CS6650

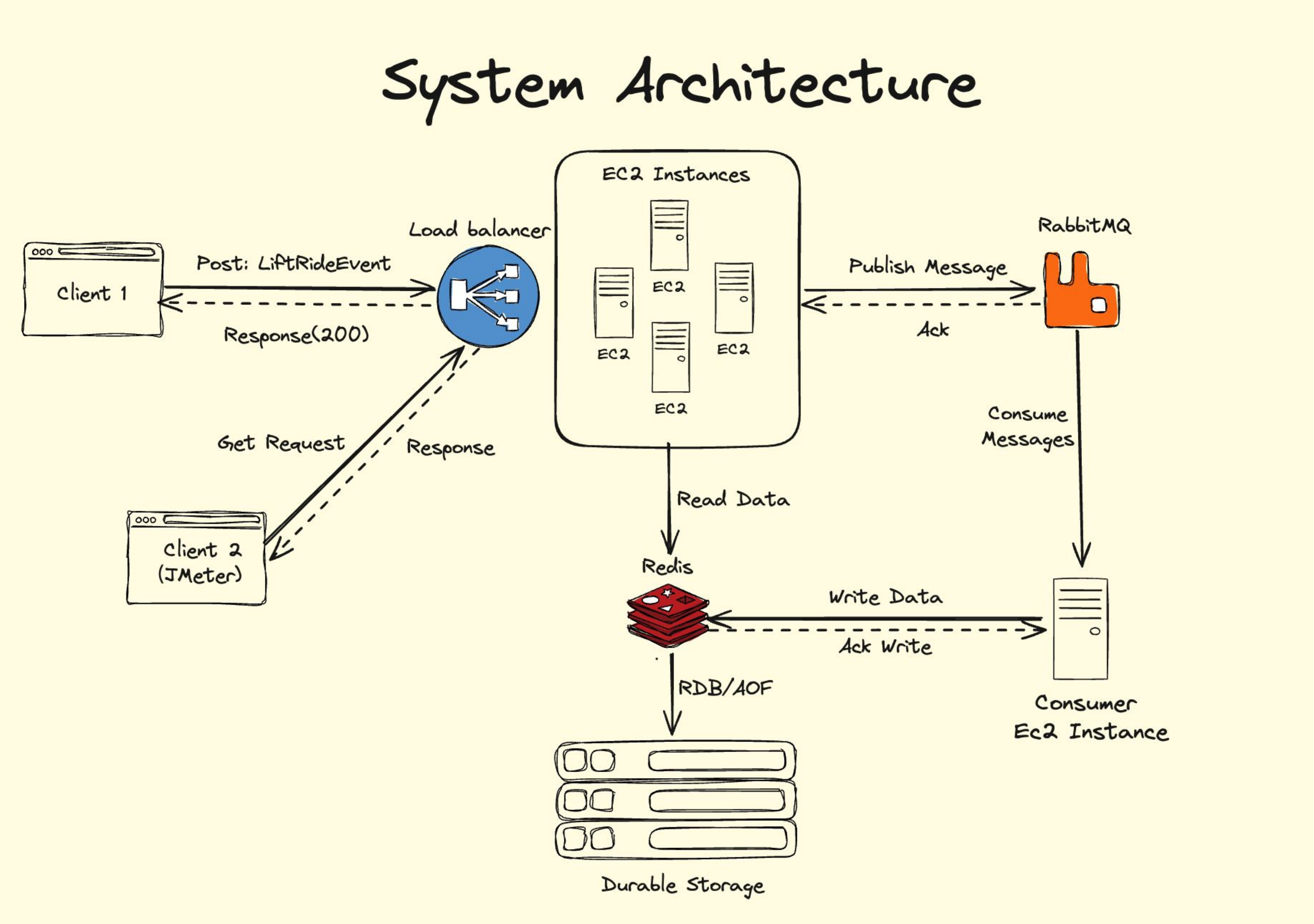
Cache Money

CONTENTS

- Architecture
- Data model
- Jmeter tests
- Future improvement

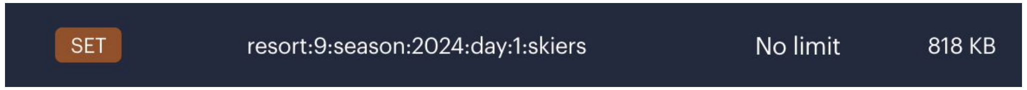
Architecture Overview

- **Application Load Balancer** with a total of **4 EC2 Instances**
- **RabbitMQ** deployed on a single instance for message queuing and async processing
- **Redis** used as a database, with RDB and AOF for durability
- **Consumer** on a separate instance to consume messages from MQ and writes to Redis.
- **Client 1 sends** 'LiftRideEvent' POST requests to server
- **Client 2** sends GET request for load testing (JMeter)



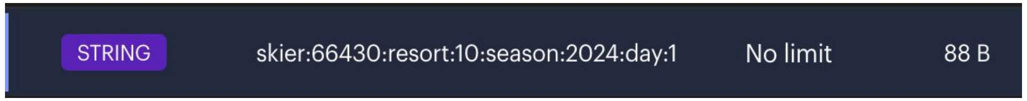
DATA MODEL

- We use three independent data structures in Redis, each corresponding to a separate GET request.
- GET/resorts/{resortID}/seasons/{seasonID}/day/{dayID}/skiers
  (Redis Set)



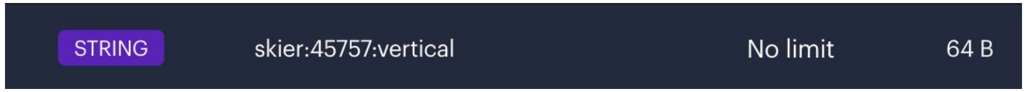
SET resort:9:season:2024:day:1:skiers No limit 818 KB

- GET/skiers/{resortID}/seasons/{seasonID}/days/{dayID}/skiers/{skierID}
  (Redis String)



STRING skier:66430:resort:10:season:2024:day:1 No limit 88 B

- GET/skiers/{skierID}/vertical
  (Redis String)



STRING skier:45757:vertical No limit 64 B

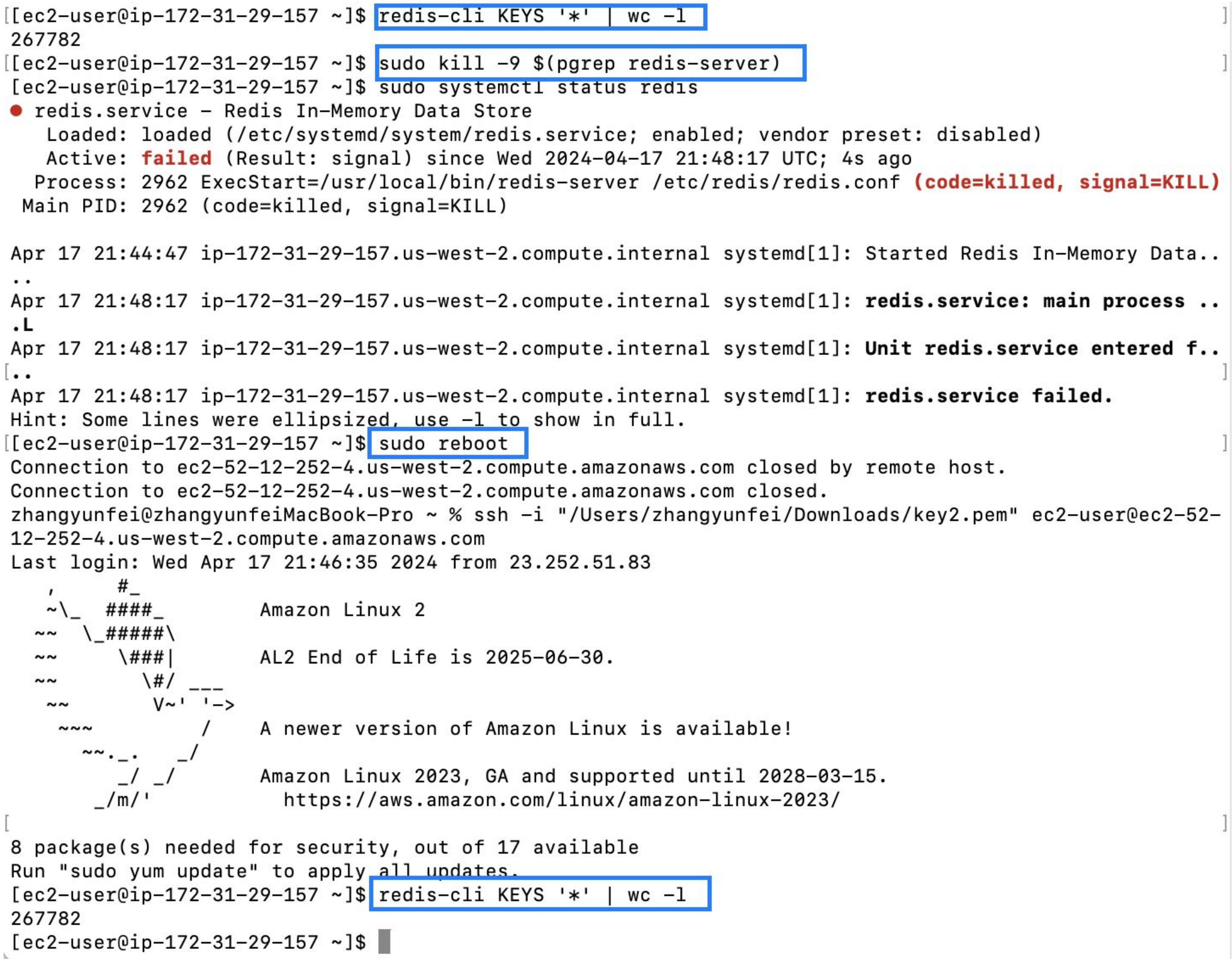
TRADE OFF

- Pros of Using Reids:
  1. Extremely High Performance
  2. Simple Data Structures
  3. Scalability: Redis can be scaled easily by increasing the number of nodes in the cluster.

- Cons of Using Redis
  1. Data Durability
  2. Memory Limitation

DISASTER RECOVERY

- Enable RDB in disk snapshot in redis.conf
- Redis RDB persistence is a snapshotting method where Redis saves its dataset to disk at specified intervals. This approach is beneficial for disaster recovery because it provides a point-in-time backup of data, which can be used to restore the system after a crash.
- Using RDB will lose some performance, but considering the disaster recovery and availability improvements, this is a reasonable trade off.



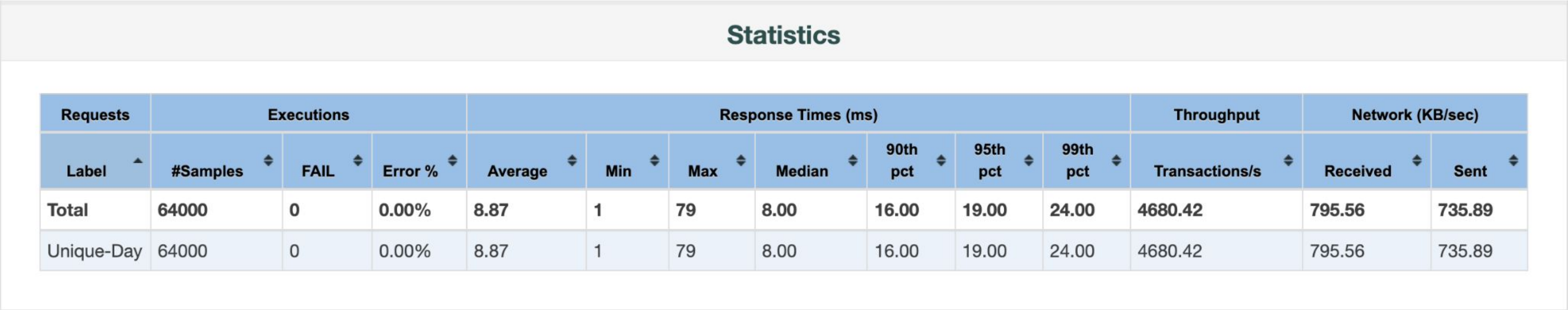
```
[ec2-user@ip-172-31-29-157 ~]$ redis-cli KEYS '*' | wc -l
267782
[ec2-user@ip-172-31-29-157 ~]$ sudo kill -9 $(pgrep redis-server)
[ec2-user@ip-172-31-29-157 ~]$ sudo systemctl status redis
● redis.service - Redis In-Memory Data Store
   Loaded: loaded (/etc/systemd/system/redis.service; enabled; vendor preset: disabled)
   Active: failed (Result: signal) since Wed 2024-04-17 21:48:17 UTC; 4s ago
  Process: 2962 ExecStart=/usr/local/bin/redis-server /etc/redis/redis.conf (code=killed, signal=KILL)
 Main PID: 2962 (code=killed, signal=KILL)

Apr 17 21:44:47 ip-172-31-29-157.us-west-2.compute.internal systemd[1]: Started Redis In-Memory Data..
..
Apr 17 21:48:17 ip-172-31-29-157.us-west-2.compute.internal systemd[1]: redis.service: main process ..
.L
Apr 17 21:48:17 ip-172-31-29-157.us-west-2.compute.internal systemd[1]: Unit redis.service entered f..
..
Apr 17 21:48:17 ip-172-31-29-157.us-west-2.compute.internal systemd[1]: redis.service failed.
Hint: Some lines were ellipsized, use -l to show in full.
[ec2-user@ip-172-31-29-157 ~]$ sudo reboot
Connection to ec2-52-12-252-4.us-west-2.compute.amazonaws.com closed by remote host.
Connection to ec2-52-12-252-4.us-west-2.compute.amazonaws.com closed.
zhangyunfei@zhangyunfeiMacBook-Pro ~ % ssh -i "/Users/zhangyunfei/Downloads/key2.pem" ec2-user@ec2-52-
12-252-4.us-west-2.compute.amazonaws.com
Last login: Wed Apr 17 21:46:35 2024 from 23.252.51.83
   ,     #_
   ~\_  ####_        Amazon Linux 2
  ~~  \_#####\
  ~~     \###|       AL2 End of Life is 2025-06-30.
  ~~       \#/ ___
   ~~       V~' '->
    ~~~         /    A newer version of Amazon Linux is available!
      ~~._.   _/
         _/ _/       Amazon Linux 2023, GA and supported until 2028-03-15.
       _/m/'           https://aws.amazon.com/linux/amazon-linux-2023/

8 package(s) needed for security, out of 17 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-29-157 ~]$ redis-cli KEYS '*' | wc -l
267782
[ec2-user@ip-172-31-29-157 ~]$ 
```

JMETER TESTS

- The JMeter tests were conducted on an EC2 instance, employing two different server configurations: a single server and a load-balanced setup with 4 instances. Each server configuration was evaluated using three different thread counts: 128, 300, and 400.
- Test with 128 threads gave us the lowest response times and p99s.

JMETER tests results: GET/resorts/{resortID}/seasons/{seasonID}/day/{dayID}/skiers

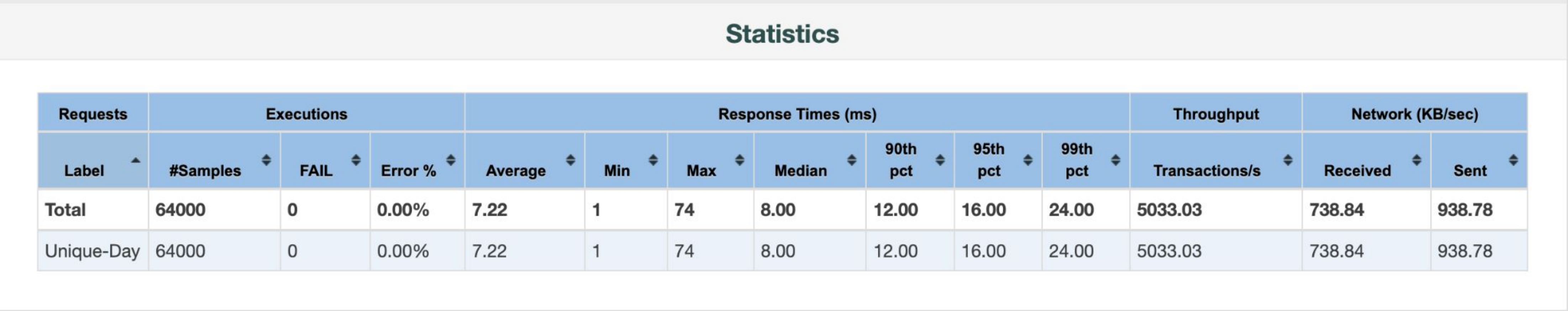
- Single service



Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**64000**	**0**	**0.00%**	**8.87**	**1**	**79**	**8.00**	**16.00**	**19.00**	**24.00**	**4680.42**	**795.56**	**735.89**
Unique-Day	64000	0	0.00%	8.87	1	79	8.00	16.00	19.00	24.00	4680.42	795.56	735.89

- Load balance

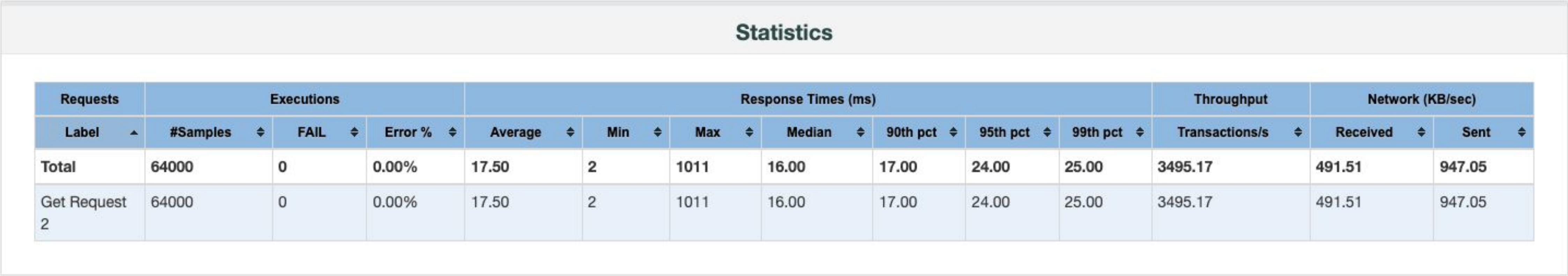


Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**64000**	**0**	**0.00%**	**7.22**	**1**	**74**	**8.00**	**12.00**	**16.00**	**24.00**	**5033.03**	**738.84**	**938.78**
Unique-Day	64000	0	0.00%	7.22	1	74	8.00	12.00	16.00	24.00	5033.03	738.84	938.78

JMETER tests results: GET/skiers/{resortID}/seasons/{seasonID}/days/{dayID}/skiers/{skierID}

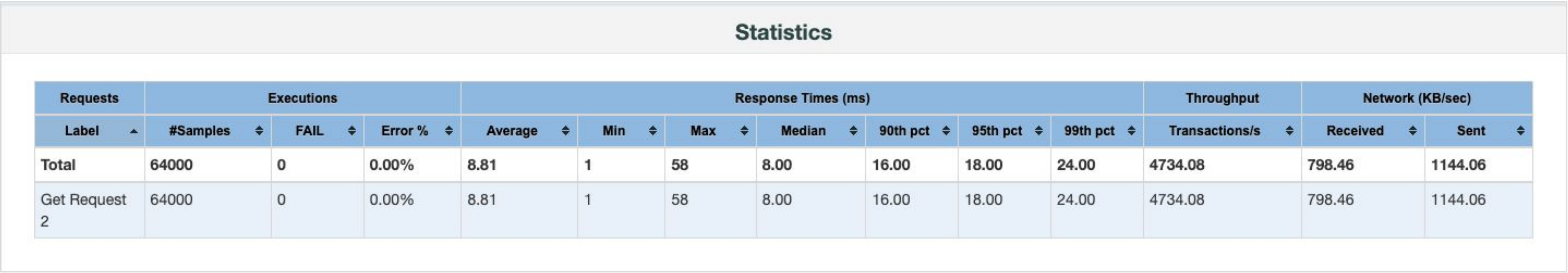
- Single service



Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**64000**	**0**	**0.00%**	**17.50**	**2**	**1011**	**16.00**	**17.00**	**24.00**	**25.00**	**3495.17**	**491.51**	**947.05**
Get Request 2	64000	0	0.00%	17.50	2	1011	16.00	17.00	24.00	25.00	3495.17	491.51	947.05

- Load balance

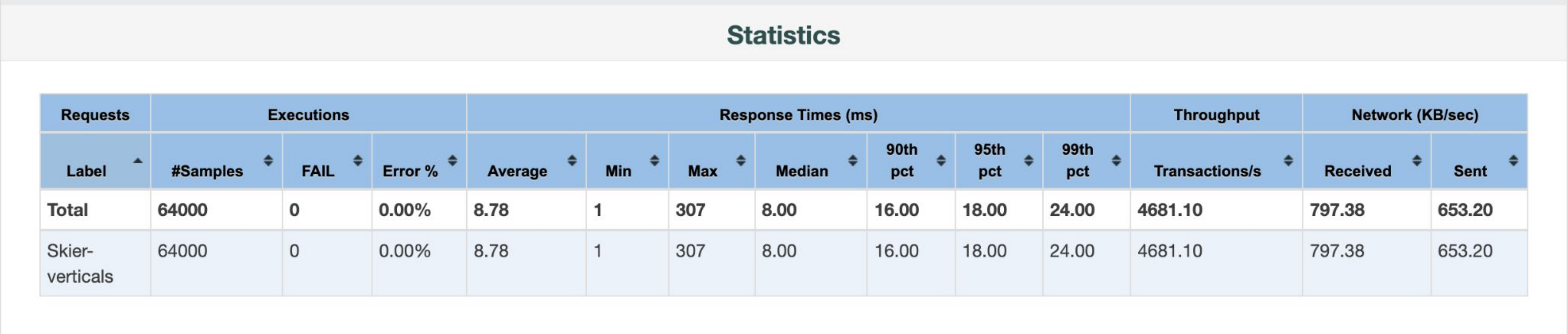


Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**64000**	**0**	**0.00%**	**8.81**	**1**	**58**	**8.00**	**16.00**	**18.00**	**24.00**	**4734.08**	**798.46**	**1144.06**
Get Request 2	64000	0	0.00%	8.81	1	58	8.00	16.00	18.00	24.00	4734.08	798.46	1144.06

JMETER tests results: GET/skiers/{skierID}/vertical

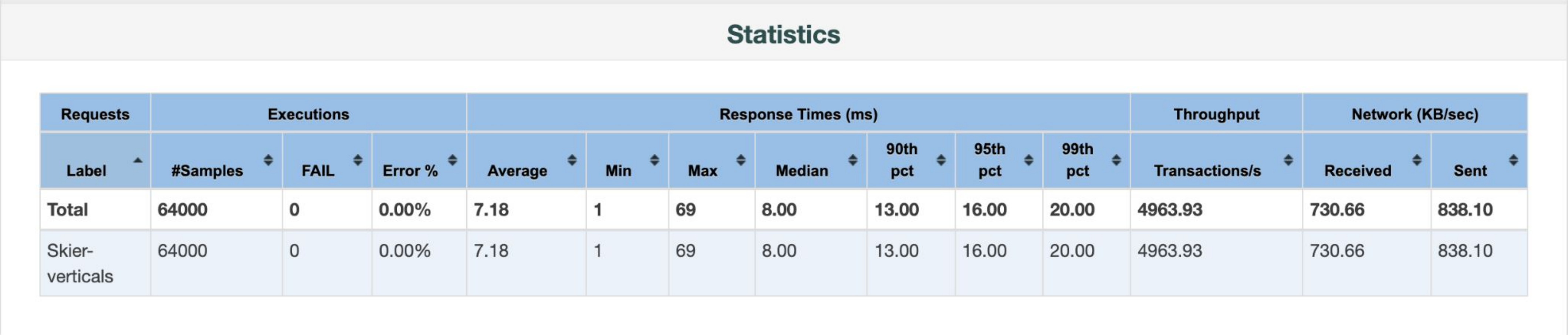
- Single service



Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**64000**	**0**	**0.00%**	**8.78**	**1**	**307**	**8.00**	**16.00**	**18.00**	**24.00**	**4681.10**	**797.38**	**653.20**
Skier-verticals	64000	0	0.00%	8.78	1	307	8.00	16.00	18.00	24.00	4681.10	797.38	653.20

- Load balance

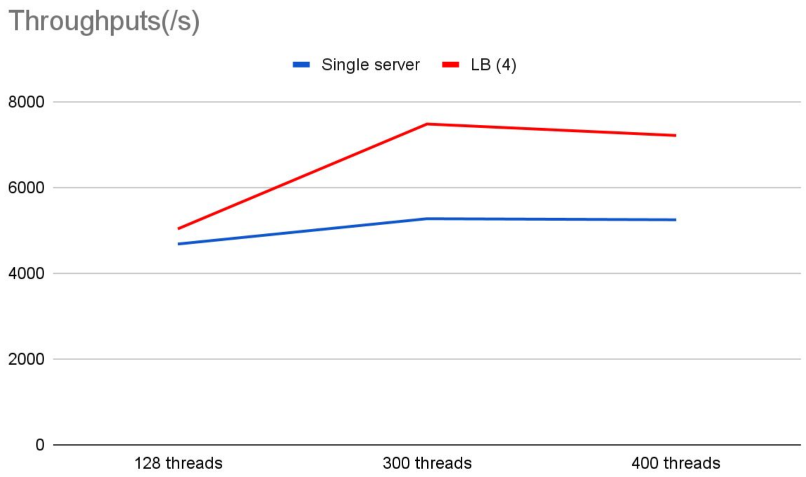


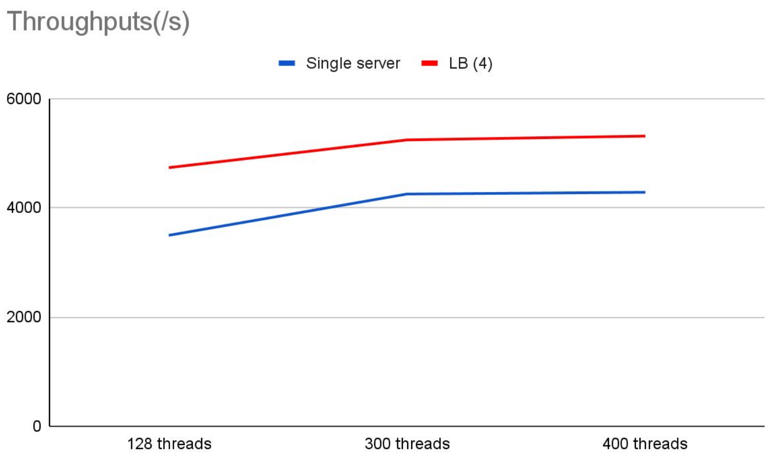
Statistics

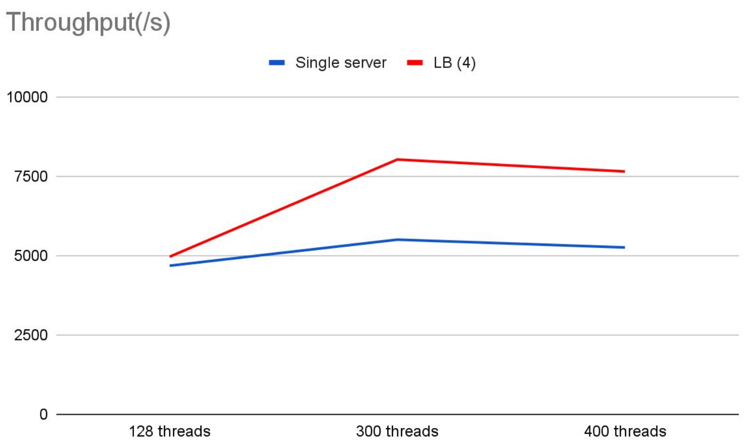
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**64000**	**0**	**0.00%**	**7.18**	**1**	**69**	**8.00**	**13.00**	**16.00**	**20.00**	**4963.93**	**730.66**	**838.10**
Skier-verticals	64000	0	0.00%	7.18	1	69	8.00	13.00	16.00	20.00	4963.93	730.66	838.10

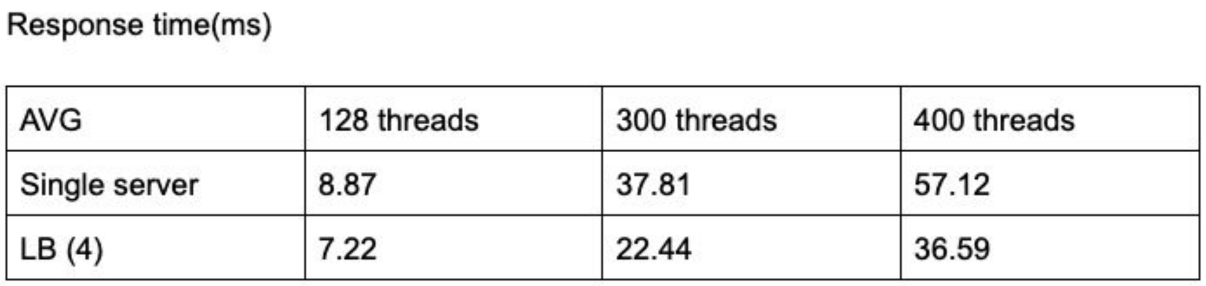
JMETER tests results

- Single service VS Load balance



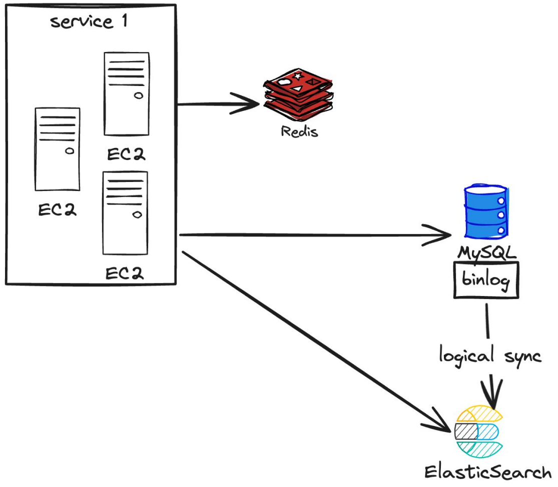





Response time(ms)

AVG	128 threads	300 threads	400 threads
Single server	8.87	37.81	57.12
LB (4)	7.22	22.44	36.59

Future improvement



- Current Design (store all the data in the redis)
  - Occupied much memory
  - Can’t support other type of query very easily
- Further Design (use MySQL to store data, use ES to speed up query)
  - Use MySQL to store origin data (skier’s data)
  - Use Redis as cache to speed up the GET query
  - Use ElasticSearch to Speed up complex search (join search) and aggregation