Transaction-Aware SSD Cache Allocation for the Virtualization Environment

Zhen Tang, Institute of Software, Chinese Academy of Sciences

Heng Wu, Institute of Software, Chinese Academy of Sciences

Lei Sun, Tianjin Massive Data Processing Technology Laboratory

Zhongshan Ren, Institute of Software, Chinese Academy of Sciences

Wei Wang, Institute of Software, Chinese Academy of Sciences

Wei Zhou, KSYUN

Liang Yang, KSYUN

March 26, 2018

ISCAS 中国科学院软件研究所 Institute of Software Chinese Academy of Sciences

Transactional Applications in the JointCloud Environment

- Web-based applications rely on multiple services from different cloud providers
- Rely on different types of storage service; access through the Internet
  - VM: Aliyun
  - Images, Audios, Videos: Qiniu
  - Relational database: UDB from UCloud
- In this scenario, host-side SSD caching may be a preferred solution

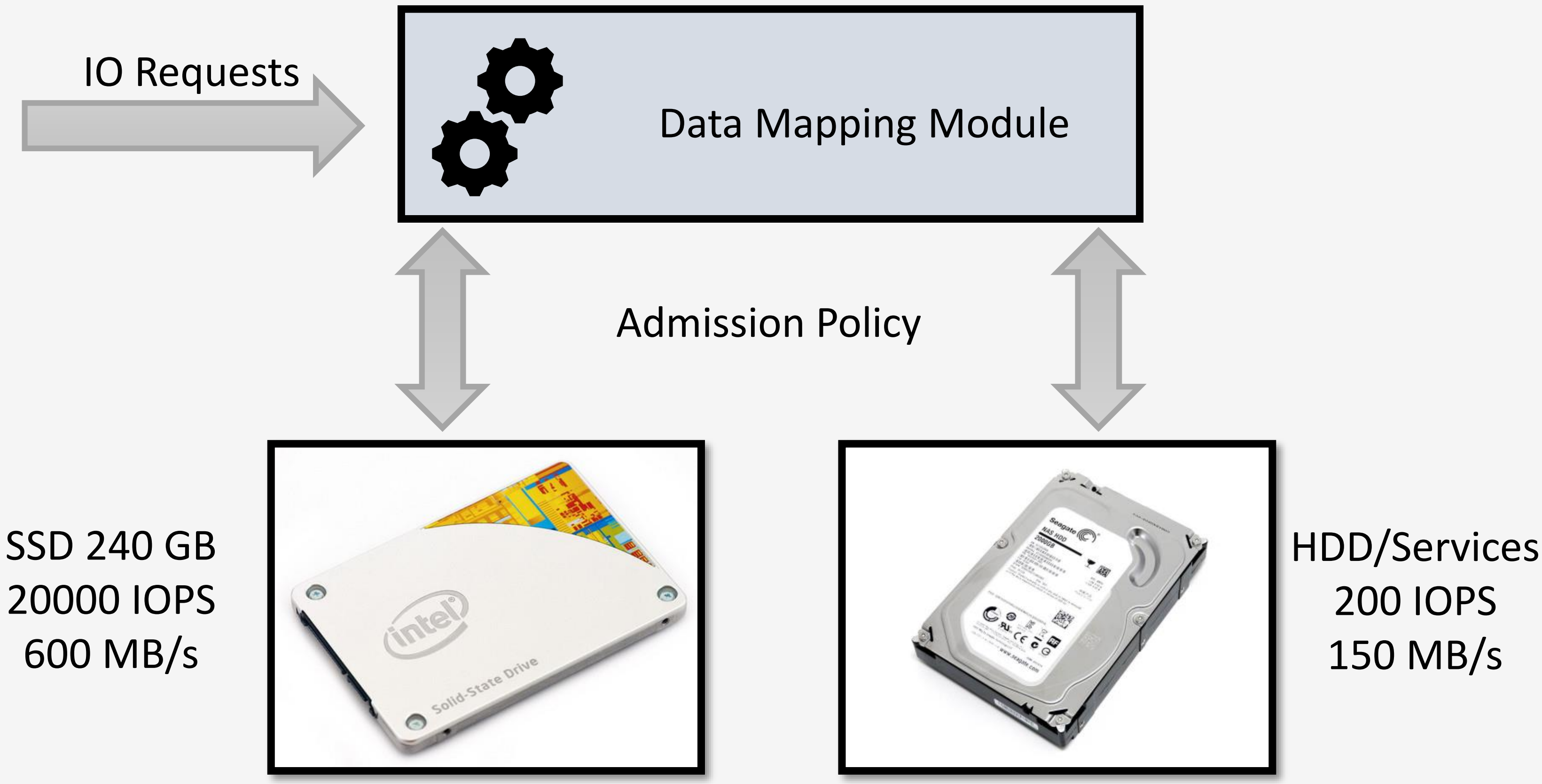
The Host-side SSD Caching System

- Higher Performance, especially in random access
- Expensive in Per GB Capacity

- Large Capacity
- Lower random IOPS (I/O Operations Per Second)

The Host-side SSD Caching System is a balance between **Cost** and **Performance**

The Host-side SSD Caching System



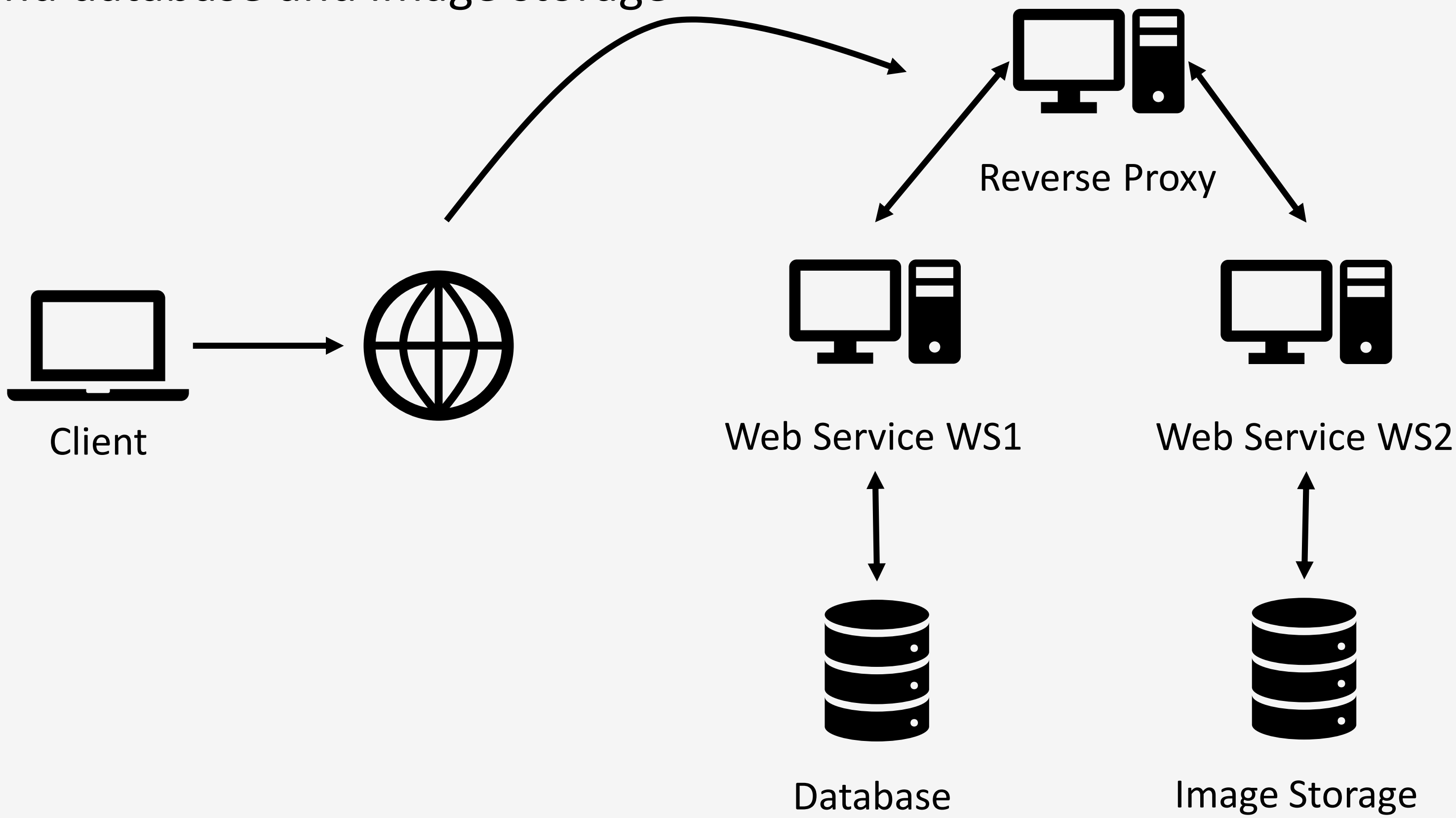
The Data Mapping Module is the Fundamental Part

Application-level Performance

- The transactional application consists of multiple virtual machines (VMs)
  - VMs inside the application face the similar workload
- The critical performance metric from the application view is the **latency**
  - Related to the IO performance of all VMs inside the application

Example

- An online book store consists of two VMs on one hypervisor, with back end database and image storage

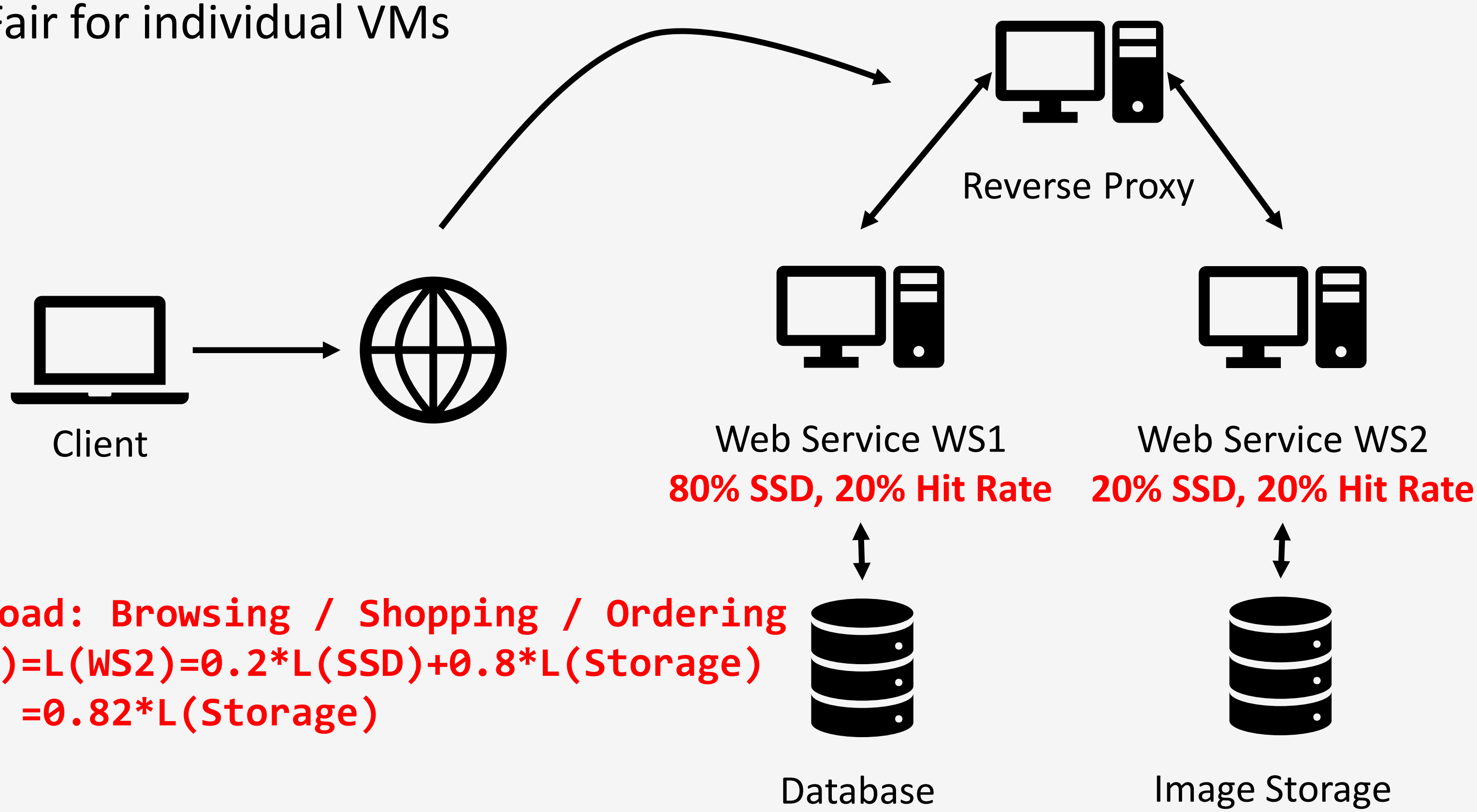


Types of workloads

- Three types of workloads
  - Browsing: Heavy read operations on images
  - Shopping: Read operations on images, write operations on database
  - Ordering: Heavy write operations on database
- The latency L is divided into 2 parts: on WS1 and on WS2
  - Browsing: L(Browsing)=0.3*L(WS1)+0.7*L(WS2)
  - Shopping: L(Shopping)=0.5*L(WS1)+0.5*L(WS2)
  - Ordering: L(Ordering)=0.7*L(WS1)+0.3*L(WS2)
- The working set for WS1 is **8GB**, for WS2, **2GB**. The total cache space is **2GB**
- Assuming that **L(SSD)=0.1*L(Storage)**

Working Set Based Cache Scheme

- Allocate SSD according to the working set
- Fair for individual VMs



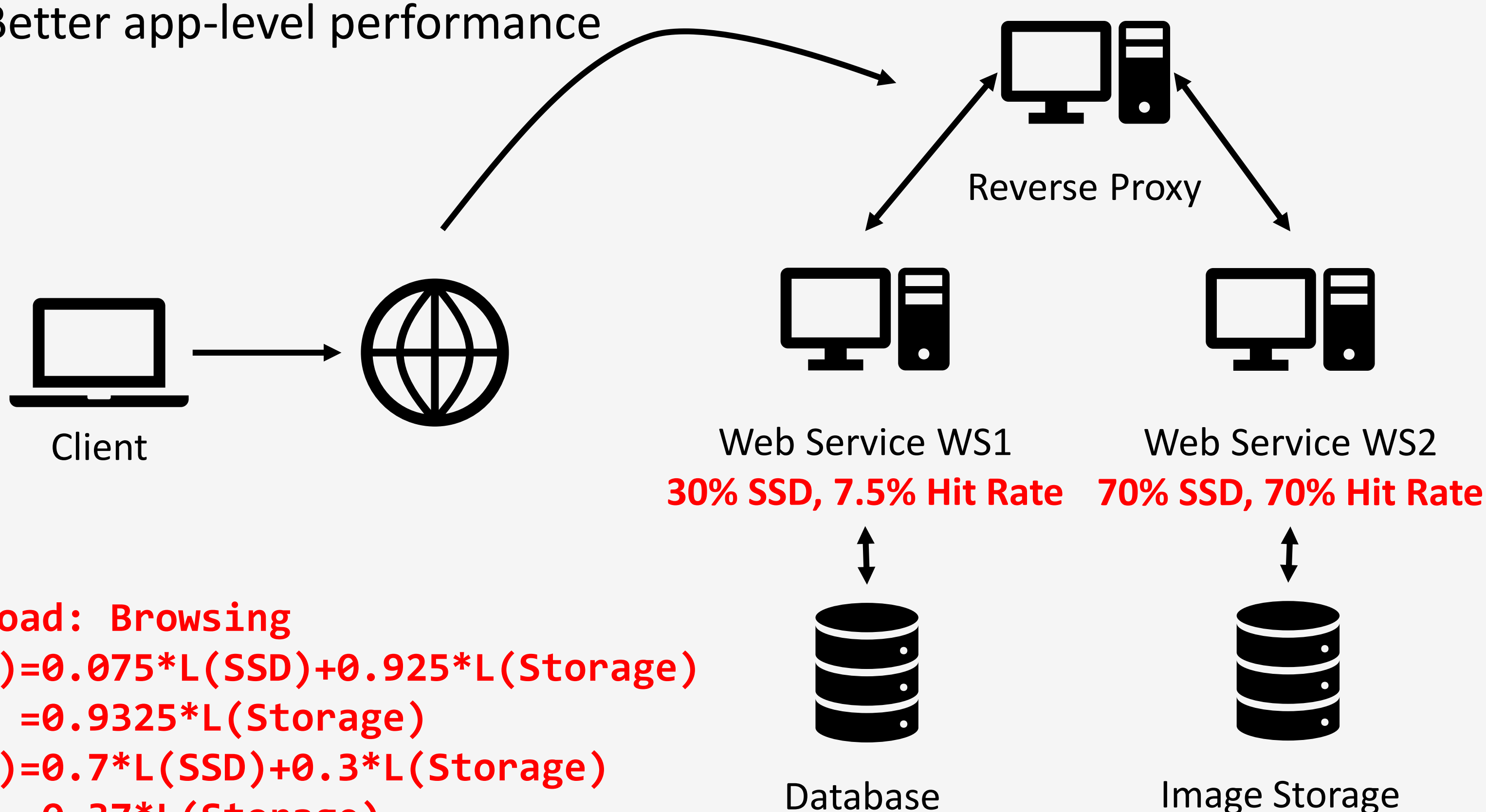
Workload: Browsing / Shopping / Ordering

L(WS1)=L(WS2)=0.2*L(SSD)+0.8*L(Storage)

=0.82*L(Storage)

Transaction-aware Cache Scheme

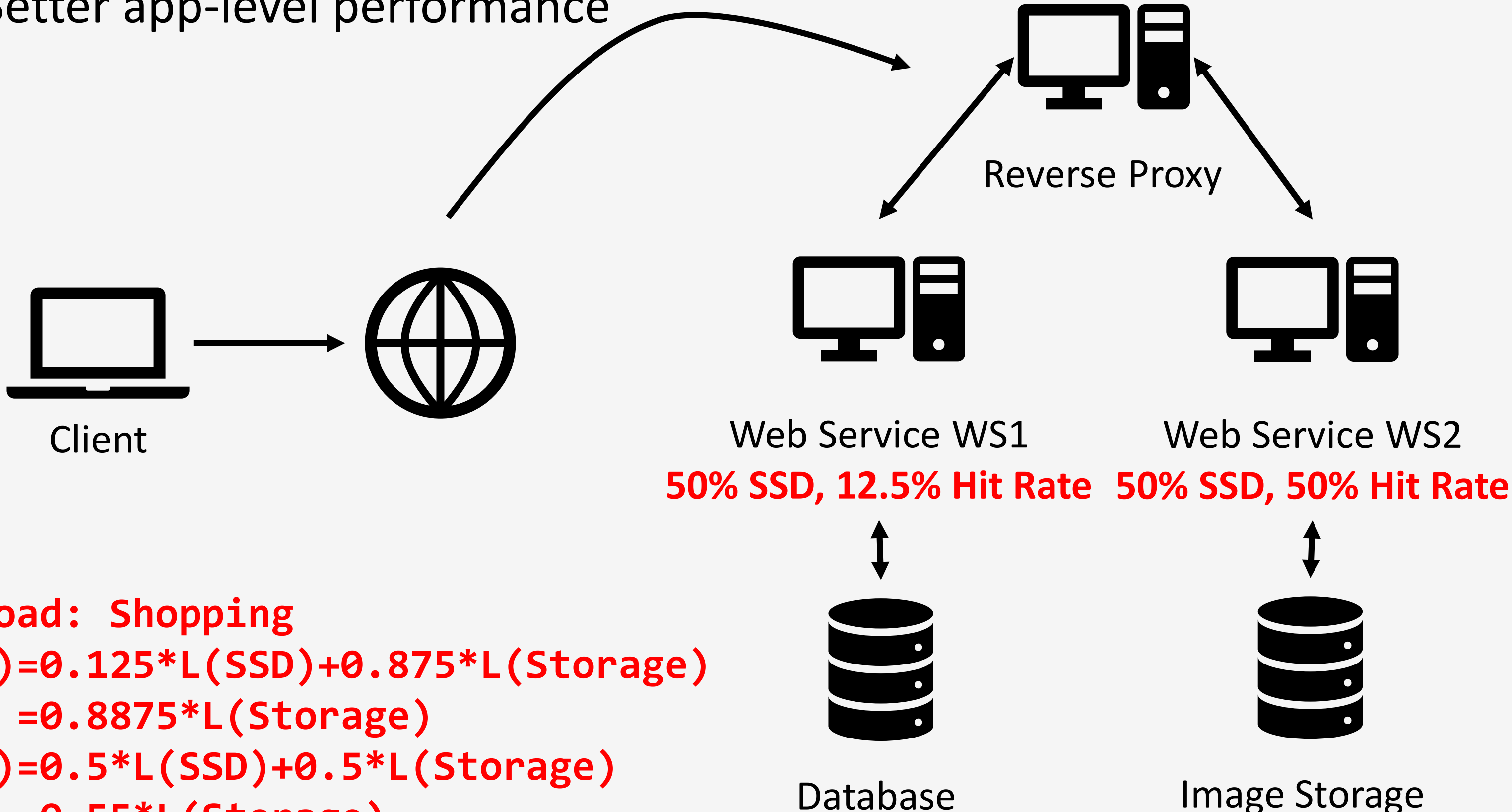
- Allocate SSD cache according to latency distribution for workloads
- Better app-level performance




```
Workload: Browsing
L(WS1)=0.075*L(SSD)+0.925*L(Storage)
      =0.9325*L(Storage)
L(WS2)=0.7*L(SSD)+0.3*L(Storage)
      =0.37*L(Storage)
```

Transaction-aware Cache Scheme

- Allocate SSD cache according to latency distribution for workloads
- Better app-level performance



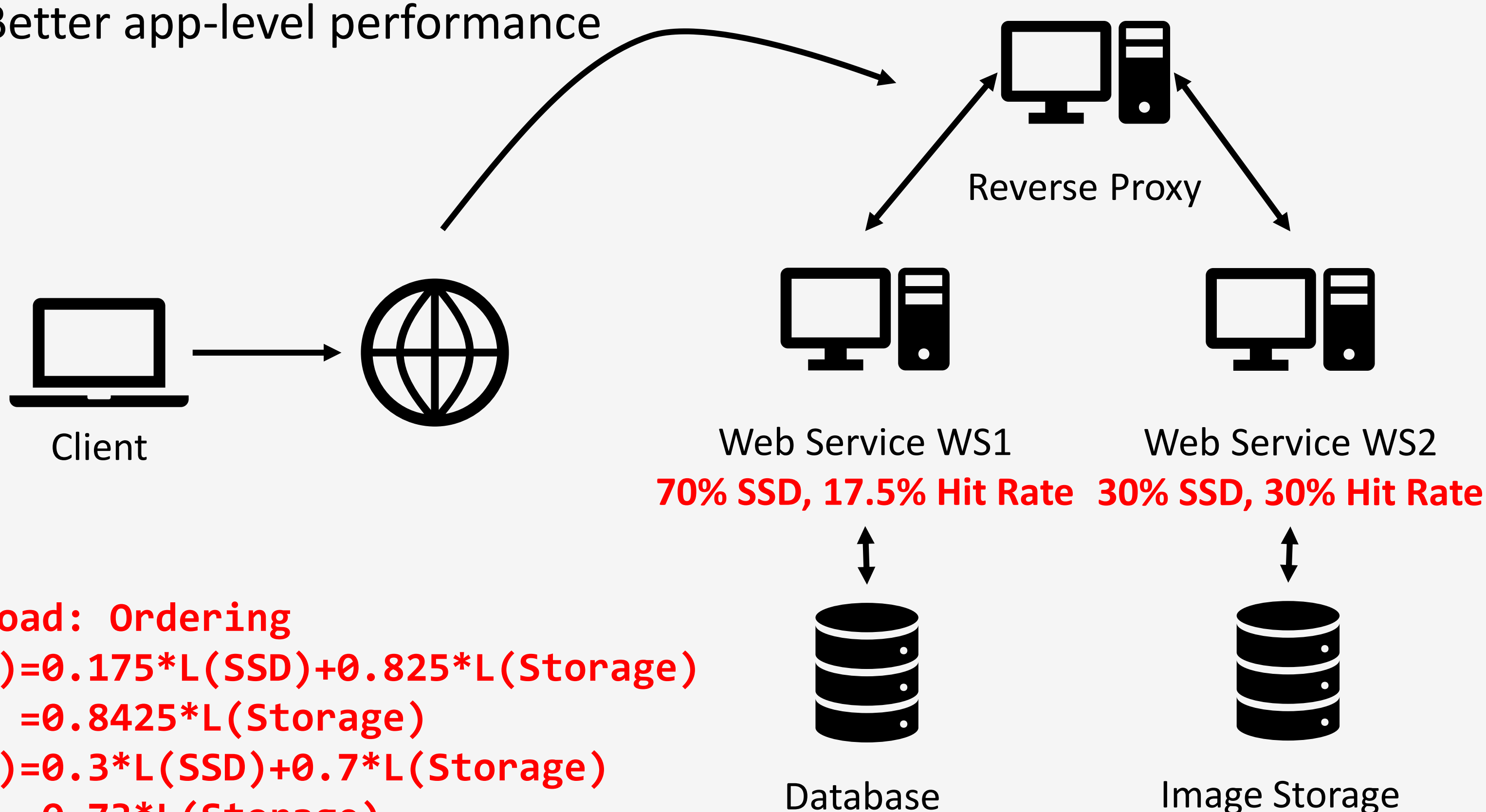
Workload: Shopping

L(WS1)=0.125*L(SSD)+0.875*L(Storage)
=0.8875*L(Storage)

L(WS2)=0.5*L(SSD)+0.5*L(Storage)
=0.55*L(Storage)

Transaction-aware Cache Scheme

- Allocate SSD cache according to latency distribution for workloads
- Better app-level performance




```
Workload: Ordering
L(WS1)=0.175*L(SSD)+0.825*L(Storage)
      =0.8425*L(Storage)
L(WS2)=0.3*L(SSD)+0.7*L(Storage)
      =0.73*L(Storage)
```

Performance

- Latency for working set based cache scheme
  - Browsing: L=0.3*L(WS1)+0.7*L(WS2)=0.82*L(Storage)
  - Shopping: L=0.5*L(WS1)+0.5*L(WS2)=0.82*L(Storage)
  - Ordering: L=0.7*L(WS1)+0.3*L(WS2)=0.82*L(Storage)
- Latency for transaction aware cache scheme
  - Browsing: L=0.3*L(WS1)+0.7*L(WS2)=0.53875*L(Storage)
  - Shopping: L=0.5*L(WS1)+0.5*L(WS2)=0.71875*L(Storage)
  - Ordering: L=0.7*L(WS1)+0.3*L(WS2)=0.80875*L(Storage)
- Transaction-aware cache scheme will lead to better application-level latency

Challenges

- How to create the connection between the latency from the app-level and the IO performance from the VM-level?
- How to detect the type of workloads and trigger the cache adjustment? How to react to changing workloads?

Transaction-aware SSD Cache Allocation

- How to create the connection between the latency from the app-level and the IO performance from the VM-level?
  - Application: A set of **VMs**
  - Workload: Mix of **transactions**
  - The average latency: Weighted sum of the **latency** of transactions
  - Latency of transactions: The sum of **time consumed on VMs**
  - The latency of VMs: **Can be reduced by SSD cache**
- Thus, we use the **latency distribution** as the app-level metric

Closed Loop Adaptation

- How to detect the type of workloads and trigger cache adjustment? How to react to the changing workloads?
  - We introduce closed loop adaptation inspired by MAPE-K
  - **Monitor**: Monitor the low-level IO performance and the status of transactional applications
  - **Analyze**: Detect the workload type; Calculate the latency distribution
  - **Plan**: GA based approach to calculate the nearly optimal weights for VMs
  - **Execute**: Allocate the SSD cache

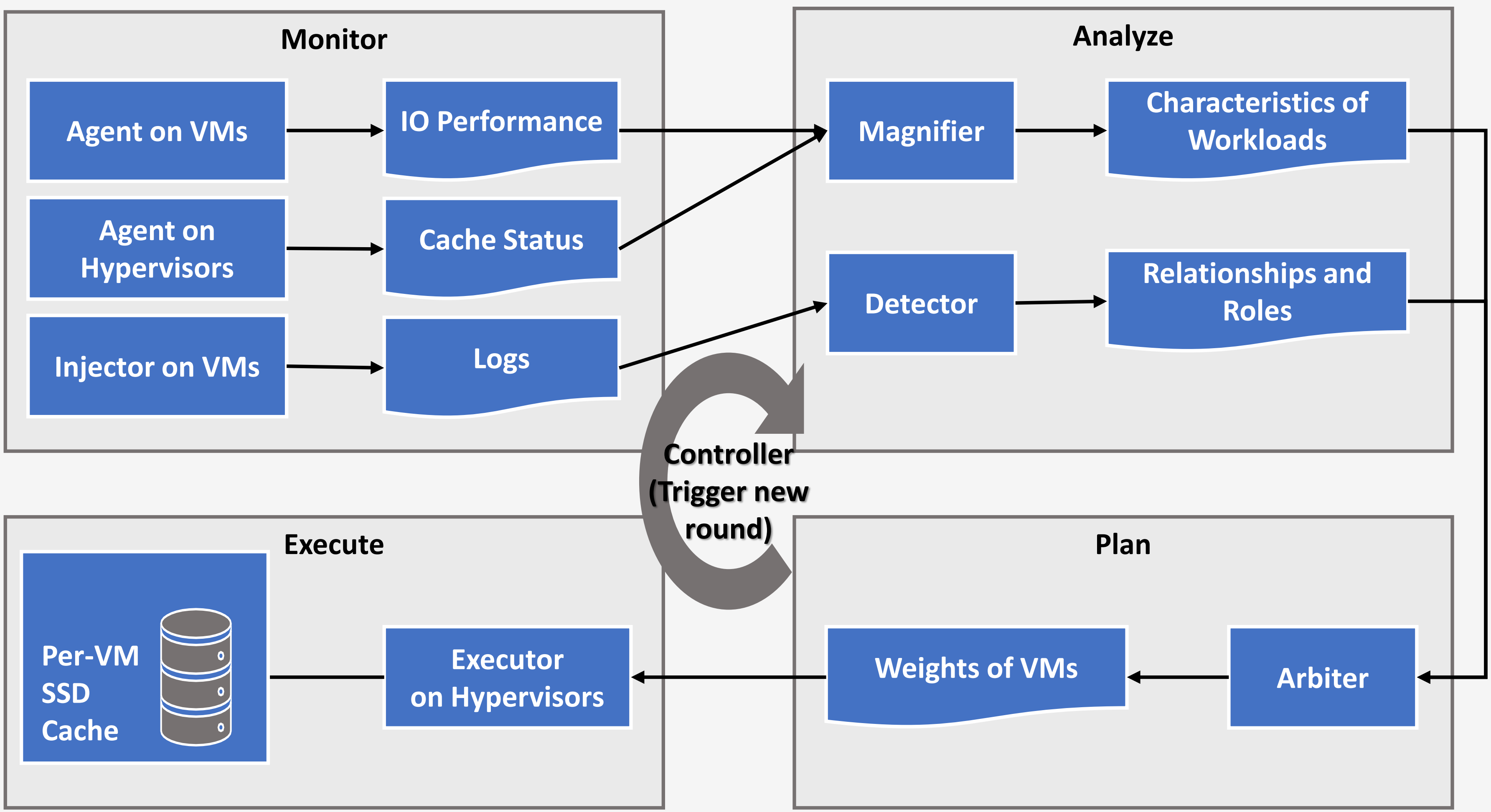
GA based approach

- Goal: Find the weights to minimize the Latency
- Structure
  - Chromosome: The weight of a specific VM
  - Genome: The SSD cache allocation plan
  - Fitness: Create connection between low-level IO performance and high-level latency by using latency distribution
  - The selection, crossover, and mutation operations: Like the general GAs

Fitness Calculation

- Calculate from three metrics:
  - **VM intensity** (App-level): calculated from the latency distribution
  - **IO ratio** (Low-level IO): the ratio of IO time and non-IO time of CPU
  - **Random access intensity** (Characteristics of SSD): Calculated from the average IO request size
- For a given genome, use the **Euclidean metric** to represent the match degree of the weights and the normalized three metrics

Implementation

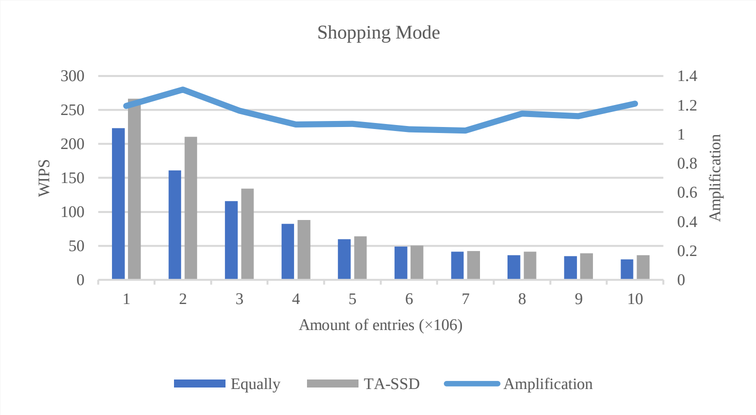


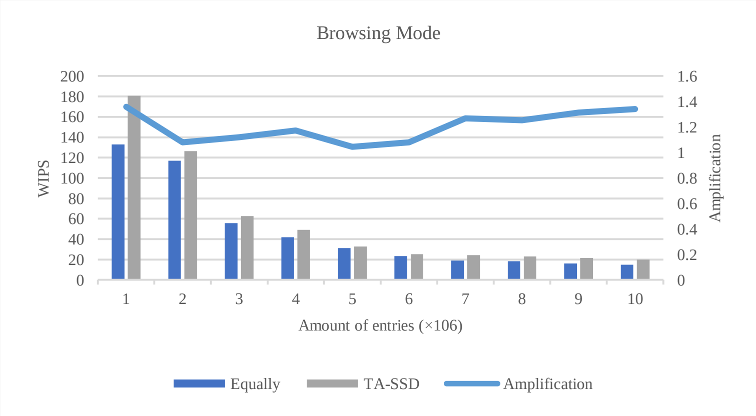
Experiment Setup

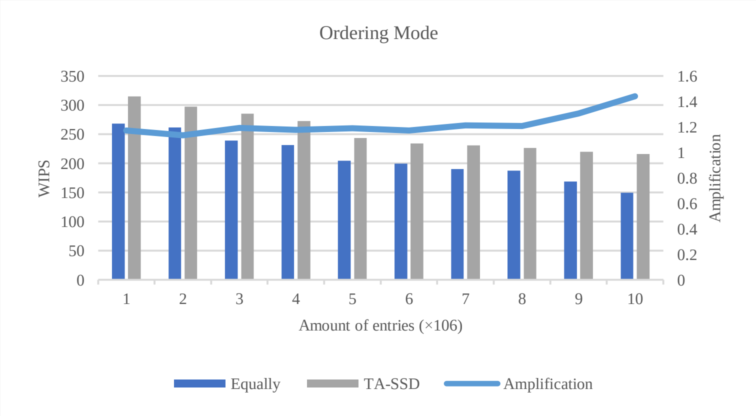
- Comparing to the equally partitioned cache
- Benchmark
  - TPC-W, an e-commerce benchmark (online book store)
  - Bench4Q, a TPC-W based load testing tool
  - Three modes: Browsing, Shopping, Ordering
- Environment
  - SSD and HDD
  - 3 applications placed on 2 hypervisors, each consists of 2 VMs
  - One web server and one database server
  - Use 3 VM to generate workload
  - 768MB SSD cache for each hypervisor (256MB for each VM for equally partitioned cache)

Performance Focusing on One Application

- WIPS (Web Interactions Per Second) of 3 modes (Browsing, Shopping, Ordering)
  - 100 vUsers
  - Data scales up from 100,000 to 1,000,000 entries
  - WIPS is improved by up to 40%



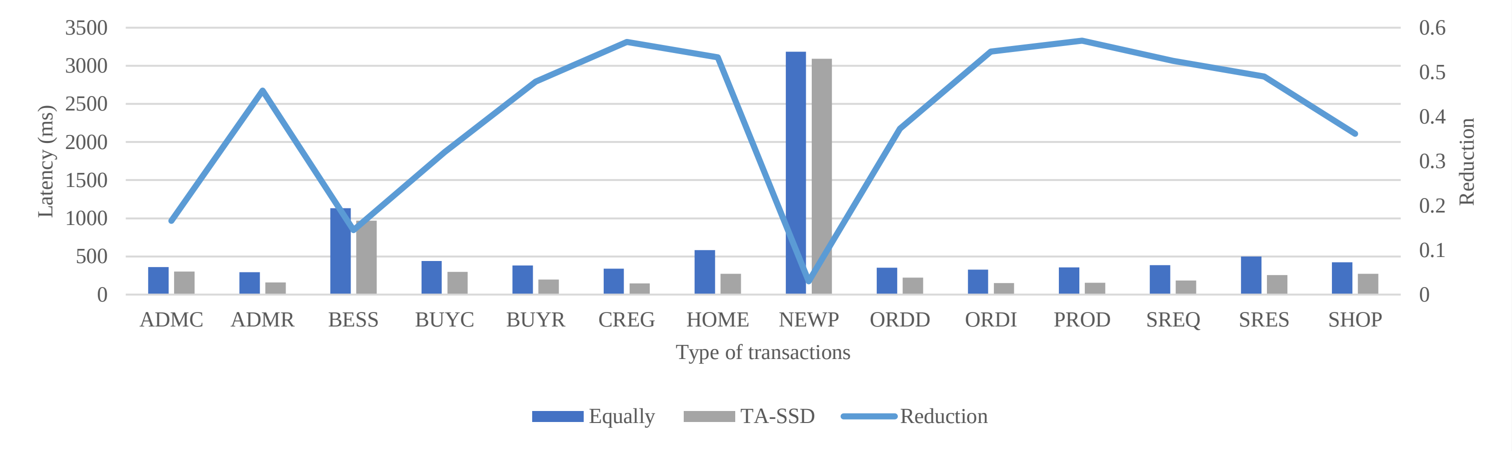




Latency of different types of transactions

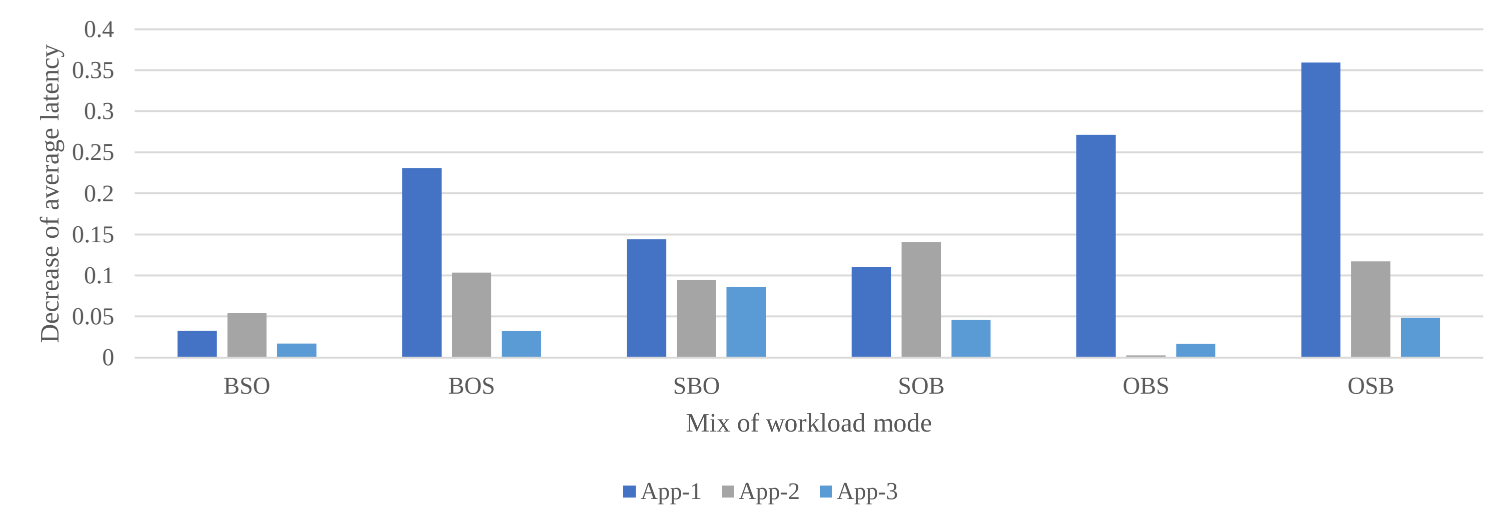
- Data scale: 1,000,000 entries
- Up to 50% decrease on latency
  - However, not efficient when facing transactions which may trigger full table scan (BESS, NEWP, SRES)

Abbreviation	Description	Dominant IO
ADMC	Admin Confirm	(No Dominant IO)
ADMR	Admin Request	Random read
BESS	Best Seller	Sequential read
BUYC	Buy Confirm	Random write
BUYR	Buy Request	Random read
CREG	Registration	(No Dominant IO)
HOME	Home	Random read
NEWP	New Products	Sequential read
ORDD	Order Display	Random read
ORDI	Order Inquiry	(No Dominant IO)
PROD	Product Detail	Random read
SREQ	Search Request	(No Dominant IO)
SRES	Search Result	Sequential Read
SHOP	Shopping Cart	Random write



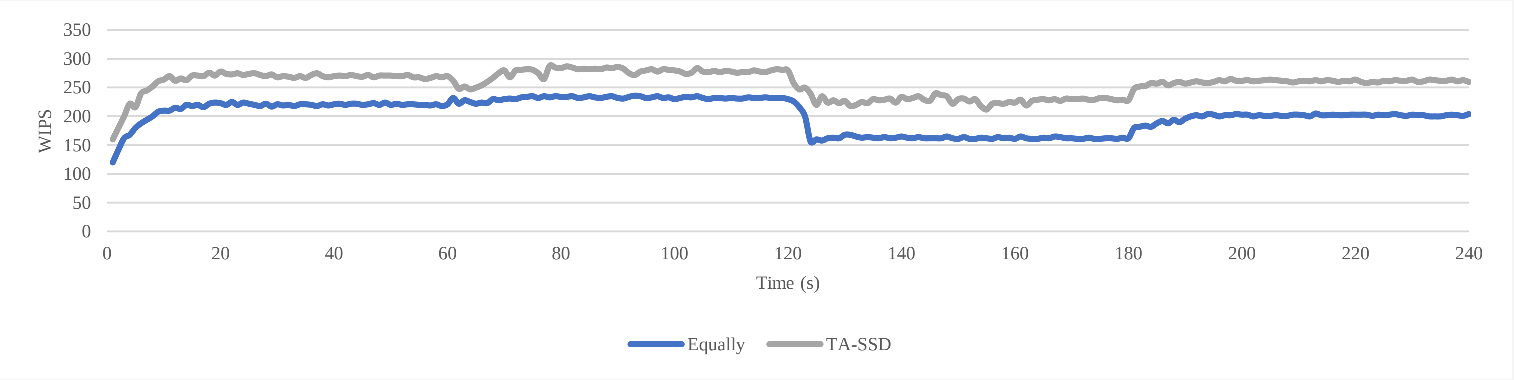
Performance Among all Applications

- 3 Applications, mixed of 3 modes
  - 300,000; 500,000 and 700,000 entries for 3 applications
  - Latency decreased by up to 45%, and by 20% in average

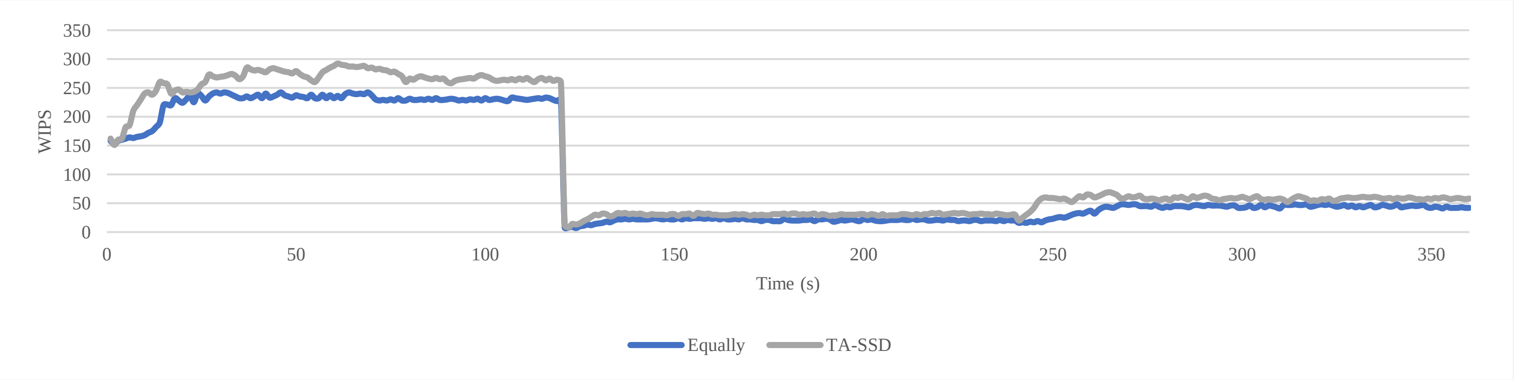


Self Adaptation

- Change of base load



- Change of workload type



Discussions

- TA-SSD uses the HDD and SSD as the example, but can be applied to the joint cloud environment
- TA-SSD can also be applied to other types of applications rely on multiple storage services in the joint cloud environment
- AC-SSD [Internetware'17] aims to control the capacity of both cache space and IOPS to reduce the job completion time of elastic Hadoop applications

Conclusion

- We present TA-SSD
  - Use application-level metrics to guide the SSD cache allocation
  - Use genetic algorithm based approach to calculate the weights for VMs
  - Introduce the closed loop adaptation to react to changing workloads
  - Improve the performance of transactional applications

Thanks

Zhen Tang

tangzhen12@otcaix.iscas.ac.cn