GOAL TRAC.KR

Let's do this together

A scalable and highly available web application to track your goals along with your friends

UCSB CS290 B – FALL 2013

NAZLI DERELI
DIBYENDU NATH
ARVIND CR
DIVYA SAMBASIVAN

Introduction

1 GOAL TRAC.KR

Have you ever wanted to learn surfing?
Do you really want to finish reading that book?
Is it impossible to lose some pounds?

Just create a goal and find some friends to join!
Give and get suggestions, cheer on and get motivated to complete your ever-haunting goals!

2 FEATURES

Facebook Integration - Users can login through their application using their Facebook accounts; they do not have to create "yet another account". All their friends on Facebook, currently using Goal Trac.kr are instantly accessible to the users.

Search – Users can use the "Goal Search" feature to look for any goals that may already exist. If they find such a goal, they will be able to view the current followers of the goals, which may include their friends and join the goal.

Create new Goal – If the users are not able to find a Goal of their choice, they may create their own goal and add their favorite pictures to keep it fun and motivating. Once a user creates a goal she becomes the owner of that goal and is free to edit or delete it, only if no one else is currently following that goal.

Milestones – To help track their goal, users may create upto three milestones for each goal and specify a weight to each of them. As they complete each goal, the user may check-off the milestone as complete. The goal is marked as complete when all the milestones associated with that goal have been completed.

Cheer Ons - An important feature of our application is the ability of the user's friends to cheer them on in their goals. This helps to keep users motivated to complete their goals.

Charts – Charts enable the users to easily visualize their progress. A time-based chart shows them how much time they have to complete a particular goal. Another milestone-based chart shows the progress of their goal in terms of milestone completion.

Design

1 UI DESIGN

Since our application is intended for a very generic user set, we have taken efforts to keep the UI simple and intuitive. The thumbnails, pagination and graphs further enhance our user experience, keeping our

users engaged. We have paid attention to the colors and theming and ensured that all the colors on the page gel together and make it visually appeasing.

2 APPLICATION DESIGN

Scalability - Our application is multi-user and transaction based and hence has to be able to scale very easily. As our application becomes more popular, we have designed it to scale out well. The simple and indexed database queries, optimized joins and caching of general actions enhance scalability. Our results section explains this in greater detail.

Availability – As availability is directly tied to user satisfaction and our reputation, we have taken measures to minimize our application downtime. Our application is load balanced and runs on multiple application instances; currently 8. This ensures that even if some of our app-servers go down, our users should not be able to notice too much of a slack.

We have also used Amazon RDS to have our database in multiple availability zones and a read slave database that services read requests even when our primary database is down.

Apart from providing good availability the above two features also enable us to do hot deployments to facilitate seamless upgrades to our application.

3 TOOLS AND TECHNOLOGY

Ruby on Rails - Our entire application runs on "Ruby on Rails". The "convention over configuration" approach has a learning curve but once mastered, ensures rapid and clean development. Most of the configuration and heavy-lifting is taken care of by rails, leaving only the implementation logic to the application designer. The plethora of readily available gems and abundant tutorials for commonly used tasks makes application development quick and hassle free.

Amazon EC2 – Our entire application runs on the Amazon EC2 infrastructure. The speed with which we could get our application up and running was truly amazing. Using EC2 also facilitates availability and scalability by taking advantage of multiple availability zones, load balancing and database scalability features offered by EC2, with minimal effort from our side. We have extensively used the Load Balancing, S3 storage and RDS features provided by EC2.

Twitter Bootstrap – We used twitter bootstrap to style our UI. True to its name, twitter bootstrap got us up and running really quickly (http://getbootstrap.com/). . Initially we faced a lot of version conflicts, due to supported/unsupported features in versions 2 and 3. However, Bootstrap proved to be an easy enough framework to learn and use. It supports the Agile Development process quite naturally. The ready-to-use widgets and styles greatly helped us develop our modules quickly and consistently. However, even though the widgets offered coverage in terms of the most commonly used UI elements, we were sometimes limited by their availability.

GCharts – Googlecharts (http://googlecharts.rubyforge.org/) is a simple ruby wrapper for the Google Charts API which supports different types of graphs line, scatter, bar, venn and pie charts amongst others. We use the pie chart to show the percentage of goal completion.

Optimization

1 Query Optimization

Initially, we had done naive querying on the database, like doing multiple selects on a condition like column_name = value. We optimized it by making a single call to the database as a select statement with column_name IN value_set in place of multiple database data fetch queries. This increased our performance by nearly 100-400 %.

2 Database Indexes

Although, we had indexed our database initially, we were surprised by the performance gain the strategic and well-placed indices gave us. The results section outlines the performance gains.

3 Caching

We use file store caching. We use action caching for caching rails actions based on parameters passed. For cache expiry of views associated with models, we use cache sweepers which observe the models for save, update or destroy calls and based on it we delete the cache using expire_fragment and doing a regex search on the cache. For views like friends which is not based on any model, we do expiry based on last time the cache was created.

Scaling

1 DATA

In order to ensure that our application scales linearly, we had to test it with a large dataset.

Users (~14000)

We use facebook login exclusively for accessing our application. As a result, we faced some issues regarding creating test data, especially test users for our application. What we ended up doing finally, is finding the range of facebook ids by going over the ids of us and our friends. Then we mined facebook ids in between the minimum and maximum ranges by using Ruby-based Facebook API called Koala to fetch valid Facebook ids in batches so as to avoid blacklisting by Facebook.

Goals(~5000)

We seeded our goals database with goals from different facebook pages and websites to create a repository of around 5000 goals. We used the Facebook Graph API, using the Koala API's to retrieve data from Facebook pages and the Nokogiri gem to scrape data from html pages.

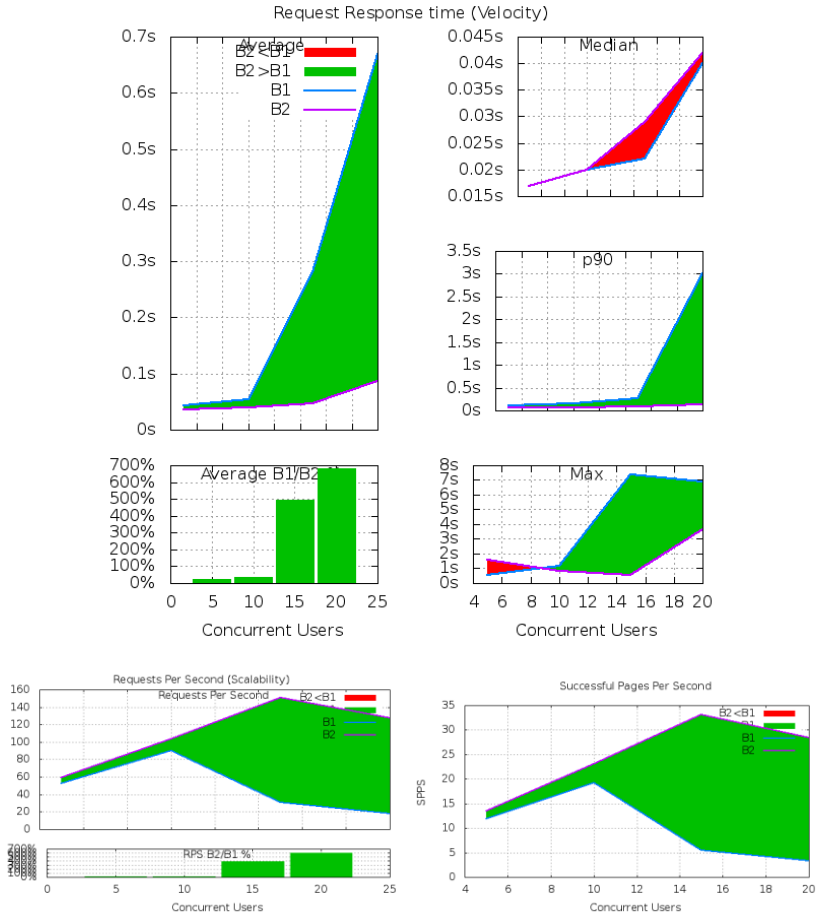
Goal Instance (~100500)

For creating goal instance test data like number of followers, number of cheer-ons, start date, end date, number of milestones, date of completion for milestones, etc. we used the ruby random number generator with proper limits to create a well distributed uniform data set.

2 RESULTS

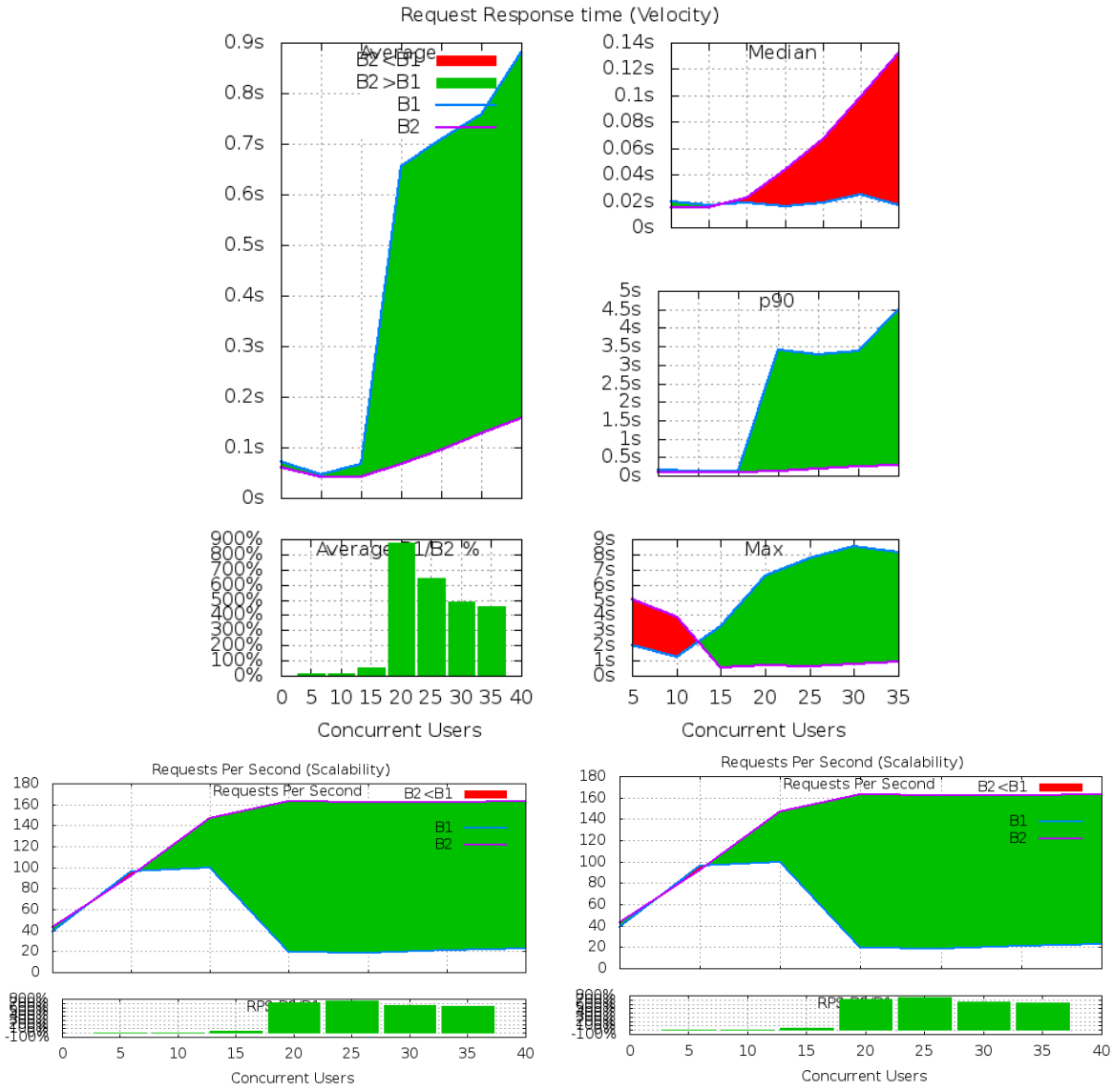
The following results show the increase in the response time and the number of successful pages when the number of servers is doubled, for different pages. We have only shown a few graphs, the ones that show the most pronounced increases in performance and scalability. We used ***httperf*** and ***funkload*** to carry out our tests on scalability and availability.

2.1 GOAL INSTANCE DETAILS PAGE



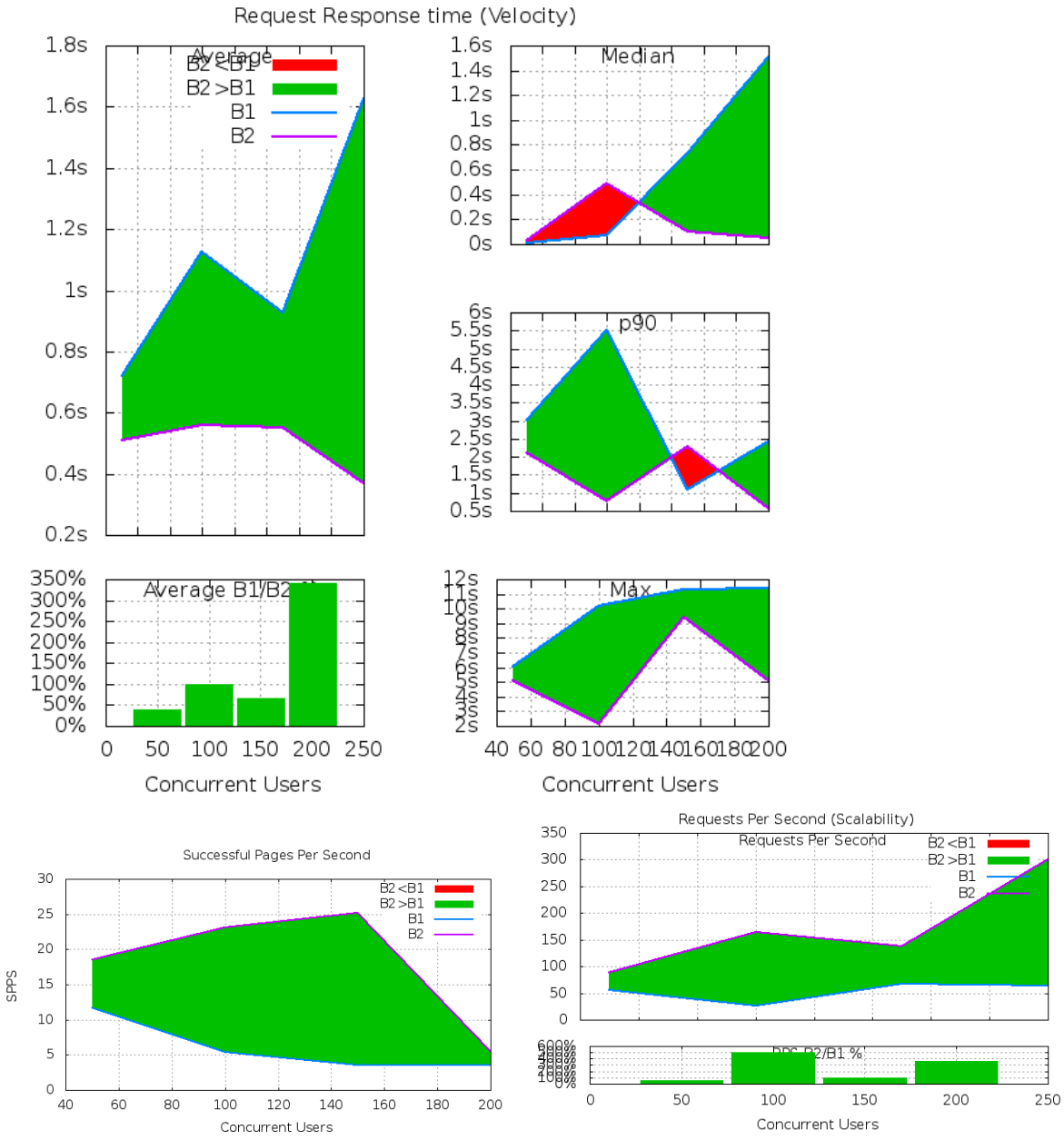
2.2 CRITICAL PATH –1

The following graphs show the increase in scalability when the user logins in, views her goal instances, views the details of one particular goal instance that she is interested in and logs out.



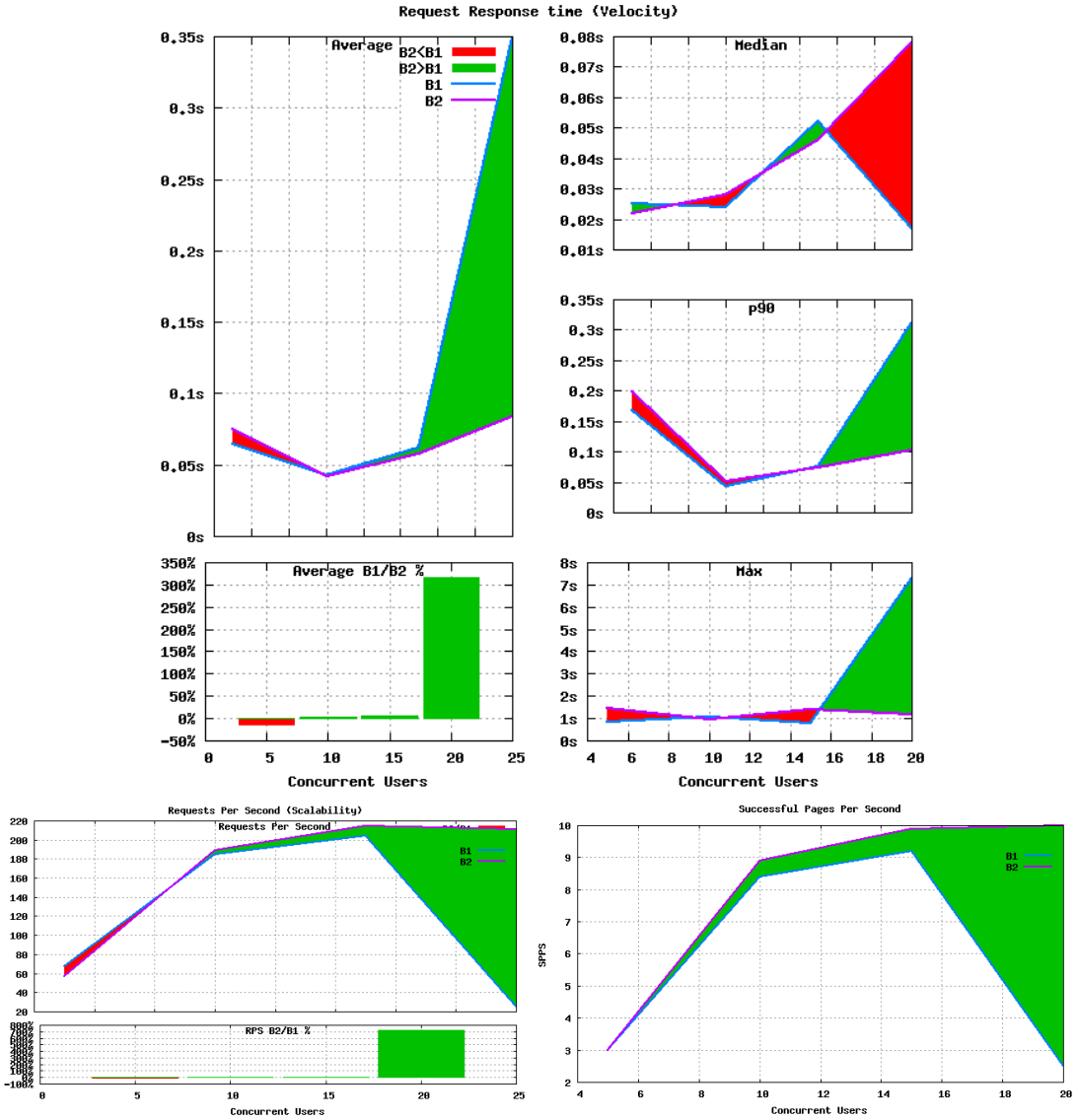
2.3 CRITICAL PATH – 2

The following graphs show the increase in the scalability when the user logs in, creates a new goal, joins the goal and logs out.



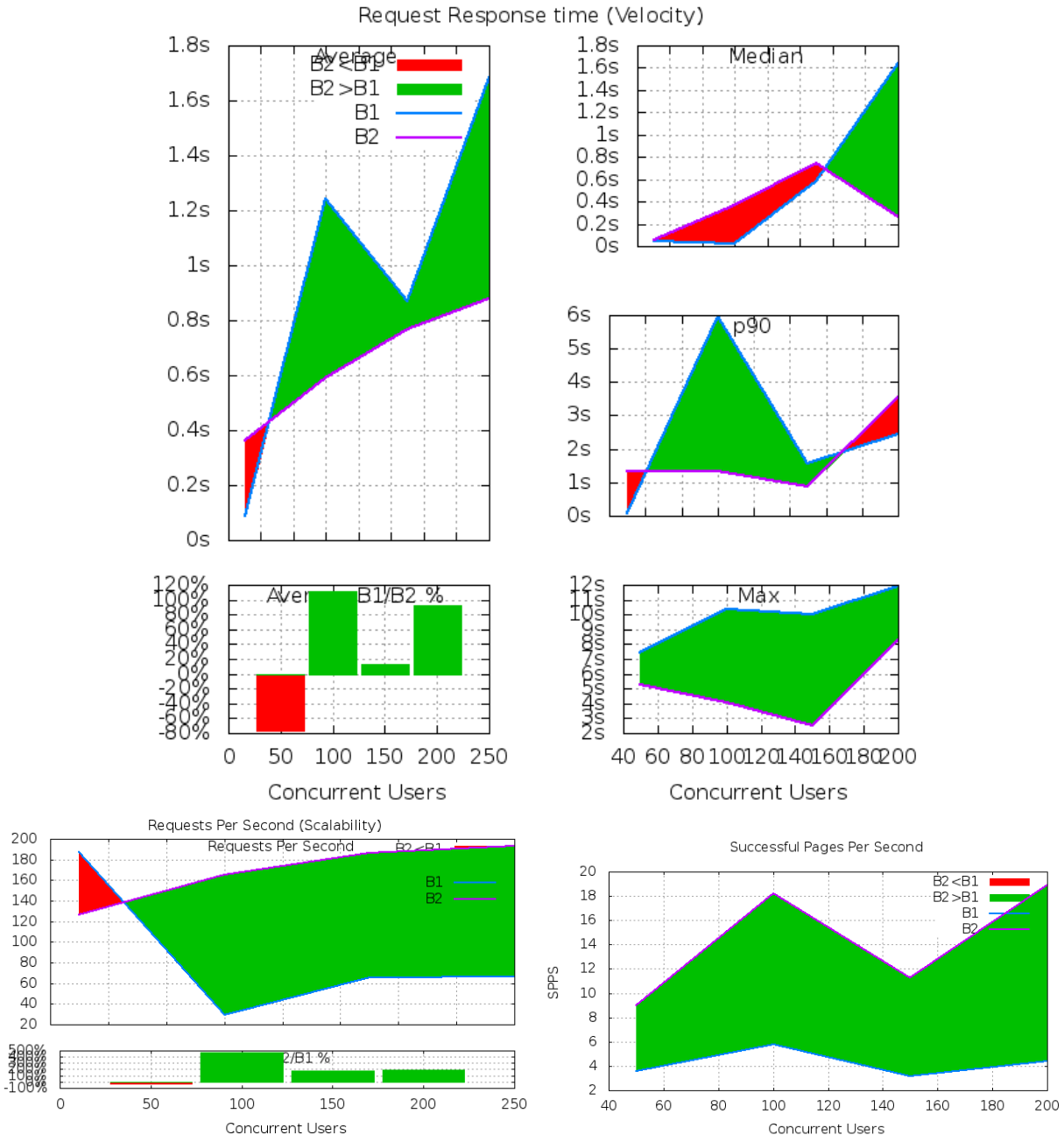
2.4 GOAL DETAILS PAGE – QUERY OPTIMIZATION

The number of concurrent users increases drastically after query optimization



2.5 FRIENDS PAGE –CACHING

The friends of a particular user are retrieved using calls to the Facebook Graph API. On caching the friends' page we see that we get an immense increase in the number of concurrent users. The cache is invalidated every half an hour.



Challenges

We faced a few technical challenges through the course of developing our application. We have outlined some of them here along with details of how we addressed those challenges.

1 Rails

As rails is a rapidly developing area, there are huge differences between two successive versions of rails. For example the earlier versions of 3.2.x (until 3.2.14) do not support the asset pipeline. We learnt this the hard way by starting off with 3.2.0 and later upgrading to rails 3.2.14

2 Twitter Bootstrap

We had similar issues with Twitter Bootstrap as we started off using version 2 and we tried upgrading to version 3, we faced plenty of breakages due to difference in class names etc.

3 Facebook User Data

Obtaining a large number of users was essential to test the scalability of our application. However, since were using Facebook logins, we did not have our own database of users, to generate test users. We overcame this by incrementally mining different Facebook ids and checking if they were valid Facebook users by using the Facebook Graph API.

4 Login

We are using Facebook OAuth2 to login. Facebook authentication works by redirecting the user to the Facebook login page and redirecting them back to our application after a successful login. This worked perfectly for our browser application. However, when we had to test it programmatically through tools like httperf and funkload, we did not have control over the facebook redirection. Hence, we wrote a dummy login route just for the programmatic performance testing. This login directly provided the facebook access token (through a url parameter) to overcome this limitation.

Future Enhancements

1 Features

Comments - We would like to include a “comments” section in our goals details page. This section could be used for friend to comment on a user’s goal instance and provide suggestions to help her achieve it faster and better.

Ranking – We would like to rank goals and friends, to dictate the order in which they are shown to the user respectively. The goals would be ranked based on popularity and friends could be ranked on how "close" they are based on Facebook data.

2 AVAILABILITY

Our application currently has app servers only in one geographic location. We would like to increase it to more than one geographic location using the Amazon Route 53 DNS Service to ensure the availability of our application even if one of the datacenter in one of the geographical regions is affected by unprecedented events.

Conclusion

We have developed a scalable and highly available web application as a part of the course 290B-Scalable Web Applications. We followed the Agile Methodology and we worked in sprints of one week. This made our application development extremely fast-paced and fun as we saw it take shape quickly. The Agile Methodology also allowed us to make changes easily, when required. Working in a team with people from different backgrounds was an enriching experience. During the course of this project we have picked up and incredible amount of knowledge in various tools, languages and programming practices and learnt to make our application scalable.