



Recitation 8

15440 - Distributed Systems - Spring 2023

P3 Checkpoint 2 and 3

Michelle Deng & Joseph Salmento, March 22, 2023



Agenda

- Timeline
- Project 3 Overview
- Tips for Project 3
 - General Tips
 - Checkpoint 2
 - Checkpoint 3



Timeline

Project 3:

~~Checkpoint 1 due: Tuesday **March 21, 2023**, 11:59 pm EST~~

Checkpoint 2 due: Tuesday **March 28, 2023**, 11:59 pm EST

Checkpoint 3 due: Tuesday **April 4, 2023**, 11:59 pm EST



General Overview

- Implementing a web service for an online store
- Servers receive requests from clients and handle requests by passing them off to the database
- Simulated load balancer is used to benchmark performance of system
- Checkpoint 1
 - Fixed load, diurnal load curve
- Checkpoint 2
 - Non predictable load over time of day, constant over test run
- Checkpoint 3
 - Load changes over test run



General Tips

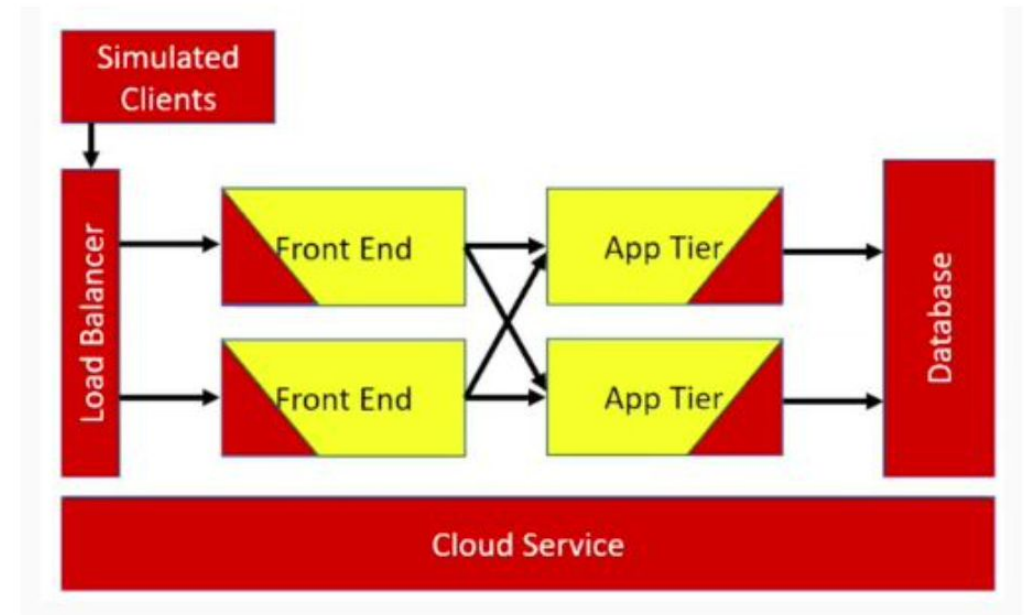
- **Start early!** Checkpoint 2 and 3 take significantly longer than Checkpoint 1
- Programming takes less time, performance tuning takes more time
- Keep benchmarking and testing throughout entire process, you will be optimizing in every stage of the process
- System is inherently non-deterministic, so expect scores to vary across runs (VMs are implemented as processes)
- Autolab can take many minutes to run, so don't rely on it for feedback; Autolab takes longer closer to the deadline, since many students are submitting



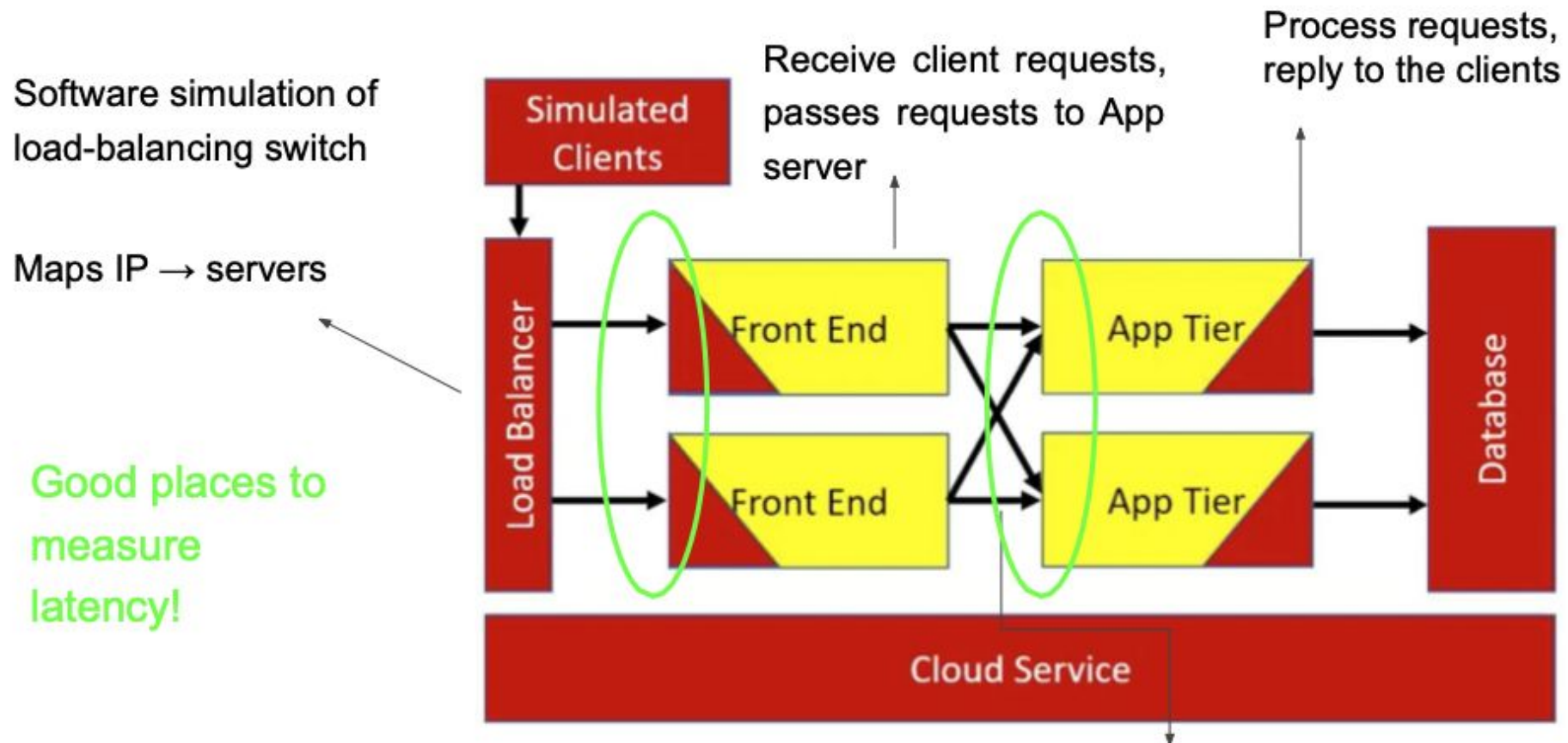
Checkpoint 2

Checkpoint 2

- Separate web front-end and application tier
- Only front ends register with load balancer and call `acceptConnection/parseRequest`
- Only App servers call `processRequest`
- Each tier needs to be scaled out independently
- Each new VM launches the server program - will need to know how to coordinate and determine its role/tier



Checkpoint 2



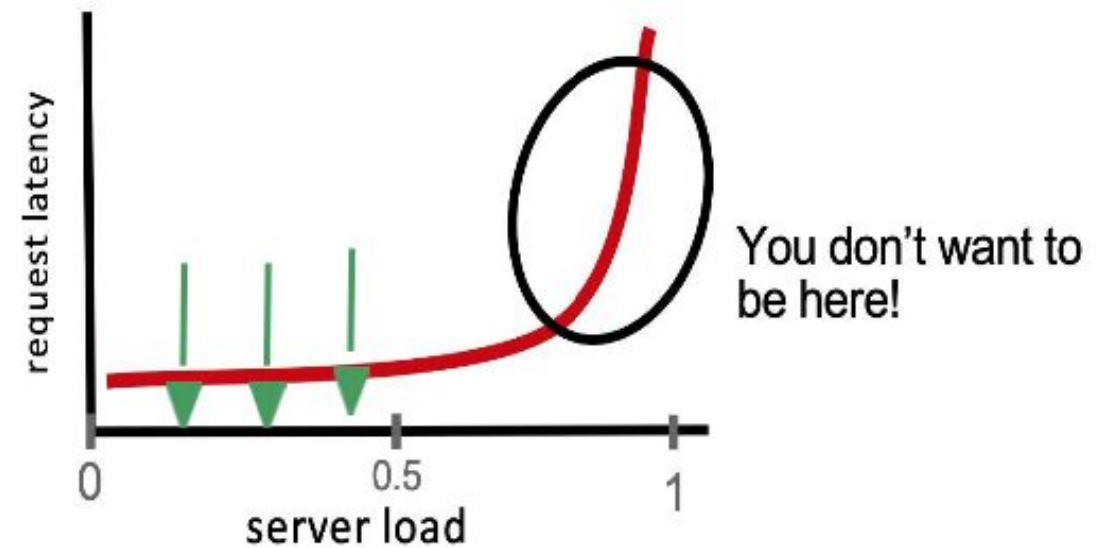


When to scale out?

- You will need to dynamically scale depending on the load you are receiving
 - What behaviors will tell you when to begin scaling?
 - How will you decide which layer to scale?
- Successful implementations take effort & trials!
 - The majority of this lab is about fine tuning
 - There is not a lot of code but lots of testing

When to scale out?

- Remember: it takes time to scale
- There will be a delay between recognizing the problem and fixing it
- Be careful to not be overly conservative when scaling out





What are some metrics you can use to determine when to scale up or down?

- Think about and discuss with your neighbor some different metrics you can use to determine when to scale up or down?
- What are the some advantage or disadvantages to that metric?
- We will discuss/ share them with the class after 5-10 minutes.



Adaptation Metrics/Heuristics

- CPU load
- Queue lengths
- Arrival times
- Number of dropped requests



Adaptation Metrics: CPU Load

- Not a great metric
- 100% load can be fine (the server can be keeping up)
- <100% load can be bad (overload due to other limitations – bottlenecks in disk, I/O, network...)
- 100% may not be obtainable but still fully loaded
- **Do not use this as your main decision-making metric**
 - You can take it into consideration with other measurements




Adaptation Metrics: Queue Lengths

- Provides useful information, but not as much as many students think
- One main takeaway: **the server is either keeping up, or it is not**
- If the server is keeping up, the queue length is 0 or 1
- If the server is overloaded, the queue length will grow indefinitely
- Many students attempt ideas such as setting the number of servers proportional to the queue length – **this will not work**



Adaptation Metrics: Arrival Times

- Inter-arrival times are great information, they provide good estimates of the arrival rate
- However, **they are noisy**, and an attempt should be made to smooth this
 - Averaging over longer time intervals, helps but increases delay before a change can take effect
- Fundamentally, there are trade-offs between getting stable, clean, and useful data vs agility (speed of reacting)



Adaptation Metrics: Number of Dropped Requests

- Why do we need to drop requests?
 - Too many requests, cannot handle them in time
 - Drop some requests, make some other clients happy
- When to drop requests
 - Try some metrics based on experiments
 - Measure time from start of `acceptConnection` to end of `handleRequest`
 - Queue length can be a good approximation
 - Not an exact science



Hysteresis

- There will be lags between your changes and their effects
- Be careful about randomness and noise
- Sometimes it may seem like more servers are needed, but immediately after, it may seem like there are too many present
- Use some form of hysteresis to avoid over-oscillating/over-reacting
- Dual-thresholding: consider having two separate thresholds, one for scaling up and one for scaling down
 - Then, noise won't cause you to continuously flip back and forth across a threshold
- Temporal delays: consider not making changes for a certain amount of time after the previous change
 - It takes time for changes to take effect, time for VMs to boot



Pitfalls: Poor VM Usage

- **Poor VM usage:** if the appropriate number of VMs are being created, but they are not being used most efficiently
- This is a coordination bug that can lead to bad imbalances and potentially dropped clients
- Having a main coordinator VM (that holds a queue) is a good strategy, but they may not be doing real work
- Every server should act as either a front end or middle tier server, even if they are also a coordinator



Pitfalls: Responsibility Mismanagement

- Be careful not to mix front end and middle tier operations in the same VM
- This will lead to delays, making servers much less capable, leading to more being needed
- Only during the initial serve should a VM do both, and this should only be for the initial wait time for the additional VMs to boot



Pitfalls: Slow Reaction Times

- Your code may make all the right analyses and correct decisions, but if the reaction is too slow, you may not be able to benefit from it
- Getting to the right number of servers too slowly will still result in many unhappy clients
- This can also lead to an excess for an intense load that no longer exists
- You can get both too many unhappy clients and too much VM time -
Your reaction time is probably off



Helpful Comments

- Minimal code, but this does not make the checkpoint easy
- Focus on performance testing + benchmarking
- Run your simulation at various loads (change `<rand_spec>` to make sure your code is truly dynamic
- If there's no chance to handle a client in time, drop request
- **Start early!!**



Checkpoint 2 Benchmark Testing Tip

- Do a static configuration for benchmark testing
- We want the number of servers to scale dynamically, but it is hard to measure the success of an implementation if you change too many variables at once - use static loads for initial testing
- Repeat tests like in Checkpoint 1, but with varying amounts of front/app servers to get the least unhappy clients with the least VMs
- Try this for a variety of different loads until the “right” number is consistently reached
- Then enable dynamic adaptation and see if your code gets the right combinations
- Repeat this process with multiple runs and different seeds



Hidden Simplifying Assumptions

- Service time is constant
- Arrival time is variable



Checkpoint 3



Checkpoint 3: Dynamic Loads

- Arrival rates will fluctuate suddenly
- Must be able to handle rapid increases/decreases in load
- Overall similar to Checkpoint 2, but speed is more critical
 - Speed referring to both detection and adaptation



Overall Takeaways

- Different types of operations can lead to different bottlenecks
 - Separation of work into tiers
- Fixing one bottleneck can move it to a different place!
- Simply adding more resources will not always help
 - Need to target specific issues
- Being able to identify a bottleneck and being able to fix it are distinct
 - Response time can be critical



Debugging

- View for logging help:
 - <https://stackoverflow.com/questions/15758685/how-to-write-logs-in-t-ext-file-when-using-java-util-logging-logger>
 - Comment out logs before submitting, they can slow you down
- Benchmark plotting different measurements can be very useful in identifying certain trends



Last Tips

- Start and **submit early**
 - There may be slowdowns in submission processing near the end of the due date, so don't expect to get efficient feedback from Autolab then
- The system is inherently non-deterministic
 - Consecutive runs will **not** always get the same score
 - This slight change can result in vastly different results
 - **We are not able to delete later submissions!**
- You can earn Bonus points for Checkpoint 2 and Checkpoint 3



Questions?