CMU 17-648 Engineering Data Intensive Scalable Systems

Project Task 4: Introduction to Scalability

Matthew Bass

Problem Description

As the volume and diversity of business continues to grows, so do the needs of the system. The system needs to be able to handle increasing volumes of data and increasing numbers of users. This project begins to look at the ability of the system to reliably handle increased number of users with larger volumes of data.

This project is focused on exploring the impact of increased demand. You will look at what impact an increased demand has on your system, and explore options for minimizing the impact. This project is designed to be an introduction to scalability, and focuses on basic fundamental decisions that can increase the capacity of the system to manage larger volumes of usage. Later in the course we will expand on this knowledge and look at some of the more complex concerns that emerge.

Learning Objectives

This project will allow you to become more familiar with basic techniques for handling an increased volume demand. Additionally, you will learn how to instrument your system to gain insight into how various aspects of your system is performing.

We will continue to work with the technologies that we have been working with in the last couple of projects.

These technologies include:
- Javascript
- Node JS
- Express
- Database of your choosing (eg):
  - MySQL
  - Mongo
  - Dynamo
- Artillery
- Amazon Web Services

You might need to evolve your thinking about the design of your system. You may find that some of the previous decisions limit the options available to you or adversely impact one or more desirable properties. When this occurs you should make mental note of the decision and its impact. This is a learning opportunity that you face repeatedly in the course and will want to explicitly recognize as you'll surely have similar choices in your professional life.

Tasks

In this project you will continue to build a simple e-commerce system and deploy it on AWS. In addition to building a system to support the requisite functionality you'll need to **manage the compute costs for your system**. The functionality will be available via web services. The functionality should be consistent with the specifications provided. You can assume that the user will reside in the USA (you don't need to worry about international addresses).

You will be provided with a data set containing multiple millions of products. Depending on how you implemented the previous system you will likely find that certain functions (e.g. modify and view products) are slowing down.

As the demands on your system increase you might find that the performance further degrades. If this is the case you might need to redesign your system to be able to scale to support the larger demand and larger volume of product data without the performance degrading (for any of the functionality). You can try to optimize your solution but while this might give you short term gains, there will be real limits to how much of a benefit this will realize.

The specification is the same as it's been in previous projects.

The requirements for the system are:

- **Name**: Register Users (does not require authentication):
  - **Description:** This allows new users to create an account. The result of this will be that customer information is in the system and the user will now have an account that will allow them to log on as a customer.
  - **Interface:** {BASEURI}/registerUser
  - **HTTP Method**: Post
  - **Input parameters**:
    - {"**fname**": "*first name", "***lname**": "*last name",* **"address"**: "*street address",* "**city**": "*city*", "**state**": "*2 letter state code or complete state name*", "**zip**": "*can be numbers or letters*", "**email**": "*email*", "**username**":"*Username – must be unique",* "**password**": "*Password"*}
  - **Return Values**:
    - ***message***:
    - Success Case
      - {"**message**":"*first name* was registered successfully"}
    - Illegal Input
      - {"**message**":"The input you provided is not valid"}
      - The system should not allow duplicate registrations – this is defined as a non-unique username. For this system the username needs to be unique
      - All of the fields are required. The registration should fail if any of the fields are left blank
- **Name:** Login
  - **Description:** Allows users to log into the system. This should create a session that will remain active until the user logs out or remains idle for 15 minutes. The user should be able to access any functionality that requires the user is authorized to use. You need to allow for someone to log in from a system with an active session. If there's an active session and there is a new log in (from the same system) you should end the previous session and initiate a new one. There could be multiple active sessions from multiple systems.
  - **Interface:** {BASEURI}/login
  - **HTTP Method:** Post
  - **Input parameters:**
    - { "**username**":"*Username of the person attempting to login",* "**password**": "*Password of the person attempting to login"* }
  - **Return Values:**
    - Successful login – *first name* = name of the person that has logged in
      - { "**message**":"Welcome *first name*"}
    - Failure case
      - {"**message**":"There seems to be an issue with the username/password combination that you entered"}
- **Name:** Logout
  - **Description:** If the user has an active session this method will end the session. If there is no session then there is no change of state.
  - **Interface:** {BASEURI}/logout
  - **HTTP Method:** Post
  - **Return values**
    - Successful login – *first name* = name of the person that has logged in
      - {"**message**":"You have been successfully logged out"}

        - Failure case
            - {“**message**”:“You are not currently logged in”}

- **Name:** Update Contact Information
    - **Description:** This method allows the user to update their own contact information
    - **Preconditions:** User is a registered user and is logged into system. The parameters are optional and only updated if provided.
    - **Interface:** {BASEURI}/updateInfo
    - **HTTP Method:** Post
    - **Input parameters:**
        - {“**fname**”: “*first name”,* **“lname”**: “*last name”,* **“address”**: “*street address”,* **“city”**: “*city”*, “**state**”: “*2 letter state code or complete state name*”, “**zip**”: “*can be numbers or letters*”, “**email**”: “*email*”, “**username**”:“*Username – must be unique”,* “**password**”: “*Password”*}
    - **Return values:**
        - Success Case
            - {“**message**”:“*first name* your information was successfully updated”}
        - Not Logged In
            - {“**message**”:“You are not currently logged in”}
        - Illegal Input
            - {“**message**”:“The input you provided is not valid”}
- **Name**: Add Products (must be logged in as an admin):
    - **Description:** This allows admin to add a product to the system. The result of this will be that the product is now in the system. All parameters are required.
    - **Interface:** {BASEURI}/addProducts
    - **HTTP Method**: Post
    - **Input parameters**:
        - {“**asin**”: “*a unique id that can be used to access this product and associate with related information (such as reviews)*”, “**productName**”: “*the name of the product*”, “**productDescription**”: “*a description of the product*”, “**group**”, “*the group(s) the product belongs to, examples below*”}
        - Example groups are (you should allow any group provided):
            - Book
            - DVD
            - Music
            - Electronics
            - Home
            - Beauty
            - Toys
            - Clothing
            - Sports
            - Automotive
            - Handmade
    - **Return Values**:
        - Success Case
            - {“**message**”:“*productName* was successfully added to the system”}
        - Not Logged In
            - {“**message**”:“You are not currently logged in”}
        - Not admin
            - {“**message**”:“You must be an admin to perform this action”}
        - Illegal Input
            - {“**message**”:“The input you provided is not valid”}
                - The ASIN must be unique/or required fields not included
- **Name:** Modify Products (only an admin can modify a product):
    - **Description:** This method allows you to modify the description and name of the product. All parameters are required.

- **Preconditions:** The user is an admin and is logged into the system
- **Interface:** {BASEURI}/modifyProduct
- **HTTP Method:** Post
- **Input parameters:**
  - {“**asin**”: “*cannot be modified, only used to reference the product”,* “**productName**”: “*the name of the product*”, “**productDescription**”: “*a description of the product*”, “**group**”, “*the group(s) the product belongs to, examples below*”}
- **Return values:**
  - Success Case
    - {“**message**”:“ *productName* was successfully updated”}
  - Not Logged In
    - {“**message**”:“You are not currently logged in”}
  - Not admin
    - {“**message**”:“You must be an admin to perform this action”}
  - Illegal Input
    - {“**message**”:“The input you provided is not valid”}
      - The ASIN must be unique/or required fields not included

- **Name:** View Users
  - **Description:** This method allows the user to view all of the users registered in the system. . If the search criteria is blank the system will return all users.
  - **Preconditions:** The user is an admin and is logged into the system
  - **Interface:** {BASEURL}/viewUsers
  - **HTTP Method:** Post
  - **Input parameters (optional parameter to filter results):**
    - **{“fname”: “***some portion (or all) of the first name of the users you’d like to view*”, “**lname**”: “*some portion (or all) of the last name of the users you’d like to view*”}
  - **Return values:**
    - Success
      - {“**message**”:“The action was successful”, “**user**”:[{“**fname**”: “*first name*”, “**lname**”: “*last name*”, “**userId**”: “*a unique id for this user*”}, {“**fname**”: “*first name*”, “**lname**”: “*last name*”, “**userId**”: “*a unique id for this user*”}, …]}
    - No Users
      - **{“message”:** “There are no users that match that criteria”}
    - Not Logged In
      - {“**message**”:“You are not currently logged in”}
    - Not admin
      - {“**message**”:“You must be an admin to perform this action”}
- **Name:** View Products (does not require log in):
  - **Description:** This method returns products that meet the search criteria. If the search criteria is blank the system will return all products.
  - **Interface:** {BASEURI}/viewProducts
  - **HTTP Method:** Post
  - **Input parameters (optional)**
    - {“**asin**”: “*id of the product that you’re interested in viewing (if you have this)*, “**keyword**”: “*if you’d like you can search for products via a keyword. The product should be returned if the keyword is contained in the name or description of the product. The keyword needs to be an exact match, meaning the “I am here” should only return a match (case insensitive) for those three words in that order. Text that contains “I” and “am” an “here” in any other order (or with words in between) should not match.*”: “**group**”, “*only products in this category should be returned*”}
  - **Return values**
    - Success

            - **{“product”:[** {“**asin**”: “*The id of the product retrieved”,* “**productName**”: “*the name of the product*”}, {“**asin**”: “*The id of the product retrieved”,* “**productName**”: “*the name of the product*”}, …]}
        - No Products
            - **{“message”:** “There are no products that match that criteria”}

- **Name:** Purchase Products:
    - **Description:** This method allows a customer that is logged in to purchase a product or a set of products
    - **Interface:** {BASEURI}/buyProducts
    - **HTTP Method:** Post
    - **Input parameters**
        - **{“products”: [{“asin”:** “*asin”***}, {“asin”:** “*asin*”**}, …]}**
        - This is an array of asin numbers representing the products the customer is purchasing.
    - **Return values**
        - Success
            - {“**message**”:“The action was successful”}
        - Not Logged In
            - {“**message**”:“You are not currently logged in”}
        - No Products
            - **{“message”:** “There are no products that match that criteria”}
    - **State Change:**
        - The system should keep a record of the purchase including,
            - The items purchased
            - The customer that made the purchase
        - Also, update the products purchased together for recommendations
- **Name:** Products Purchased:
    - **Description:** This method allows admin that are logged in to get the history of products purchased by a given user.
    - **Interface:** {BASEURI}/productsPurchased
    - **HTTP Method:** Post
    - **Input parameters**
        - **{“username”: “***This is the username for the customer who’s purchase history you’d like to retrieve*”}
    - **Return values**
        - Success
            - {“**message**”:“The action was successful”, “**products”:**[{“**productName”:** “*product name”*, “**quantity**”: “*quantity purchased”*}, {“**productName”:** “*product name”*, “**quantity**”: “*quantity purchased”*}, …]}
        - No Users
            - **{“message”:** “There are no users that match that criteria”}
        - Not Logged In
            - {“**message**”:“You are not currently logged in”}
        - Not admin
            - {“**message**”:“You must be an admin to perform this action”}
- **Name:** Get Recommendations:
    - **Description:** This method returns a set of recommendations for a given product. The recommendations are based on other products that have been purchased together with the given product. If another product was bought together twice it will have a higher recommendation than a product that was bought together once. This should return the top 5 recommendations. If there were no products purchased together with the target product than there will be no recommendations. If there was only one product co-purchased there will be only one recommendation.
    - **Interface:** {BASEURI}/getRecommendations
    - **HTTP Method:** Post
    - **Input parameters**

        - {“**asin**”: “*id of the product that you want recommendations for”*}
    - **Return values**
        - ***Success:***
            - **{“message”:** “The action was successful*”,* **“products”: [{“asin”:** “*asin”***},** **{“asin”:** “*asin*”**}, …]}**
        - No Recommendations
            - **{“message”:** “There are no recommendations for that product*”*}
        - Not Logged In
            - {“**message**”:“You are not currently logged in”}
-

You will need to load the provided product data into your system (there should be 2.225 million records). A script is provided to assist with the data loading.

You should assume that multiple people will be accessing your system simultaneously and be sure that your system operates correctly when you have concurrent users. You are expected to use good programming practices with proper error handling. You will be given a set of users to load in your database prior to submitting your system.

You will build the application using Node JS and Express and can use the database of your choice. The system will be deployed and tested on AWS.

You’ll continue to have to handle the following faults:
- Node Instance Failure: If the instance that’s hosting your node server fails (either hardware or VM) the system should be able to mask this fault without noticeable delay to the user (response time should not be more than the average response time normally experienced) and without losing any data/requests.

Tagging: You’ll need to tag all of your instances in this project. The tags should be:
- Application server: AppServer
- Database (if you’re deploying a db directly on an EC2 instance): DB

Default administrator to have in your system at the time of testing:
- Jenny Admin
    - Username: jadmin
    - Password: admin

Technologies and Resources

You will be using Javascript, Node js, Express, MySQL, and Amazon Web Services (AWS) for this project. Links for general descriptions and resources are listed below. In the next section we will give detailed instructions on installing the technologies, getting started with AWS, and deploying an application to AWS.

General links:
- Javascript: http://www.w3schools.com/js/
- NodeJS: https://nodejs.org/
- Express: http://expressjs.com/
- Database of your choosing e.g.:
    - MySQL: https://www.mysql.com/
    - Mongo: https://www.mongodb.com/
    - Dynamo
- AWS: http://aws.amazon.com/

Getting Started

There are two data files provided. One has product related data (might be too large to open). A script is provided to assist with the data loading. You can load the data from your personal system (it will take about a ½ hour or so) of set up an instance on AWS. To run the script you'll need to install the dependencies (run: "npm install: in the directory with package.json). You'll need to change the IP address of the database and update the script if you're not using mysql (or if your schema is different than specified).

The other is for testing and has 5000 users (There shouldn't be any duplicate usernames in it). If you'd like to test your system over time you can increase the number of threads and ramp up time in the artillery script and it will read the data one row per iteration.

Submissions

The initial submission:

- You are to provide the logs from CloudWatch metrics that you've collected from your running system. The logs should contain measures for each element in your system (e.g. each instance, load balancer, database, …). You are responsible for defining the metrics of interest, but they should help you to evaluate the performance and diagnose issues associated with your system. Each service in AWS has a different set of metrics associated with it. Examples of metrics associated with RDS are: ReadLatency, ReadIOPS, WriteIOPS, and DatabaseConnections. You can find more information about CloudWatch at: http://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/WhatIsCloudWatch.html.

The final submission is a two-step process:

- *Run the auto grader:* *****Before you run the auto grader you need to ensure that the database is cleared (other than the product data and the admin user identified in the spec)***.** The next step (and the one that counts as the time you submitted the project) is to run the auto grader. In order to do this you'll need to have your system deployed and allow any IP to access the Node server that your application is running on. You will then go to the autograder and enter your IP address, port, and andrewID. The system will run the auto grader and give you an excel file with the results. That's all you need to do. You do not need to submit the excel file. This is for your own records. You will have 2 chances to submit to the autograder. **Please report any issues executing the auto grader to the Instructor and TAs immediately.**
- *Submit your code:* You will need to submit your code to project 4 in moodle.

Grading

Your grade will be based on three criteria:

- Average latency
- Compute cost (measured during the execution of the auto grader)
- Percentage of correct responses

For each category the submissions will be ranked. Points will be applied as follows:

- 20 points for top 1/3 per category
- 15 points for middle 1/3 per category
- 10 points for bottom 1/3 per category
- 40 points for turning in a working solution

➢ **You must have registered your account prior to starting this project in order to get credit**

Budget

The budget for this project is $5.00 (or resources that would total $5 without the free tier).