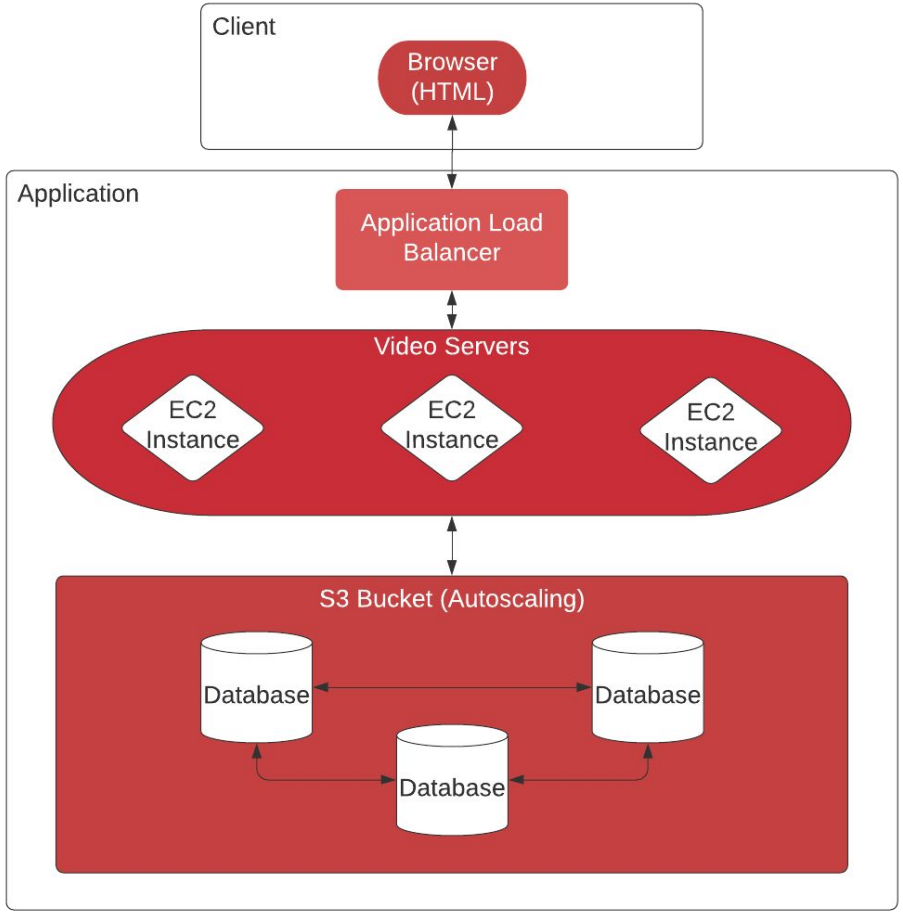
Use Case

- Load balancing is a key server architecture process
- Problem: insufficient public knowledge on dynamic load balancing
  - Commercial options only offer “black-box” solutions
  - Useful for building scalable and more integrated open-source-based servers

- Solution: Testing and documenting performance of LB algorithms of interest in a real world environment video streaming server
  - Comparing traditional parameter-based LBs and experimental ML models

Server Architecture

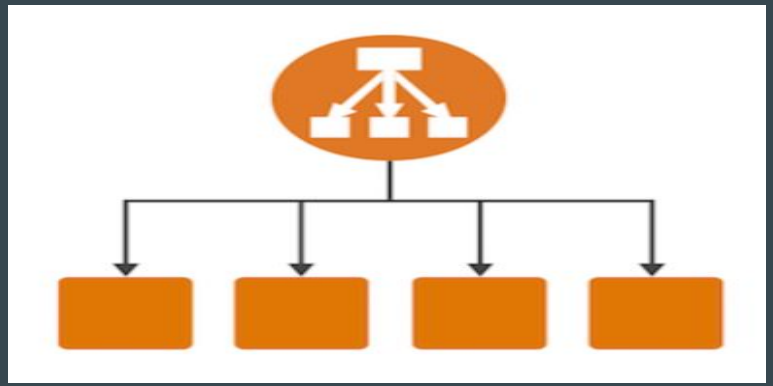
- More streamlined
  - Now implementing Application LB, full-stack servers
  - Amazon S3 DB is auto-scaling; removes need for internal LB
- NodeJS Frameworks
  - HTTP-proxy load balancer
  - AWS-SDK video retrieval
- AWS group CodeDeploy



Load Balancing Algorithms (Benchmark)

- Fixed-decision traditional algo's: RandomLB and RoundRobin

- AWS Elastic Load Balancing
  - Standard load balancing performance (commonly used blackbox solution)
  - Can be preset at high-level to consider various metrics (CPU%, Network I/O, etc.)

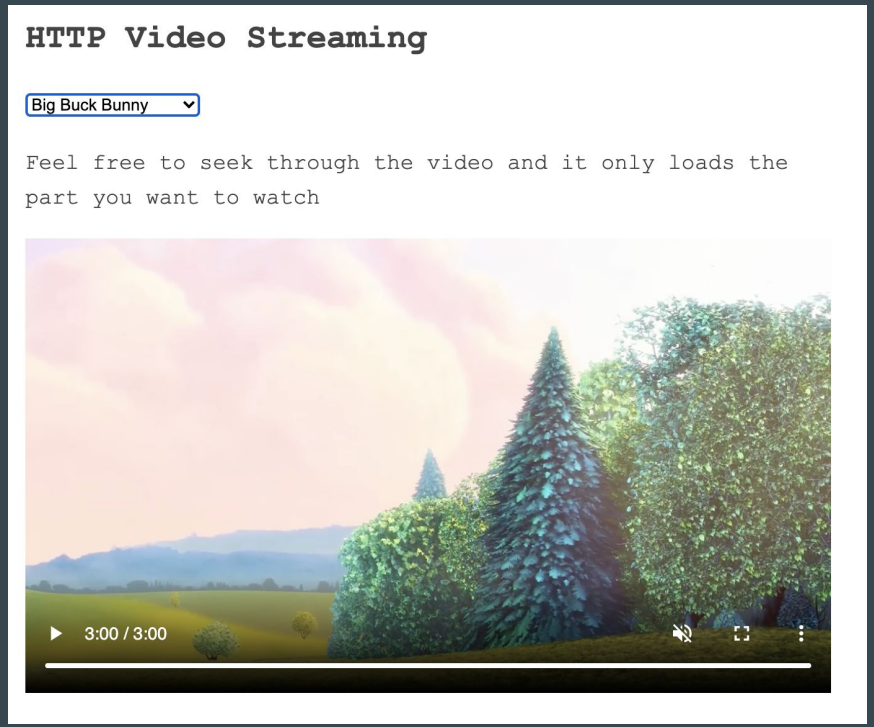


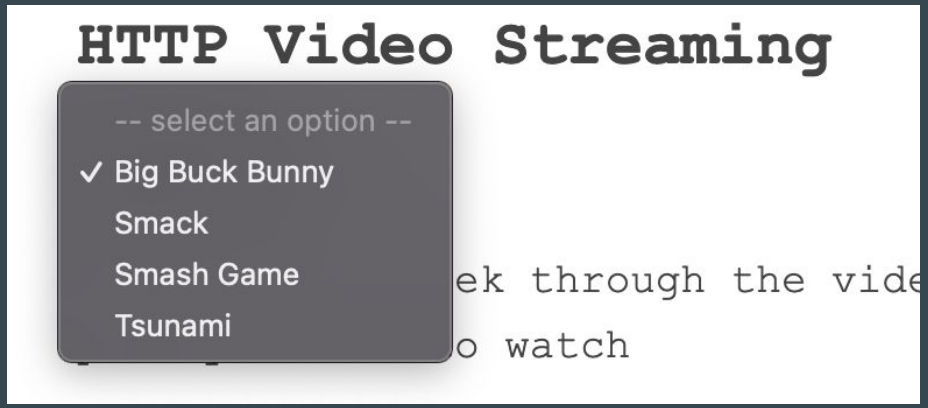
Load Balancing Algorithms (Custom)

- Multi-armed Bandit Reinforcement Learning Scenario
  - Online learning with reward function of a server metric
  - Exploitation vs Exploration
  - Key difference is variable reward with no asymptotic limit
    - Consider recent results and consistently explore

- Two algorithm classes based loosely on Epsilon-Greedy and UCB1
  - Two LB decider types with 2 different server metric (4 total)
    - local response time (ms) or network I/O (bytes)
  - Epsilon-greedy has fixed chance of naive exploration (e.g. ½ chance to pick at random)
  - Custom UCB1 explores more consistently by only picking from 50% least recent

User View - video streaming server



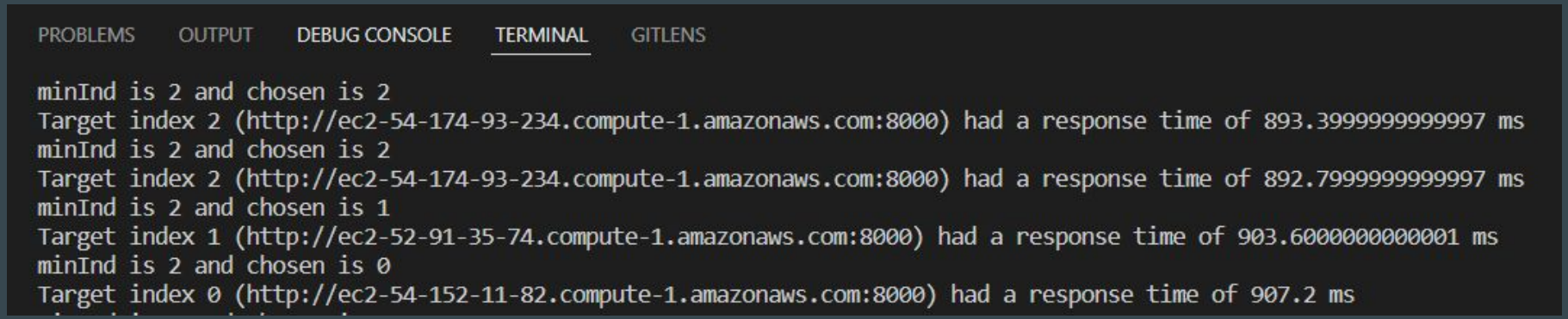


- Can pick b/w 4 videos of varying length and quality
- Streamed in 1MB chunks to HTML5 video element

Internal View: LB Proxy Server

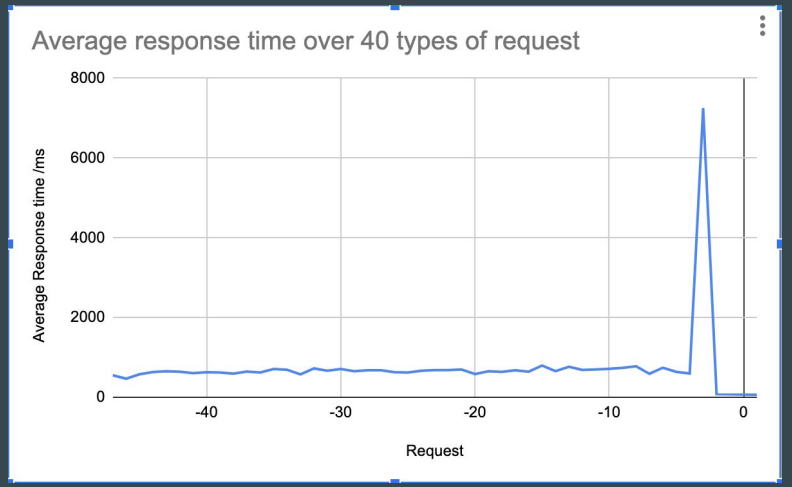
The load balancer makes the decision as to which video server to pick based on the algorithm it is running

Output shown from load balancer using egreedyRT algorithm:



Data View: User Simulation Results

- .csv generated by JMeter scripts (average response time, bit rate, etc.)
- Transformed into insightful graphs using external tools (excel, Python)



Label	# Samples	Average	Min	Max	Std. Dev.	Error %
	100	7257	112	36153	11468.40	0.00%
-1	25	62	51	87	9.02	0.00%
-10	25	714	309	1475	344.95	0.00%
-11	25	697	321	1343	317.65	0.00%
-12	25	686	380	1586	288.37	0.00%
-13	25	764	373	1896	393.19	0.00%
-14	25	654	322	1263	263.71	0.00%
-15	25	795	344	1393	319.81	0.00%
-16	25	642	321	1713	331.67	0.00%

Quantitative Specifications

Metric	Benchmark	Description
User Metrics		
User Latency	2s	Time elapsed between the video request from the user and receiving video data from the server
Bit Rate	Video-based	How many bits are transmitted over a specified time
Load Balancer Metrics		
LB Latency	500 ms	Time elapsed between the LB receiving the request until a response from the video server is received
CPU Utilization		Processor utilization (%) of a server/VM (async)
Network I/O		Data volume (bytes) of input/output to a server (async)

Testing Plan

- Cannot simply have scripts sending requests for video chunks at regular intervals
  - Does not properly simulate how video chunks are being requested by video player
- Need to rely on simulating behavior of HTML5 video player
- Tried tools such as Selenium, Taurus, and JMeter with third party load testers such as dotcom-monitor, flood.io, and BlazeMeter

Final Testing Suite

- Generate different classes of users by recording browser behavior and converting them into JMeter files
- Run JMeter scripts through BlazeMeter to retrieve user data for each algorithm
- Compare between algorithms by generating graphs using resulting .csv files

Measures of Success

- Meeting basic benchmarks mentioned before while servers are under load
- Comparing performance of our custom load balancers with that of the benchmark load balancers
    - Finding user contexts where our custom load balancers performs better over benchmark
    - Altering algorithm values such as the value of epsilon or fraction of servers considered in UCB-1

Design Trade-offs

- Metric Retrieval Change
  - LBs now decide on local server metrics instead of user metrics
  - Less useful decision indicator but more reliable retrieval
    - Faster and more reliable metric retrieval
    - Cannot expect user metrics in typical server-client contract

- Architecture Streamlining
  - Streamlined architecture corresponds less to large-scale video streaming
  - However, makes load-balancing decisions more relevant to user data

Project Management

TASK TITLE	TASK OWNER	04/04 - 04/10	04/11 - 04/17	04/18 - 04/24	4/25 - 05/01
Project Building and Launch					
Set Up and Test AWS Connections	Jason				
Initial Video Server	Nakul				
Implement Benchmark Load Balancers	Mitul				
Obtaining Response Time for Load Balancer	Mitul				
Implementing Video Chunks	Jason				
Implement Epsilon-Greedy Load Balancer	Mitul				
Begin End User Simulation	Jason				
Obtaining Network I/O	Nakul				
Implement UCB1-Based Load Balancer	Mitul				
User Simulation Data Graphing/Comparison	Nakul				
AWS Deployment	Jason				