sigmoidal increase followed by decrease
- Bursty: Stepped increase and decrease

Approximate 2x increase in GET requests.
Pisces Adapts to Dynamic Demand

- Constant
- Diurnal (2x wt)
- Bursty

GET Requests (kreq/s)

Time (s)

Tenant Demand

Demand

even

~2x
Conclusion

● Pisces Contributions
  - Per-tenant weighted max-min fair shares of system-wide resources w/ high utilization
  - Arbitrary object distributions
  - Different resource bottlenecks
  - Novel decomposition into 4 complementary mechanisms

PP Partition Placement  WA Weight Allocation  RS Replica Selection  FQ Fair Queuing