- A scalable automobile rental web service

Michael Zhang
Sammy Guo
Sujaya Maiyya
Kyle Carson
Justin Pearson

CS 291A: Scalable Internet Services
Prof Bryce Boe
Fall 2017
University of California, Santa Barbara

Outline

- App demo & details
- Tsung test setup
- Optimizations
  - Horiz. & vertical scaling
  - Pagination & Caching
  - Concurrent Nginx connections

Motivation

- Sharing economy is efficient, environment-friendly and accessible to all.

- A Uber or Lyft ride is not sufficient for all travelling demand, in case of family trip, long journey or private event.

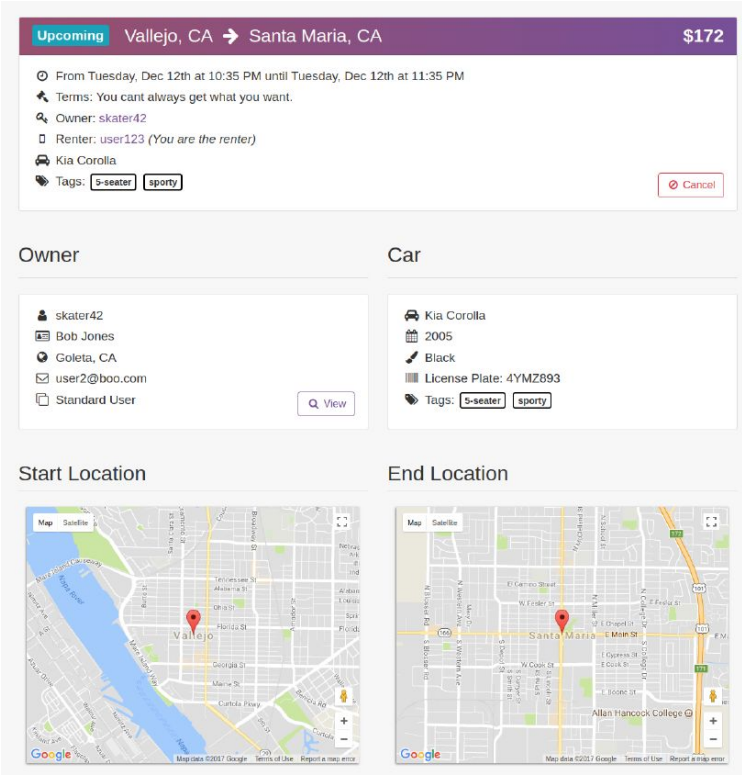
- We are proposing a Uber-Lyft-like long-term car-sharing app.

- Cheaper option for less-populated area

Functionality

- Car owners add their cars with make, model, color, year and tags.
- Car owners set parameters to renting their car,
    - start and end times
    - start and end locations
    - any additional terms they see fit

- Car renters browse rentals with details, such as owner info, car info, time duration and geo-location on Google maps.
- Car renters rent cars and monitor their progress



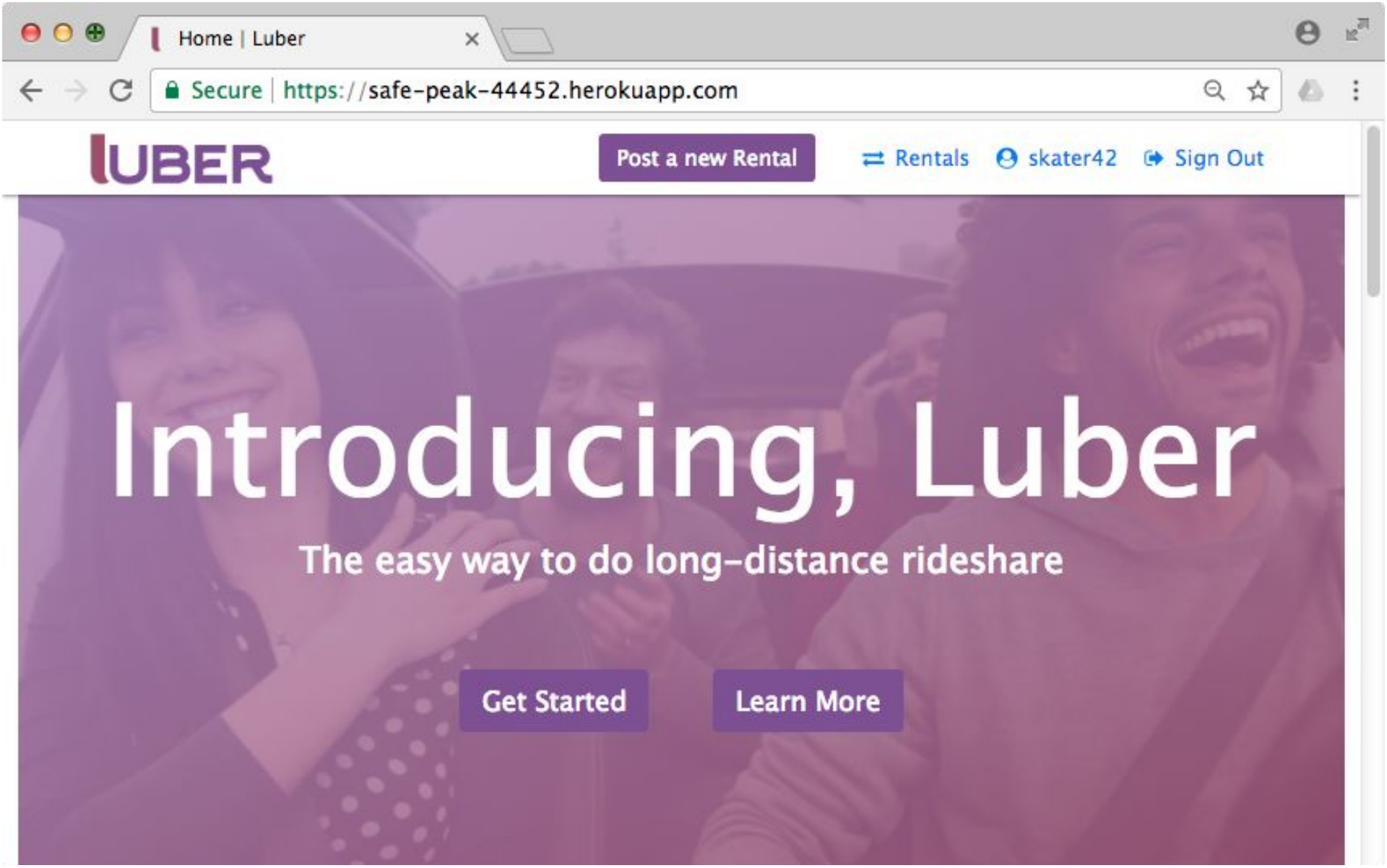
Implementation

- **Framework :** Ruby on Rails
- **Database :** sqlite3 in dev and postgreSQL in production
- **Gems :** bcrypt, will_paginate, geocoder, byebug
- **Server :** AWS Elastic Beanstalk
- **Continuous Integration :** Travis
- **Load testing :** Tsung

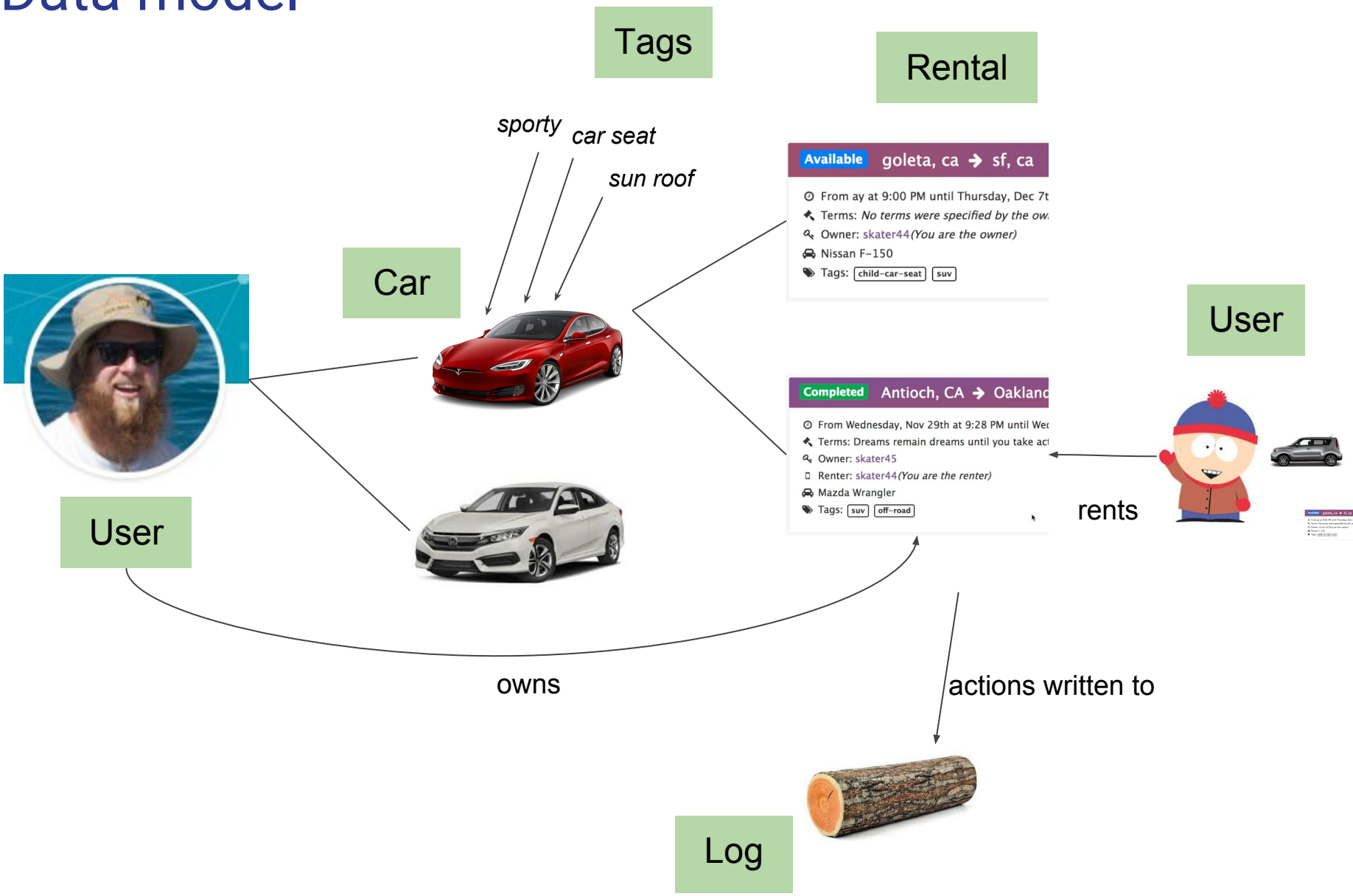
App demo

https://safe-peak-44452.herokuapp.com/

http://luber.fun -- coming soon



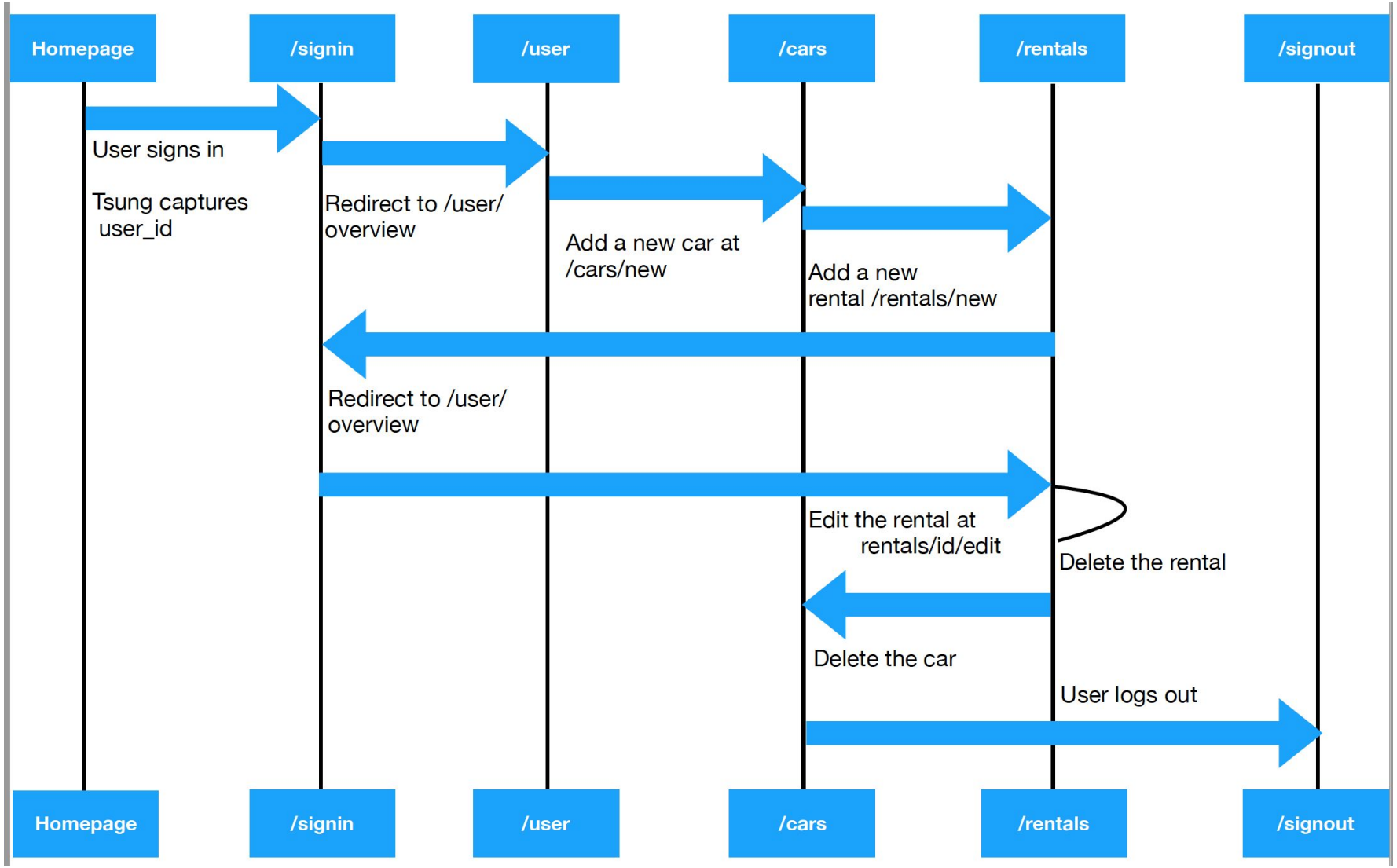
Data model


Tags
Rental
sporty
car seat
sun roof
Available goleta, ca → sf, ca
From ay at 9:00 PM until Thursday, Dec 7t
Terms: No terms were specified by the ow
Owner: skater44 (You are the owner)
Nissan F-150
Tags: child-car-seat suv
Car
User
Completed Antioch, CA → Oaklanc
From Wednesday, Nov 29th at 9:28 PM until We
Terms: Dreams remain dreams until you take ac
Owner: skater45
Renter: skater44 (You are the renter)
Mazda Wrangler
Tags: suv off-road
User
rents
owns
actions written to
Log

Outline

- App demo & details
-  Tsung test setup
- Optimizations
  - Horiz. & vertical scaling
  - Pagination & Caching
  - Conncurrent Nginx connections

Tsung tests: Workflow of a “Typical User”



Tsung tests: Phases

Exponentially increase “new users spawned per sec”

Sessions don’t overlap

```
<load>
  <arrivalphase phase="1" duration="60" unit="second" wait_all_sessions_end="true">
    <users interarrival="1" unit="second"/>
  </arrivalphase>
  <arrivalphase phase="2" duration="60" unit="second" wait_all_sessions_end="true">
    <users arrivalrate="2" unit="second"/>
  </arrivalphase>
  <arrivalphase phase="3" duration="60" unit="second" wait_all_sessions_end="true">
    <users arrivalrate="4" unit="second"/>
  </arrivalphase>
  <arrivalphase phase="4" duration="60" unit="second" wait_all_sessions_end="true">
    <users arrivalrate="8" unit="second"/>
  </arrivalphase>
  <arrivalphase phase="5" duration="60" unit="second" wait_all_sessions_end="true">
    <users arrivalrate="16" unit="second"/>
  </arrivalphase>
  <arrivalphase phase="6" duration="60" unit="second" wait_all_sessions_end="true">
    <users arrivalrate="32" unit="second"/>
  </arrivalphase>
  <arrivalphase phase="7" duration="60" unit="second" wait_all_sessions_end="true">
    <users arrivalrate="64" unit="second"/>
  </arrivalphase>
  <arrivalphase phase="8" duration="60" unit="second" wait_all_sessions_end="true">
    <users arrivalrate="128" unit="second"/>
  </arrivalphase>
  <arrivalphase phase="9" duration="60" unit="second" wait_all_sessions_end="true">
    <users arrivalrate="256" unit="second"/>
  </arrivalphase>
</load>
```

Tsung tests: Sessions

Idempotent & each user acts in isolation => avoids concurrency problems

```
<sessions>
  <session name="main_session" probability="100" type="ts_http">

    <transaction name="SIGN_IN">...
    </transaction>

    <transaction name="ADD_CAR"> ...
    </transaction>

    <transaction name="ADD_RENTAL"> ...
    </transaction>

    <transaction name="EDIT_RENTAL"> ...
    </transaction>

    <transaction name="DELETE_RENTAL"> ...
    </transaction>

    <transaction name="DELETE_CAR"> ...
    </transaction>
  </session>
</sessions>
```

Tsung tests: Transaction

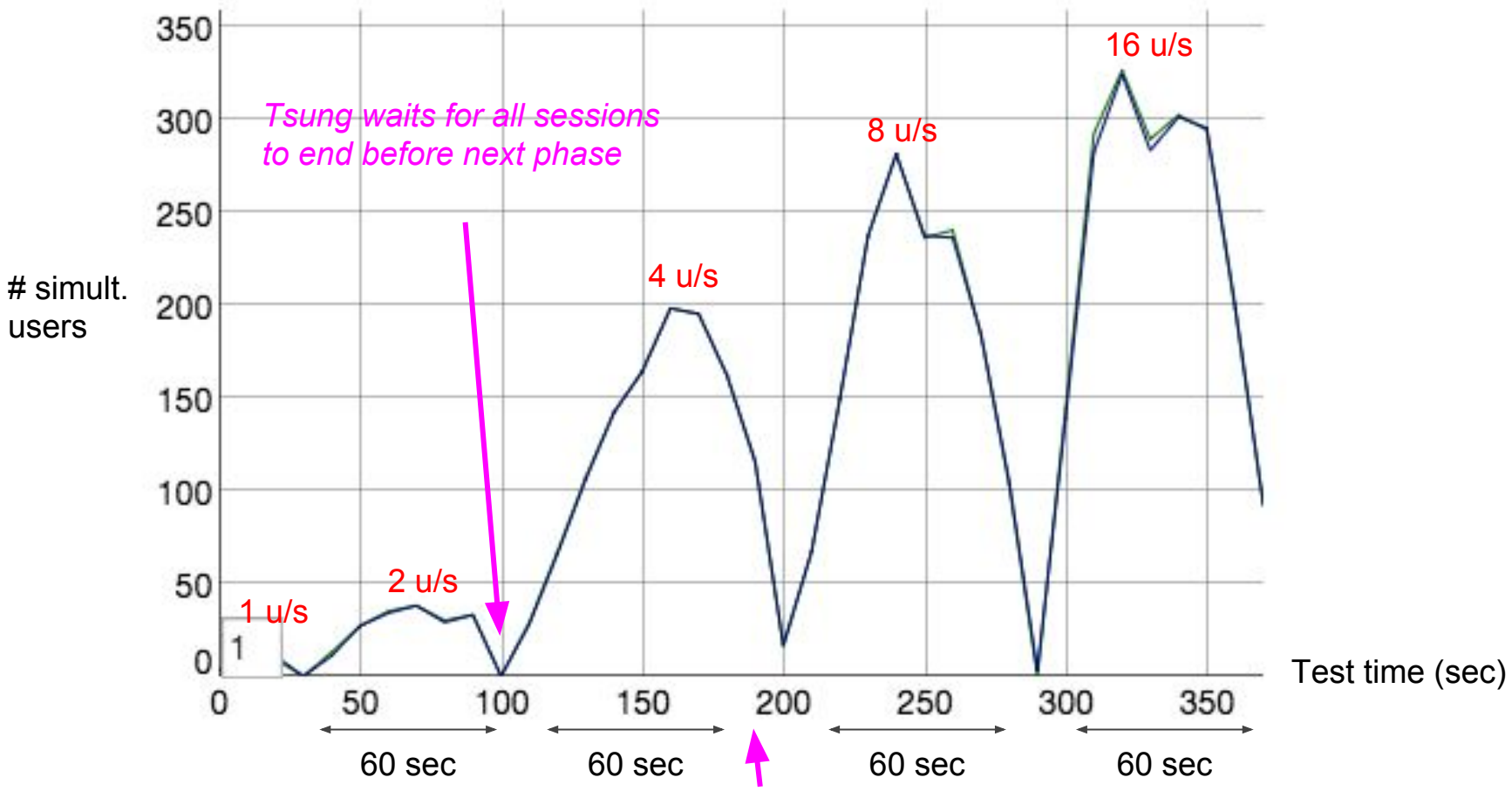
Users selected from CSV file

```
<transaction name="ADD_CAR">
  <request subst="true">
    <http url="/users/skater4%%_random_user_number%%/cars" method="GET" version="1.1"/>
  </request>
  <request><http url="/cars/new" method="GET" version="1.1"/></request>
  <thinktime value="1" random="false"/>
  <request subst="true">
    <dyn_variable name="car_redirect" re="[Ll]ocation: (.*)\r"/>
    <http
      url="/cars"
      method="POST"
      version="1.1"
      contents="car[make]=Ford%%_random_user_number%%
        &amp;car[model]=Mustang
        &amp;car[year]=2017
        &amp;car[color]=Red
        &amp;car[license_plate]=1234567
        &amp;car[all_tags]=by-User%%_random_user_number%%
        &amp;commit=Add+Car"
      content_type="application/x-www-form-urlencoded"/>
  </request>
  <request subst="true">
    <dyn_variable name="car_id" re="href='/cars/(.*)/edit'"/>
    <http url="%%_car_redirect%%" method="GET" version="1.1"/>
  </request>
</transaction>
```

Posting redirects; capture the redirect URL from the HTTP header

Follow the redirect, then get the id of the first editable car in the resulting HTML

Tsung tests: simultaneous users

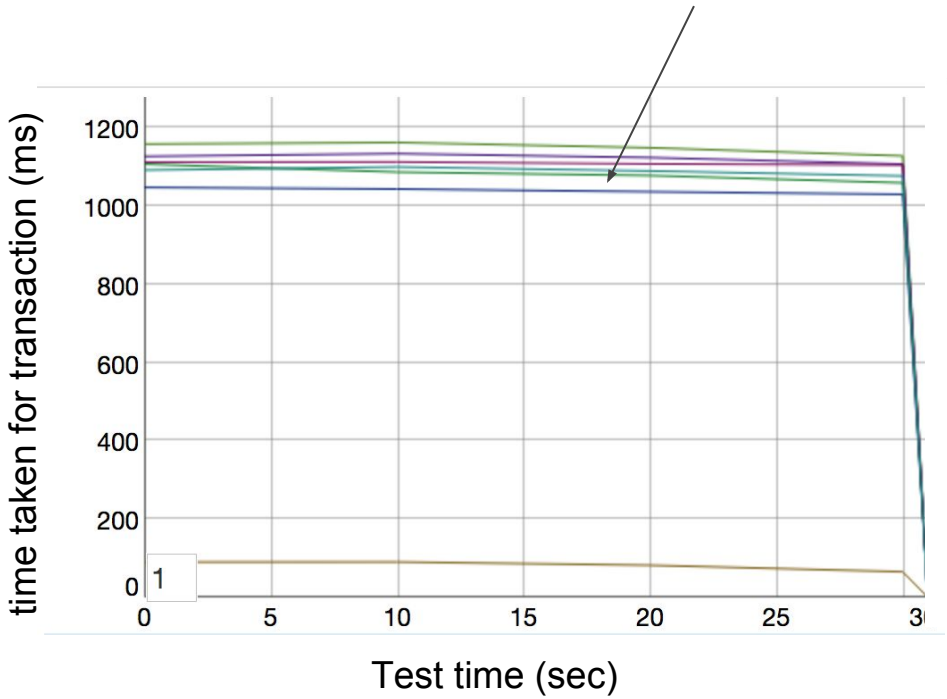


Theory: 60-sec phase + trailing session takes 6-8 sec
=> humps should be max 70 sec wide
This graph: 100 sec wide?
=> long server resp times / errors (4xx's & 5xx's)
=> this particular hw configuration cannot support 4usr/sec

Tsung tests: transaction time

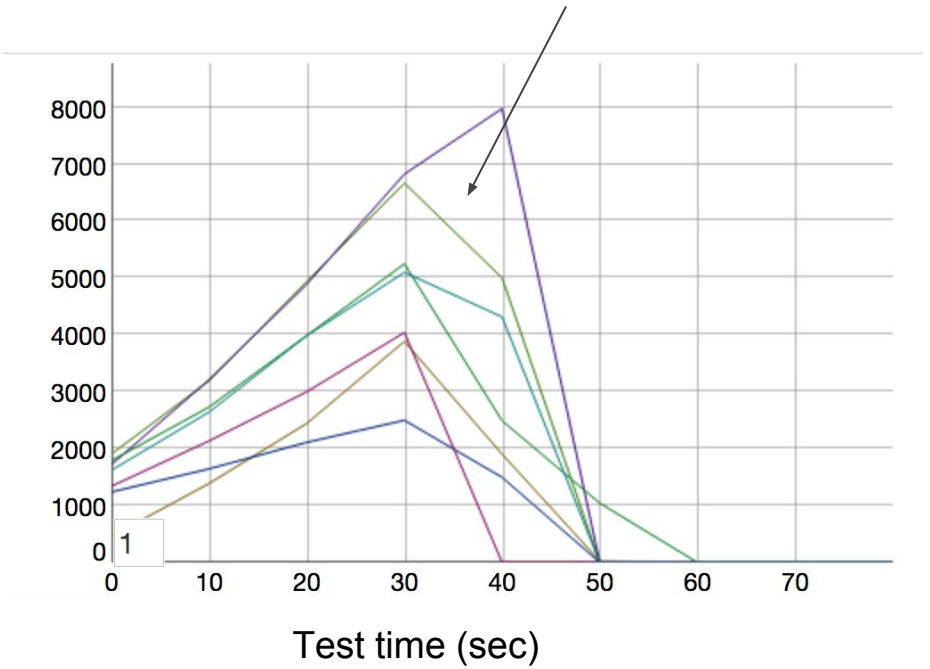
Good

Each tx has 1-sec think-time
=> 10-200ms "actual" waiting



Bad

2-8 sec for page load



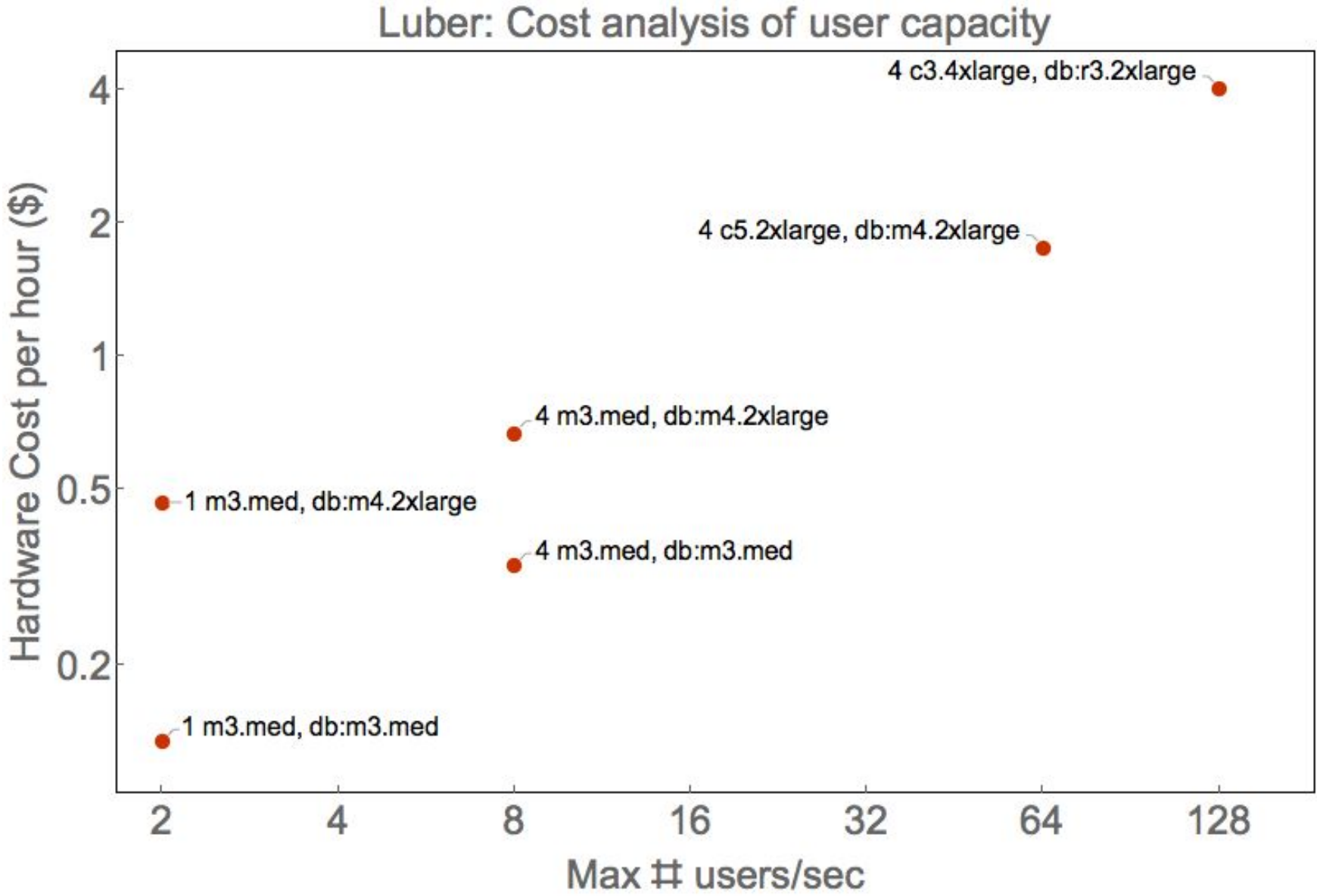
Outline

- App demo & details
- Tsung test setup
-  Optimizations
  - Horiz. & vertical scaling
  - Pagination & Caching
  - Concurrent Nginx connections

Horiz. & Vertical Scaling

Instance type	Num. instances	DB instance type
m3.medium	1	m3.medium
m3.medium	1	m4.2xlarge
m3.medium	4	m3.medium
m3.medium	4	m4.2xlarge
c5.2xlarge	4	m4.2xlarge
c3.4xlarge	1	r3.2xlarge
c3.4xlarge	2	r3.2xlarge
c3.4xlarge	4	r3.2xlarge

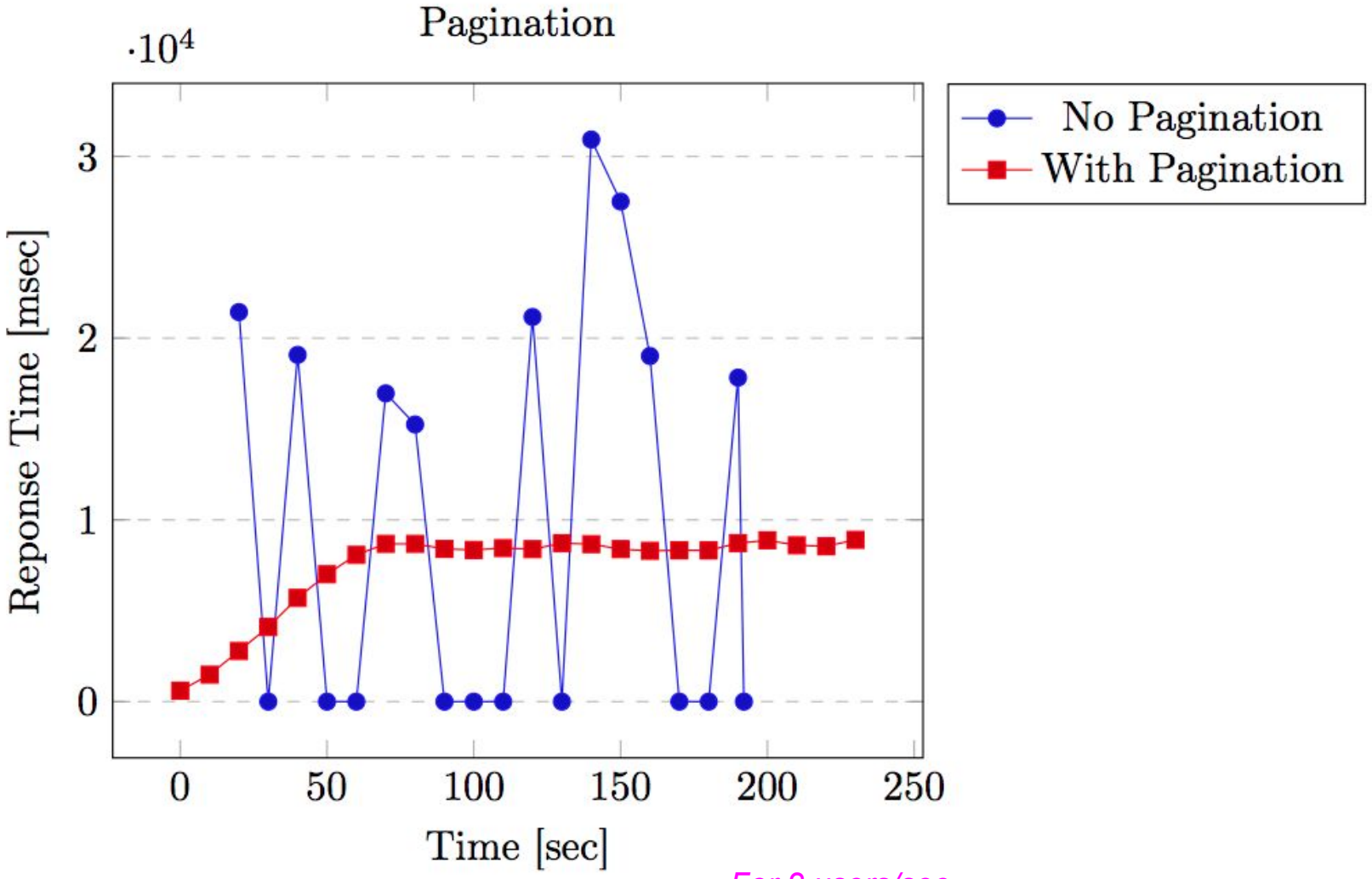
Cost analysis



(max user rate s.t. no 4xx or 5xx http codes in tsung.log)

Pagination

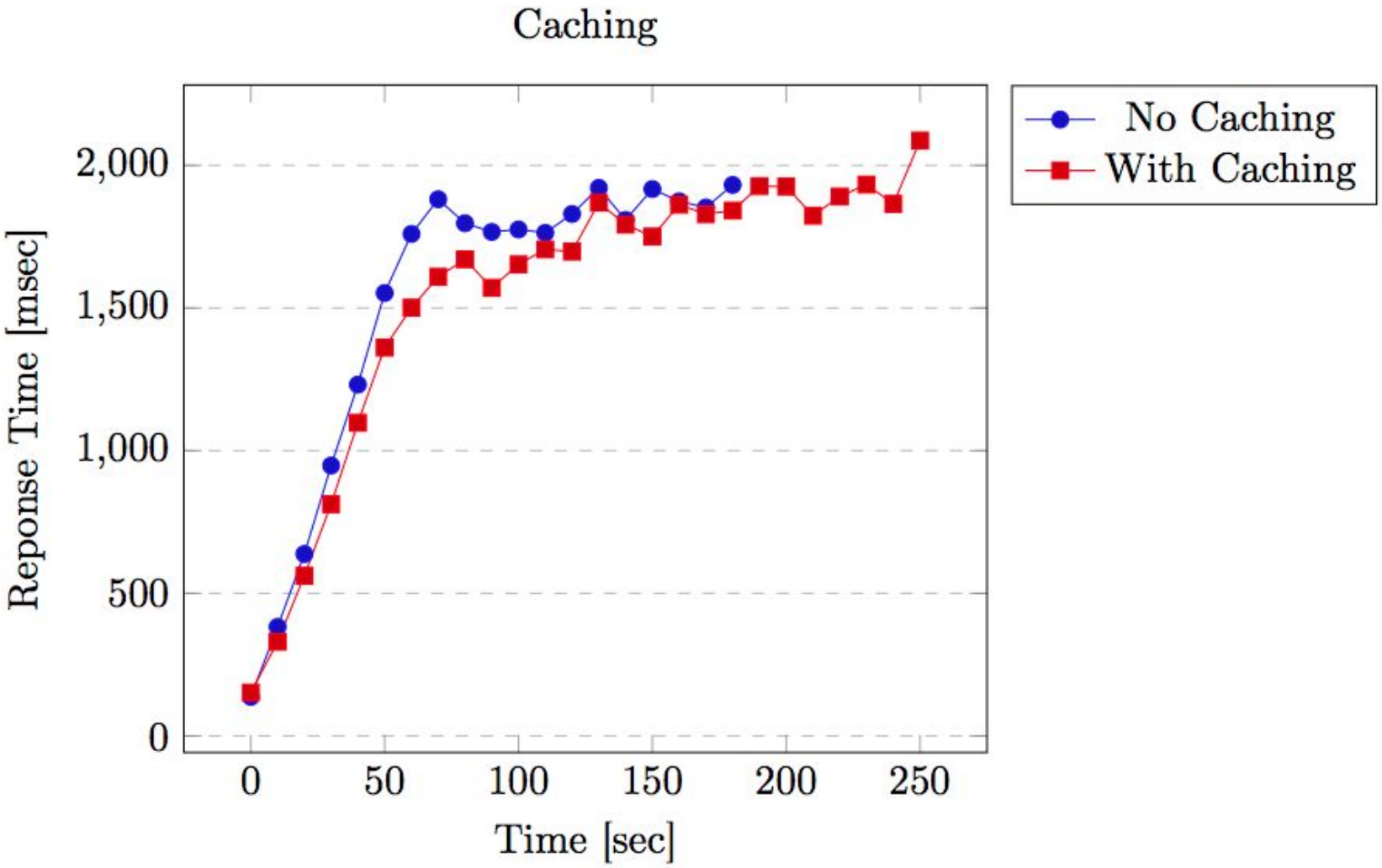
Reduced 4xx/5xx server responses & page response times.



For 2 users/sec

Caching

- Russian-doll caching on views
- Only slight improvement; perhaps views not the bottleneck.
- Should’ve cached db queries also

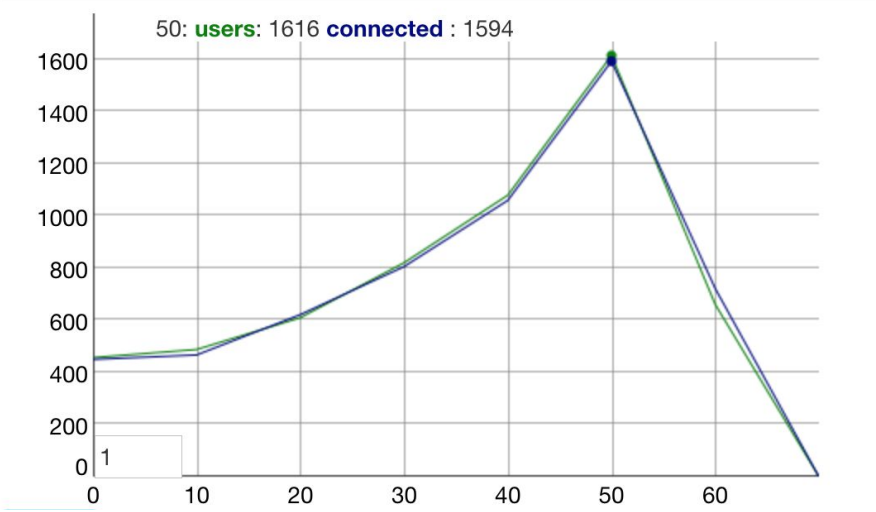


Concurrent connections

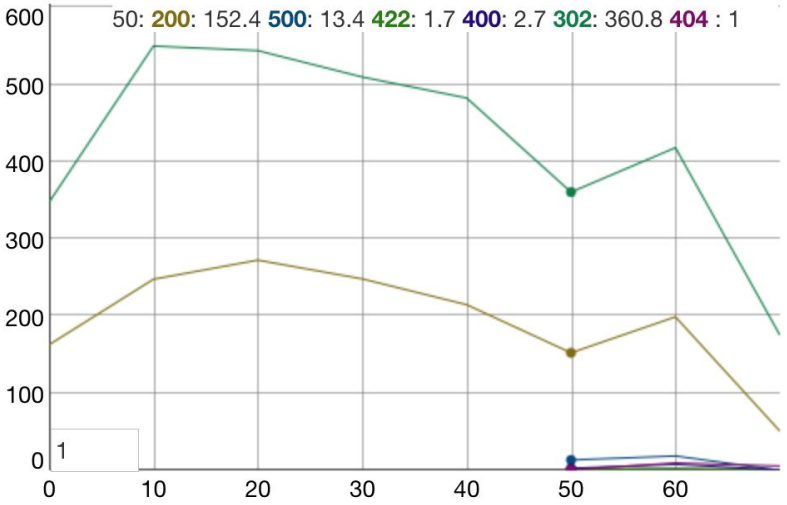
- AWS Elastic Beanstalk instances use Nginx web servers
- Web servers can be one of the biggest bottlenecks for scaling an app

Concurrent connections

Simultaneous Users



HTTP return code Status (rate)



Errors

Name	Highest Rate	Total number
error_connection_closed	4.2 / sec	64

Concurrent connections: Solutions

- Configure customized environment from project source by using `.ebextensions`
- Created a various configuration files in the `.ebextensions` directory and ran redeployed eb instances
- Manually logged into the instances, changed `/etc/nginx/nginx.conf` file

No method worked!

- Did some Network Tier optimization (connection draining, stickiness, health check..)

Is it a good idea to use third party services for your application?

Questions?