Enabling Live Migration of Containerized Applications Across Clouds

IEEE InfoCom '20

Thad Benjaponpitak, Meatasit Karakate, Kunwadee Sripanidkulchai

Chulalongkorn University

JaeHyun Lee (jhlee2021@mmlab.snu.ac.kr)

Contents

- Background
  - Container based
  - Cloud Provider and Live Migration

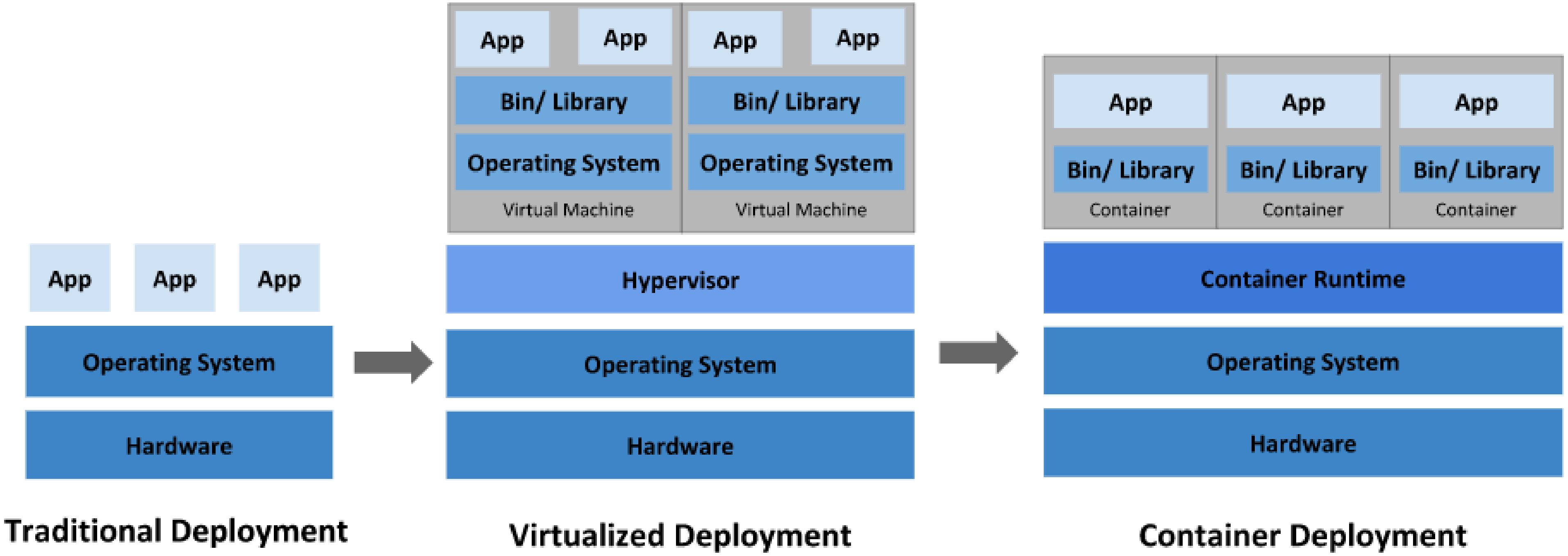
- System design
  - Components
  - Design Goals
  - Migration Flow

- Evaluation and Result

- Conclusion

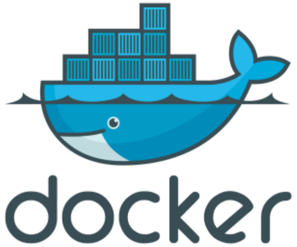
Container

- **Containers** are becoming the de facto standard
  - Application focused solution
  - Abstraction of system call and resources
  - But isolated from the host machine

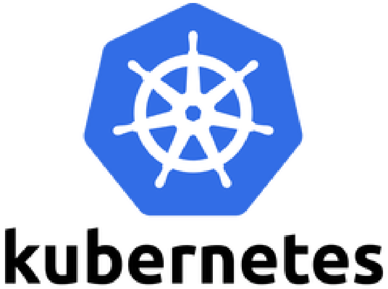


Docker and Kubernetes (common sense)

- Docker
  - OS-level virtualization to deliver software in packages called containers
  - De facto standard of **container**



- Kubernetes
  - Container orchestration system for automating deployment, scaling and management
  - De facto standard of **docker-based web service**

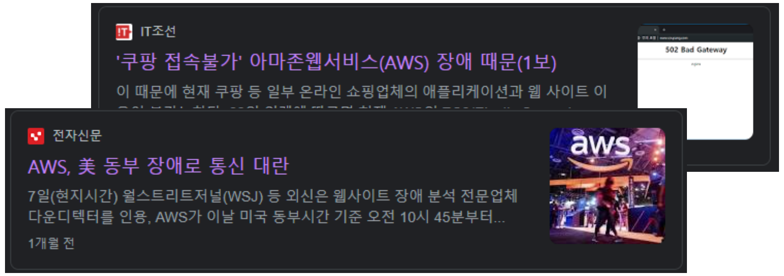


Cloud Service (common sense)

- Cloud Service Provider
  - 3 major commercial providers: Amazon / Google / MS
  - **Their Container Services**
    - AWS: EKS (Elastic Kubernetes Service)
    - GCP: GKE (Google Kubernetes Engine)
    - Azure: AKS (Azure Kubernetes Service)

Migration (common sense)

- Why?
  - In case of accident (reliability)
  - Better functionality
  - Business issue ($$$)



IT조선

'쿠팡 접속불가' 아마존웹서비스(AWS) 장애 때문(1보)

이 때문에 현재 쿠팡 등 일부 온라인 쇼핑업체의 애플리케이션과 웹 사이트 이

502 Bad Gateway

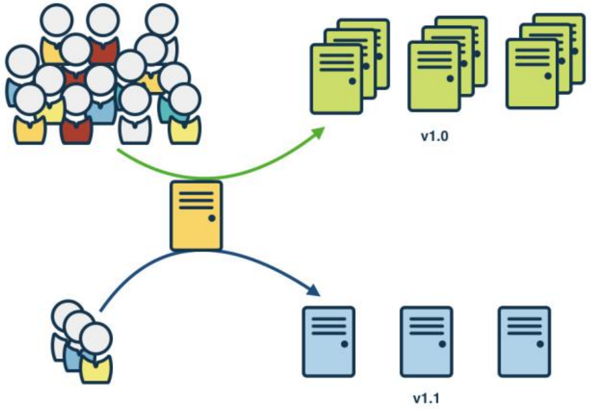
전자신문

AWS, 美 동부 장애로 통신 대란

7일(현지시간) 월스트리트저널(WSJ) 등 외신은 웹사이트 장애 분석 전문업체 다운디텍터를 인용, AWS가 이날 미국 동부시간 기준 오전 10시 45분부터...

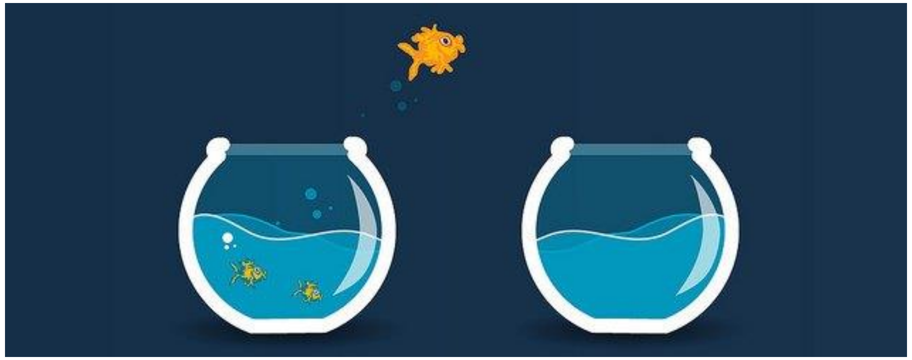
1개월 전

- How?
  - Service down (*"Service Under Maintenance"*)
  - Blue-green deployment
  - Canary deployment



Live migration

- Process of transferring the state of running application to remote with minimum downtime
  - Memory migration
  - Storage migration
  - Network migration



System Design

- Components and terms

- Design Goals

- Migration Flow

Components (or Tools/Envs)

- **StrongSwan**
  - Open source **IPSec** VPN
  - All hosts and containers can communicate using private IP addresses

- **HAProxy**
  - **handle incoming** connections at the host to support multiple container networking

- **CRIU**
  - One of the most developed implementations of **user-space-based migration** (cf. kernel-based)

- **rsync**
  - Efficient file transferring(sync) across networked computers by using **delta-copy**

- **Holding Application**
  - Extra application for holding and redirecting

- **WordPress / MySQL**
  - Sample application on this paper

Design Goals

A. Multi-cloud support
- AWS ↔ GCP ↔ Azure

B. Interdependent container support
- Web App + DB App (+ others)

C. Short migration time
- Optimization

D. Secure data transfer
- Between Source and Target cloud provider

E. No failed client connection during migration
- Bluegreen migration

F. Automated migration
- Using python based script Ansible, an open-source SW provisioning, conf. management and deployment tool

Design – key point (1/2)

C. Short migration time

✓ Memory and Storage Migration steps (typical way vs. this paper)

downtime

I. Checkpoint – freezing and dump

II. Transfer – transfer mem and storage ⛁ ⛁ ⛁ ⛁

III. Restore

<Three basic steps>

pre-migration

I. **Pre-dump** – pre-dump (CRIU)

II. **Pre-copy** – transfer pre-dump (rsync) ⛁ ⛁ ⛁

downtime

III. Checkpoint – iterative dump (CRIU)

IV. Transfer – transfer **only diffs** (rsync) ⛁

V. Restore

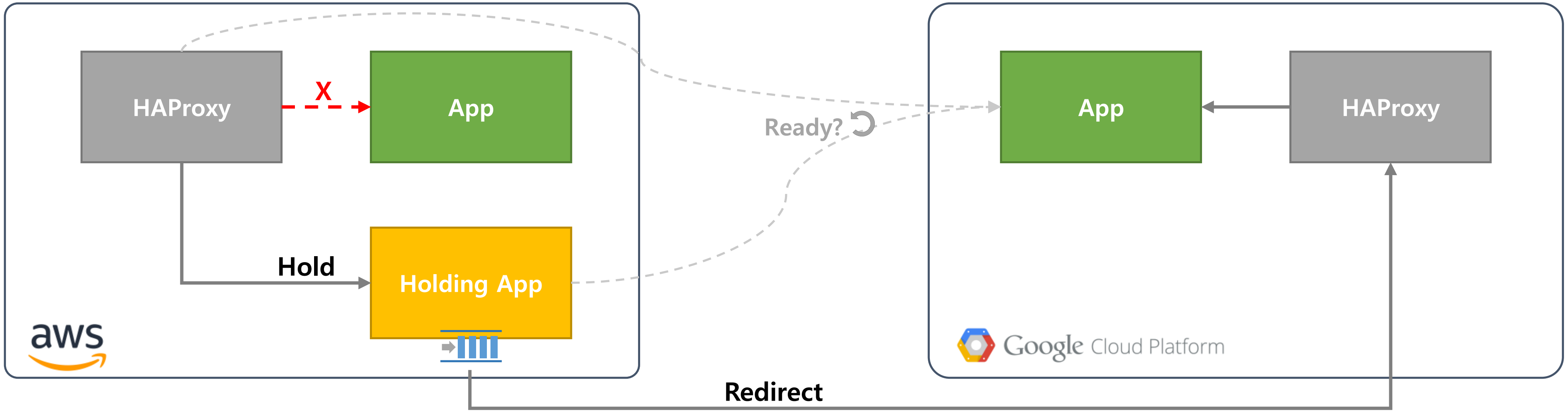
<Add pre-migration steps>

Further consideration: NFS (Network File System)
NFS also had been considered as another optimization
But, there was significant performance overhead

Design – key point (2/2)

E. No failed client connection during migration

✓ Key Idea: **Hold** and **Redirect**



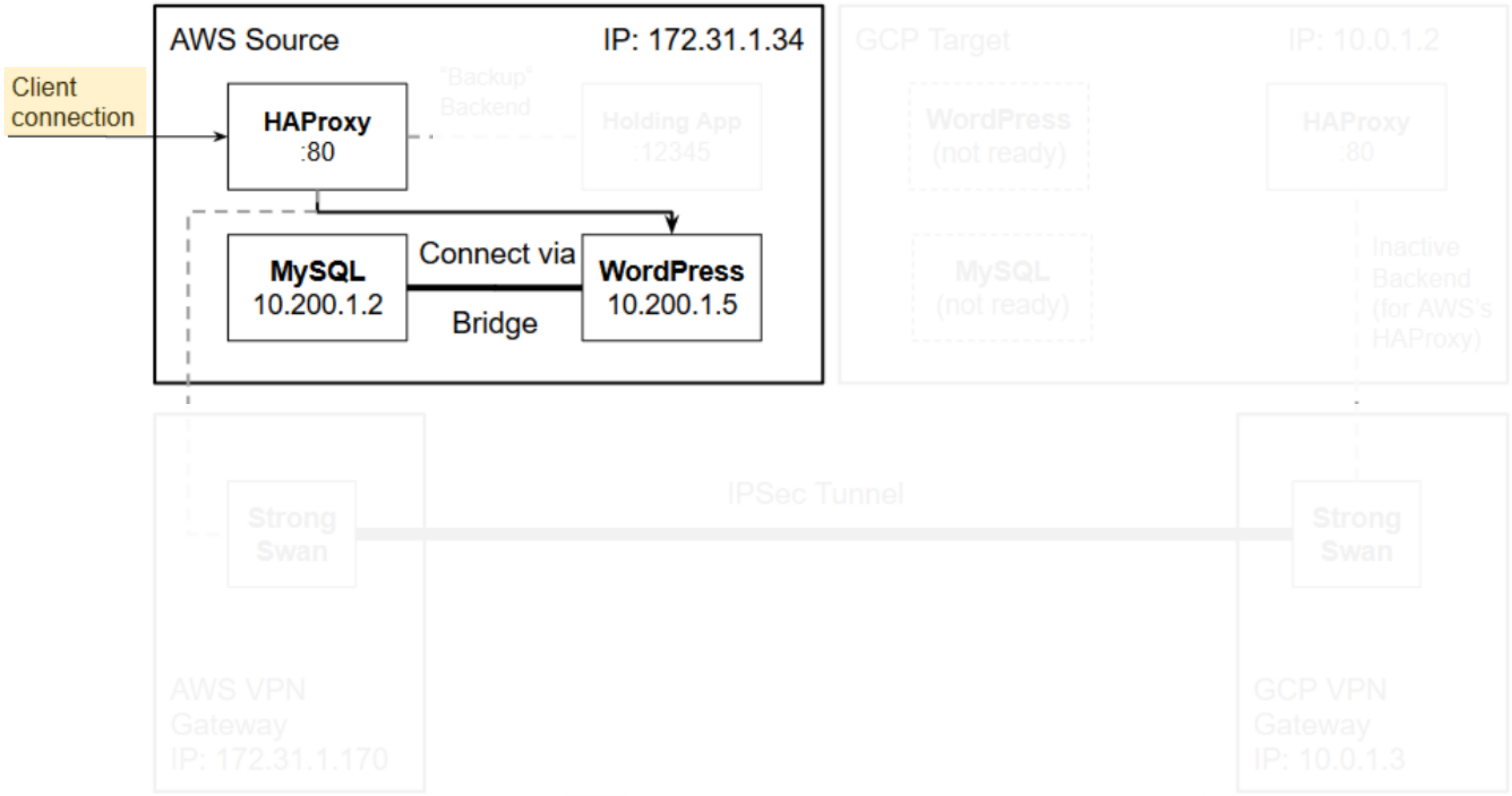
<Source Cloud> <Target Cloud>

Migration Full-Flow

- Preparation
  - Create target resources (via docker image)
  - Deploy holding app
  - Setup IPSec Tunnel
- Pre-Migration
  - Pre-dump
  - Pre-copy
- Migration
  - Checkpoint
  - Transfer
  - Restore
- Finished
  - Destroy source resources

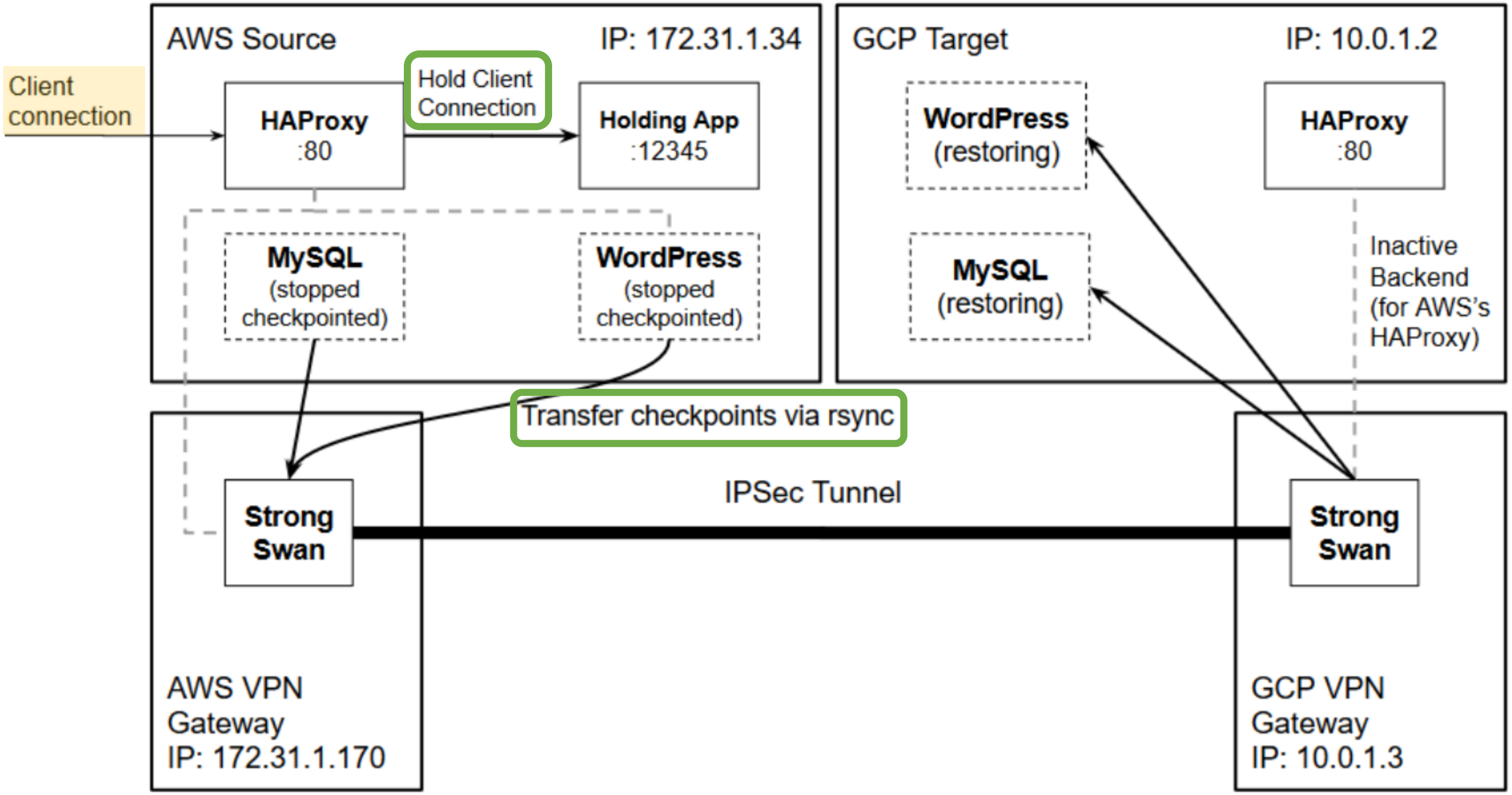
Migration Flow (1/4)

- Preparation
- Pre-migration



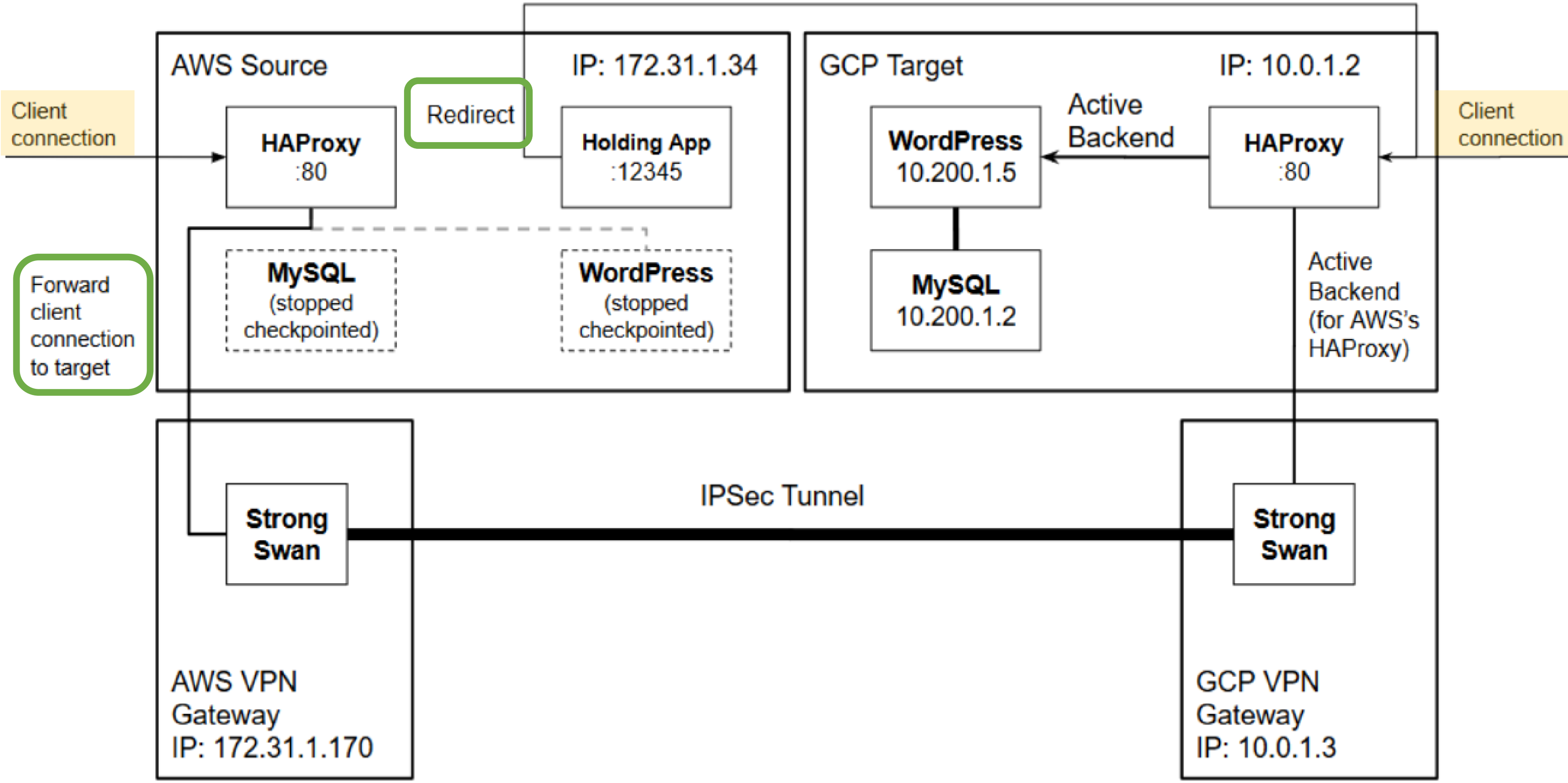
Migration Flow (2/4)

- Migration



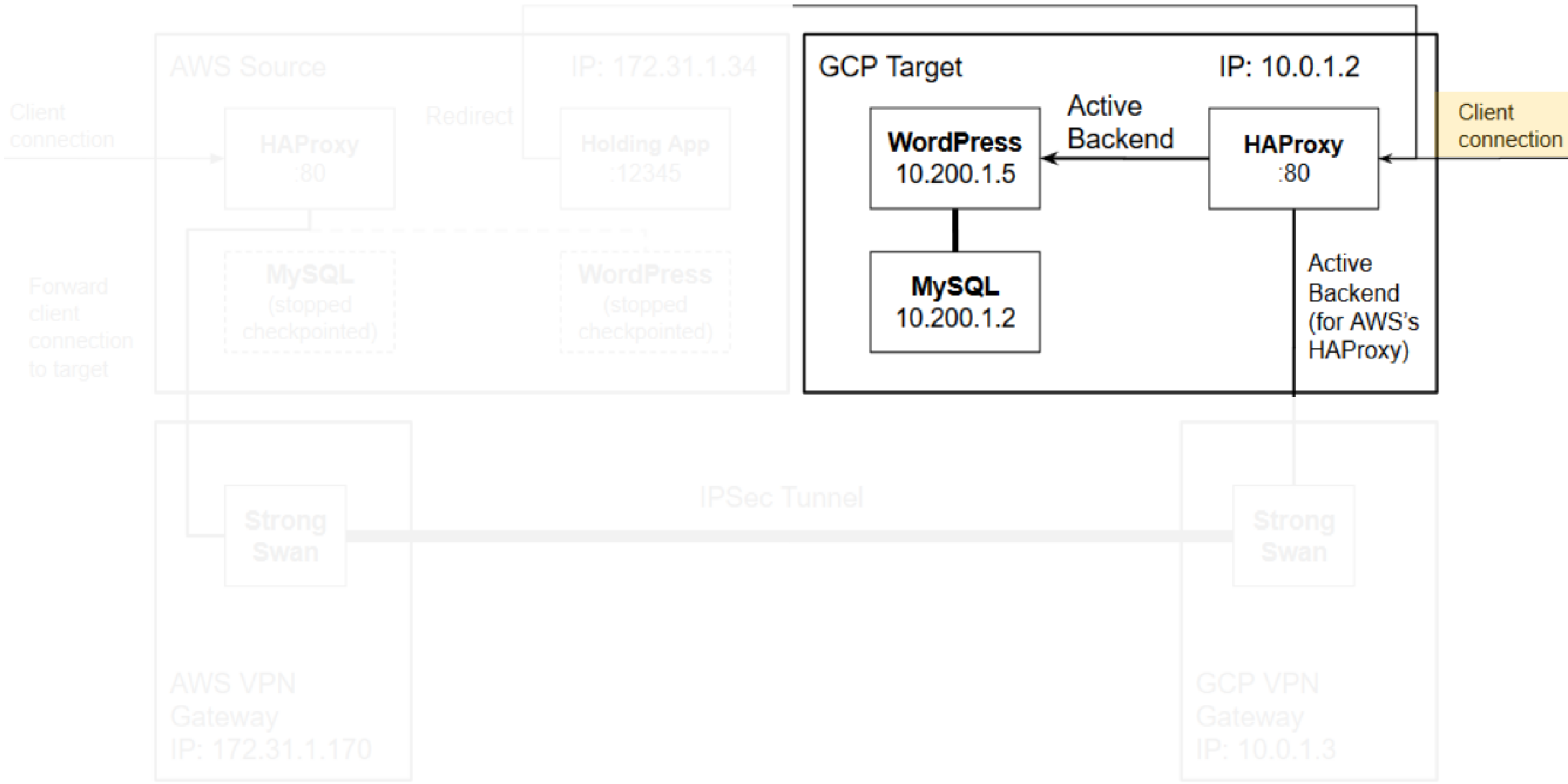
Migration Flow (3/4)

- Done



Migration Flow (4/4)

- Cleaning up



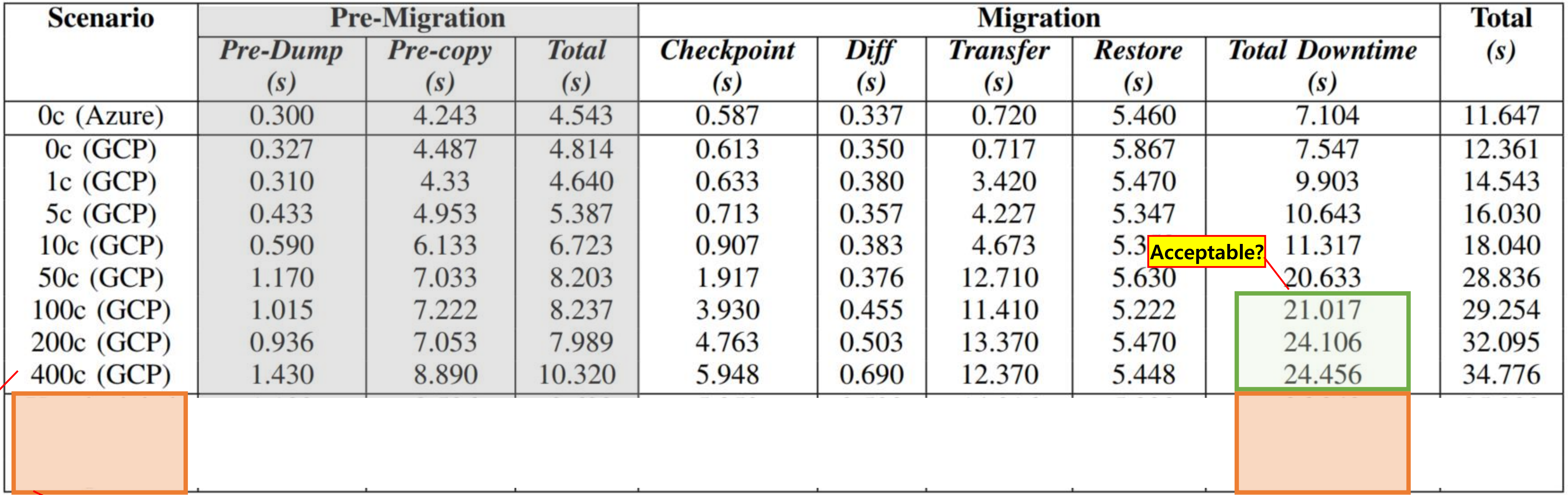
Evaluation

- Experiment specification
  - An application consisting of **2 containers**, *'WordPress'* and *'MySQL'*
  - **Live-migration** from ***AWS*** *(Amazon Web Service)* to ***GCP*** *(Google Cloud Platform)*
  - Generate **random load** (60~70 TPS) using *'Siege'*
  - Define **10 scenarios** based on the number of **concurrent clients**
  - Repeat **10 times** for each

- Constraint
  - Some outliers; perhaps due to dynamic conditions in the cloud

Result – Time Spent

don't care



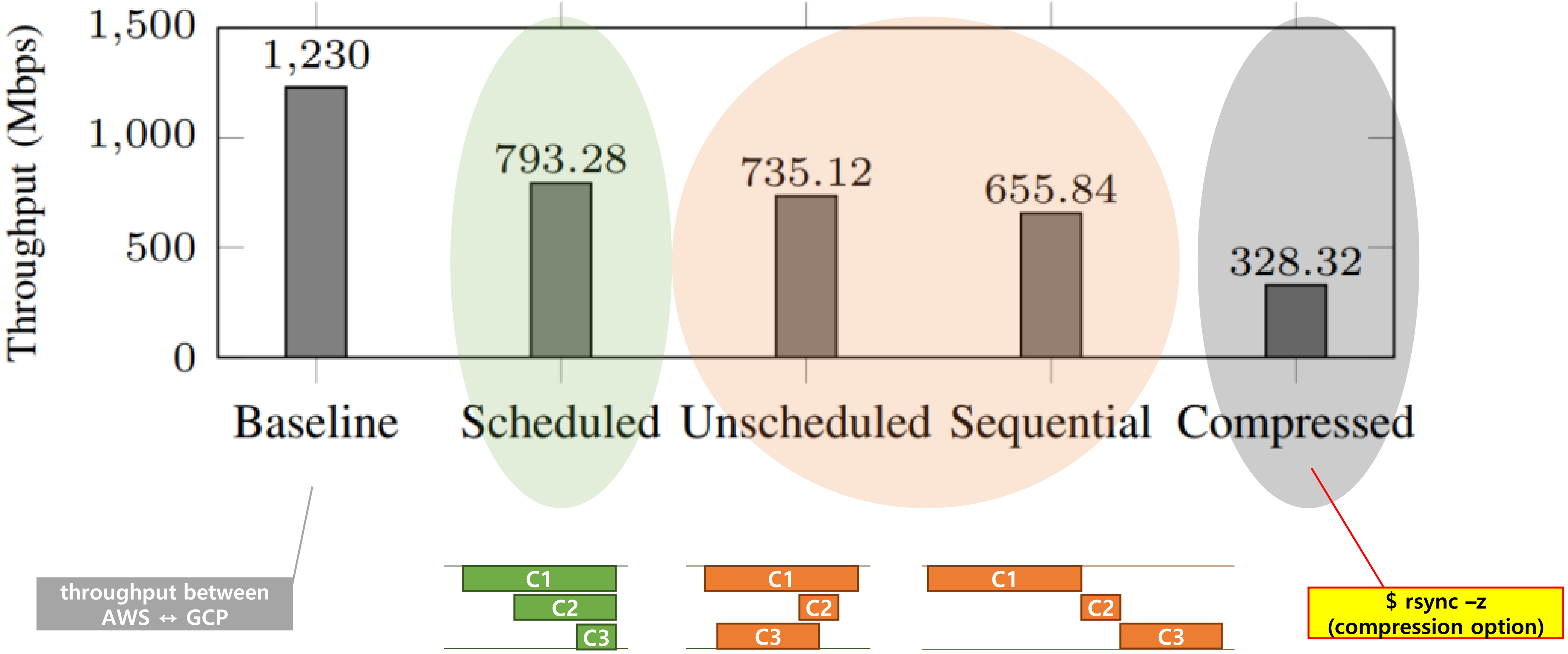
Scenario	Pre-Migration			Migration					Total
	Pre-Dump (s)	*Pre-copy (s)*	*Total (s)*	*Checkpoint (s)*	*Diff (s)*	*Transfer (s)*	*Restore (s)*	*Total Downtime (s)*	*(s)*
0c (Azure)	0.300	4.243	4.543	0.587	0.337	0.720	5.460	7.104	11.647
0c (GCP)	0.327	4.487	4.814	0.613	0.350	0.717	5.867	7.547	12.361
1c (GCP)	0.310	4.33	4.640	0.633	0.380	3.420	5.470	9.903	14.543
5c (GCP)	0.433	4.953	5.387	0.713	0.357	4.227	5.347	10.643	16.030
10c (GCP)	0.590	6.133	6.723	0.907	0.383	4.673	5.3	11.317	18.040
50c (GCP)	1.170	7.033	8.203	1.917	0.376	12.710	5.630	20.633	28.836
100c (GCP)	1.015	7.222	8.237	3.930	0.455	11.410	5.222	21.017	29.254
200c (GCP)	0.936	7.053	7.989	4.763	0.503	13.370	5.470	24.106	32.095
400c (GCP)	1.430	8.890	10.320	5.948	0.690	12.370	5.448	24.456	34.776

Acceptable?

60~70 TPS

w/o optimization

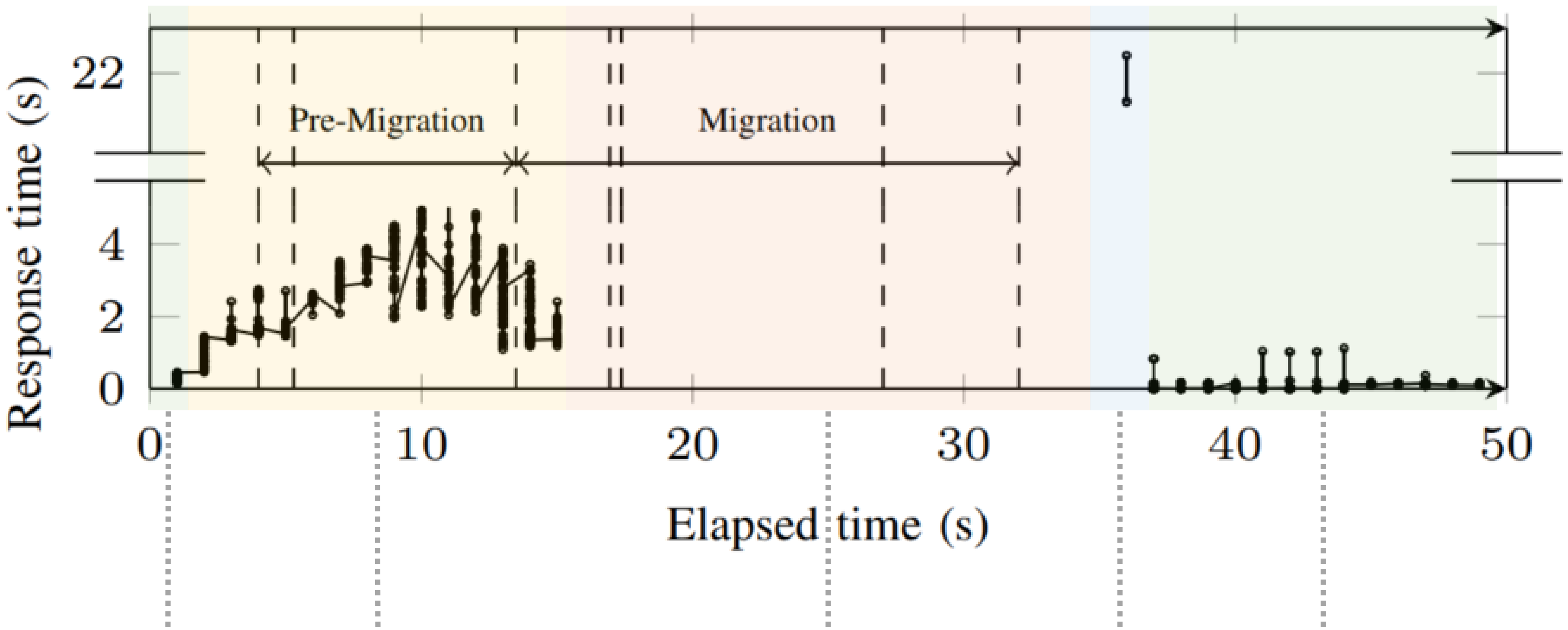
Result – Optimization



throughput between AWS ↔ GCP

$ rsync –z (compression option)

Result – Response Time



Normal State

Pre-dump
& Pre-copy

Hold

Redirect

Normal State

Critiques

- ✓ First approach of live migration using commercial cloud providers
- ✓ Well-organized migration flow and optimization techniques

- ✓ Too simple testbed (simple WebSite and DB container)
- ✓ Not enough load: 6~70rps (should be hundreds at least)
- ✓ Should consider de facto use case such as kubernetes (container orchestration)

- ✓ BTW, I cannot find ‘CloudHopper’ anywhere..

Thank you

Experiment environment

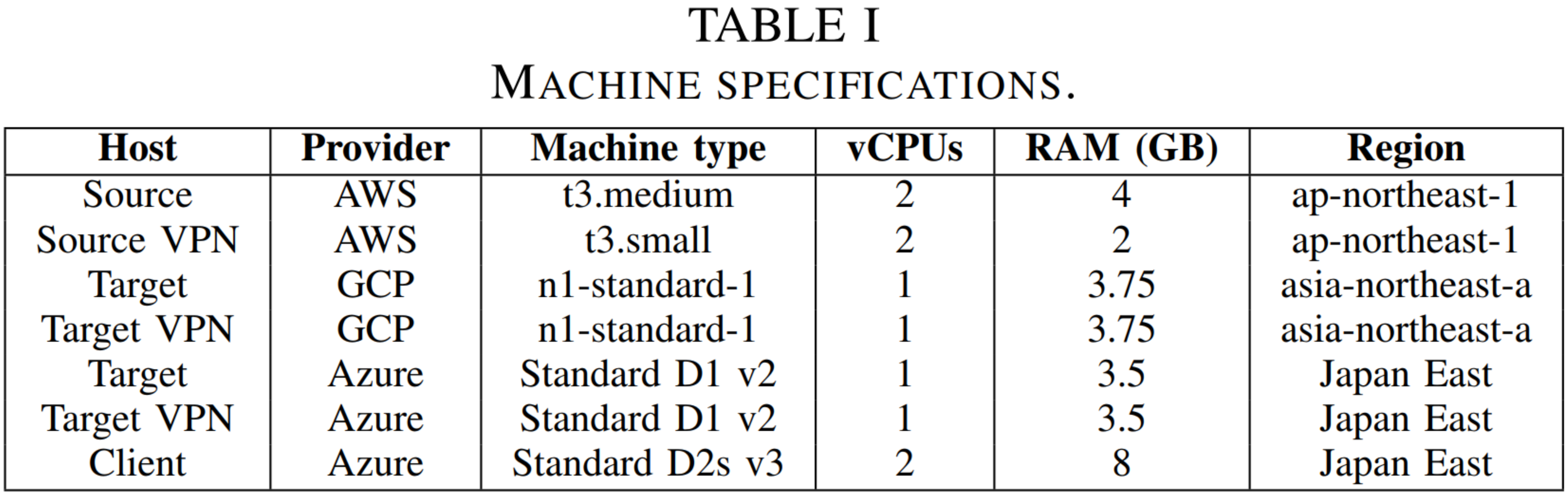


TABLE I
MACHINE SPECIFICATIONS.

Host	Provider	Machine type	vCPUs	RAM (GB)	Region
Source	AWS	t3.medium	2	4	ap-northeast-1
Source VPN	AWS	t3.small	2	2	ap-northeast-1
Target	GCP	n1-standard-1	1	3.75	asia-northeast-a
Target VPN	GCP	n1-standard-1	1	3.75	asia-northeast-a
Target	Azure	Standard D1 v2	1	3.5	Japan East
Target VPN	Azure	Standard D1 v2	1	3.5	Japan East
Client	Azure	Standard D2s v3	2	8	Japan East

TC Scenario

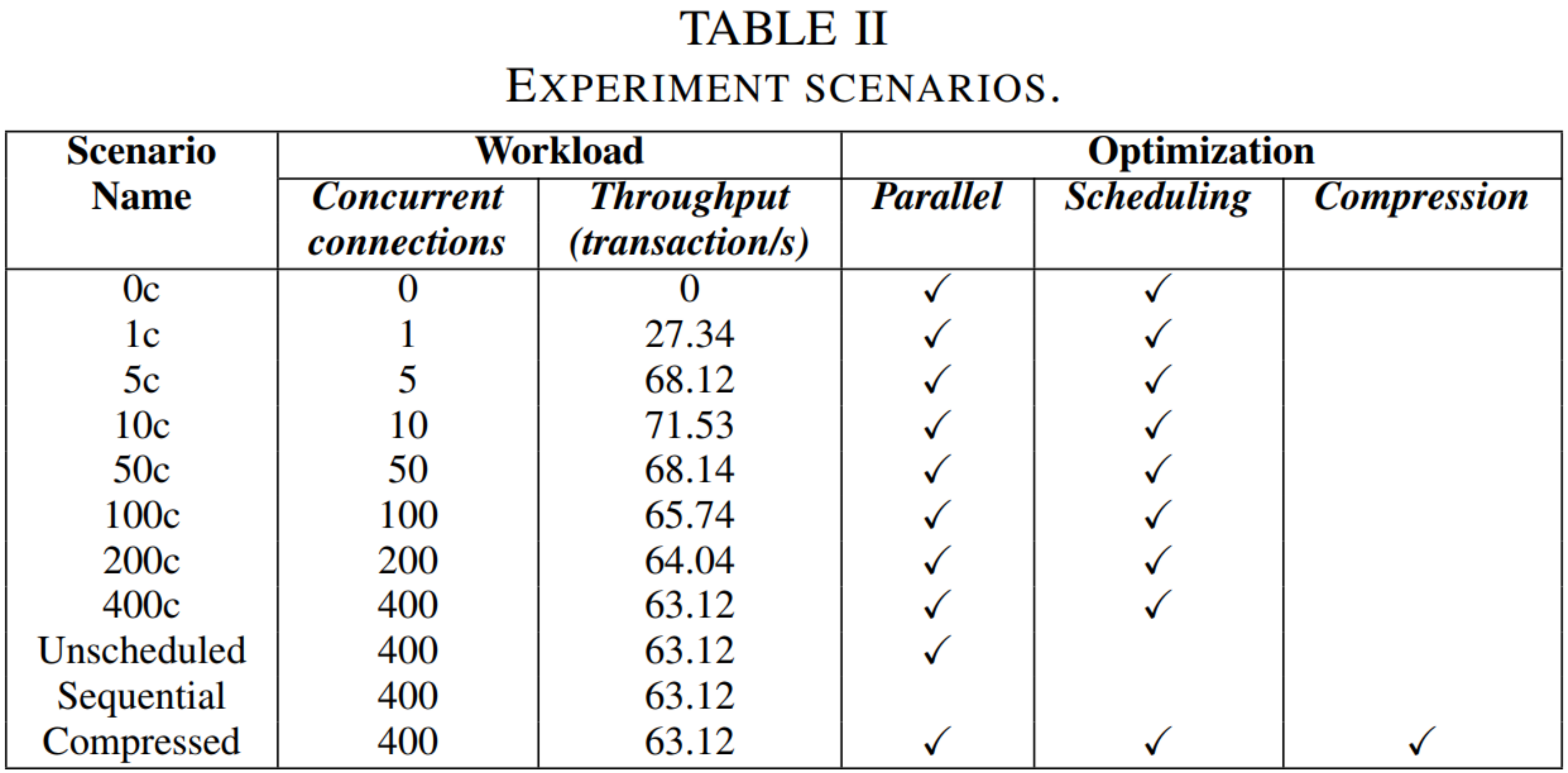


TABLE II
EXPERIMENT SCENARIOS.

Scenario Name	**Workload**		**Optimization**		
	Concurrent connections	***Throughput (transaction/s)***	***Parallel***	***Scheduling***	***Compression***
0c	0	0	✓	✓	
1c	1	27.34	✓	✓	
5c	5	68.12	✓	✓	
10c	10	71.53	✓	✓	
50c	50	68.14	✓	✓	
100c	100	65.74	✓	✓	
200c	200	64.04	✓	✓	
400c	400	63.12	✓	✓	
Unscheduled	400	63.12	✓		
Sequential	400	63.12			
Compressed	400	63.12	✓	✓	✓

Networking Interface

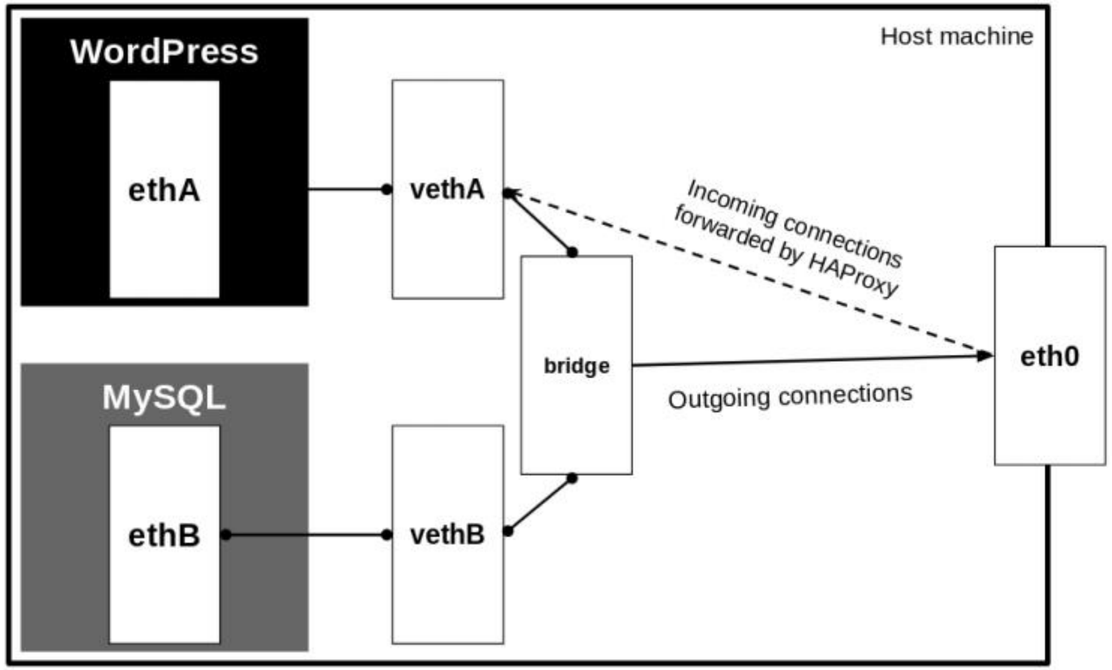
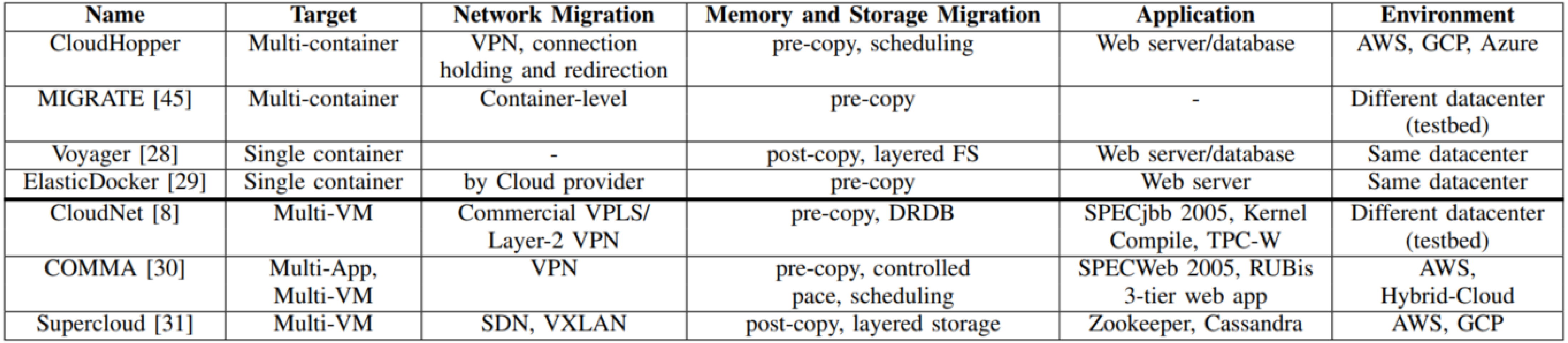


Fig. 2. Container networking setup using namespaces, virtual interfaces, bridge, and HAProxy.

Related works

TABLE V
COMPARISON BETWEEN RELATED WORK.



Name	Target	Network Migration	Memory and Storage Migration	Application	Environment
CloudHopper	Multi-container	VPN, connection holding and redirection	pre-copy, scheduling	Web server/database	AWS, GCP, Azure
MIGRATE [45]	Multi-container	Container-level	pre-copy	-	Different datacenter (testbed)
Voyager [28]	Single container	-	post-copy, layered FS	Web server/database	Same datacenter
ElasticDocker [29]	Single container	by Cloud provider	pre-copy	Web server	Same datacenter
CloudNet [8]	Multi-VM	Commercial VPLS/ Layer-2 VPN	pre-copy, DRDB	SPECjbb 2005, Kernel Compile, TPC-W	Different datacenter (testbed)
COMMA [30]	Multi-App, Multi-VM	VPN	pre-copy, controlled pace, scheduling	SPECWeb 2005, RUBis 3-tier web app	AWS, Hybrid-Cloud
Supercloud [31]	Multi-VM	SDN, VXLAN	post-copy, layered storage	Zookeeper, Cassandra	AWS, GCP