Summer Internship Project
Report

Implementing Connection Pooling/Caching on SNAP Gateway Service

Anjaneya Tripathi
106118010
National Institute of Technology, Tiruchirappalli

Under the guidance of

Priyank Rastogi
Director, Product Development
Data and AI Services

Data and AI Services
Oracle Cloud Infrastructure (OCI)
Bengaluru

Summer Internship 2021

Acknowledgment

The internship opportunity I had with Oracle Cloud was an amazing chance for me to learn and grow professionally as a developer and person. Therefore, I find myself an extremely fortunate individual as I was provided with an opportunity to be a part of this phenomenal organisation. I am also grateful for having a chance to meet so many wonderful people and professionals who led me though this internship period.

I would also like to use this opportunity to show my gratitude to my manager Priyank Rastogi who despite being extremely busy with his work, took time out to hear me, guide me, ensure I was on track and allow me to carry out my project at their esteemed organization.

I express my deepest thanks to Vivek Pathak and Ankur Jain for mentoring me through out the duration of my internship. They were instrumental in giving useful advice and guidance and arranged all facilities to make my 8 weeks pleasurable. I use this moment to acknowledge their contribution gratefully.

This opportunity is a big milestone in my life and career. I strive to use the skills and knowledge gained in the best possible way, and continue to work on the improvements suggested by my team, in order to attain desired career objectives. I wish to continue cooperating with all of you in the near future.

Sincerely,

Anjaneya Tripathi

Chennai, India

27th August, 2021

ORACLE

Oracle India Pvt. Ltd.
India Development Center

Oracle Technology Park
3, Bannerghatta Park
Bangaluru - 560 029

CIN: U74899DL1993PTC051764
Phone +91 80 4107 0000
Fax +91 80 2552 6124

Private & Confidential

Ref: Oracle-Interns- 1455577

26th Aug 2021

TO WHOMSOEVER IT MAY CONCERN

This is to certify that Mr. Anjaneya Tripathi , student of NIT Tiruchirapalli (Pursuing B.Tech (Computer Science and Engineering)). has completed his project with Oracle India Private Limited.

The project was undertaken from 02-Jun-2021 to 27-Jul-2021.He worked on "Implementing Connection Pooling/Caching on SNAP Gateway Service"

We wish him all the best in his future endeavours.

Yours sincerely

For Oracle India Private Limited.

Rambabu Jagatha
Director – HR Operations

Abstract

The Sparkline Gateway is a proxy service responsible for authenticating and authorizing users. In addition to this, it provides a Spark UI and JDBC endpoints on the same connection and allows for zero downtime upgrades. Spark provides an interface for programming clusters with implicit data parallelism and fault tolerance. Java Database Connectivity is an application programming interface for the programming language Java, which defines how a client can access databases. It is a Java-based data access technology used for Java database connectivity. Once authorized, the user is allowed to connect to the backend and execute queries. The gateway accommodates two types of requests - Spark UI requests and JDBC specific requests. However, at the moment the gateway cannot handle sizeable requests as the number of connections are limited. As a result, the number of users that it can cater to gets reduced. In order to tackle this issue, it was decided that reusing connections would be an optimal way forward. This would ensure that the connections are used in a manner in which maximum users can be catered to at any point of time. In order to achieve this, two approaches were suggested - connection pooling and connection caching. Connection pooling involves creating a large group (or pool) of connections from which connections are taken for processing queries and returned upon execution. The other approach that was implemented involves connection caching. This involves the reuse of pre-existing connections and it eliminates the need to repeatedly open and close them. These two approaches are explored in order to improve the efficiency of the service. In this process, many technologies were explored. Connection pooling frameworks such as HikariCP, C3P0 etc. and Guava cache by Google for connection caching are a few. In the end, it was determined that neither connection pooling nor connection caching were a good way forward because of limitations provided by Thrift. Thrift is a language-independent, lightweight software stack with an associated code generation mechanism for remote procedure call. It provides clean abstractions for data transport, data serialization, and application level processing. Thrift is not thread-safe, as a result the approaches discussed above aren't feasible.

Contents

Chapter 1

Introduction about the Industry

Based in Austin, Texas, Oracle is an American multi-national company that specializes in computer technology. Founded in the year 1977 by Larry Ellison, Bob Miner and Ed Oates, the company has come a far way from being a startup in the Silicon Valley to an industry leader today. The company sells technology and software pertaining to databases, enterprise software products — especially its own brands of database management systems, and cloud engineered systems.

Oracle has a mission to help the people perceive data in new ways that weren't available before, gather insights and present infinite possibilities with their data.

Oracle Cloud Infrastructure

Oracle Cloud Infrastructure, often referred to as OCI, is a vast and immersive platform for cloud services. It provides users a secure, scalable, high-performing environment to build and execute a large range of applications. Some of it's salient features are:

- **Autonomous Services**

  OCI exclusively provides Oracle Autonomous Database - a self-optimizing and self-repairing autonomous database. Routine tasks are automated via machine learning. OCI provides better performance, improved security, and higher operational efficiency, and enables developers to build and run enterprise applications.

- **Low Costs and High Performance**

  Oracle Cloud Infrastructure is built for users and enterprises that seek better performance, low costs, and easy cloud migration for pre-existing on-premises applications, and better price-performance for cloud native workloads. This has resulted in substantial reduction in their costs and improving compute platform performance.

- **Easy Migration of Enterprise Apps**

  OCI has made it easier than ever to migrate traditional, on-premises workloads that companies rely on for business to migrate to Oracle Cloud. It has been designed to provide compute services, traffic isolation on the network, and performance. Oracle Cloud enables swift migration and improved time to innovation. Enterprises can enhance the value of migrated applications faster with Autonomous Database, data science and cloud native development tools.

- **Support for Hybrid Architectures**

  OCI enables enterprises to deploy applications and databases with a wide range of options, from public regions to edge devices. Oracle offers cloud computing that works the way you expect with full support for VMware environments in the customer tenancy.

Oracle Applications

Oracle Fusion Cloud Applications provides the world's most connected and connected SaaS suite. It delivers a modern user experience and with non-stop innovation, committed to the success of enterprises with quarterly updates across the entire business: finance, supply chain, manufacturing, human resources, sales, marketing, advertising and customer service.

Chapter 2

Training Schedule

Week 1

The first week started off with the on-boarding process and installation of various software. We were introduced to what the company does and many senior folks of Oracle Cloud Infrastructure also talked to us. Followed by Oracle Cloud Infrastructure Essentials training. Was introduced to my team with which I would be working for the next few weeks.

Week 2

I was shared the code for the Sparkline Gateway Service. Went through the code and got all my doubts cleared. I also went through documentation of various technologies that were being used in the service.

Week 3

In this week I was exploring methods such as connection pooling and connection caching. Was devising ways in which I could implement the same and improve the performance of the gateway service. Various libraries and frameworks were also explored in addition to implementing a workflow that would reduce the latency.

Week 4

After the theoretical aspects were explored and learnt, low level changes were being made. During this process, there was a complication with respect to Thrift, it was not thread safe. As a result, this approach was not feasible. I prepared a detailed report stating why a different method had to be implemented to improve the gateway service. Suggestions were also made on new approaches and how AWS Athena also solves this issue.

Week 5

I was handed a new project which involved the Sparkline Service. The task was to schedule the start and stop of clusters. In this week I went through the code base and clarified doubts.

Week 6

Worked on coming up with an algorithm that would implement scheduling in a clean and concise manner. In addition to this, the algorithm was supposed to be scalable since there were a sizable number of clusters involved. Then proceeded to code the algorithm - the scheduler and the service that would run when the gateway would be started. Also made changes to the schema of the metadata which is being stored in Kiev (NoSQL database).

Week 7

Worked on cleaning the code and fixing bugs. Had to make changes in the API as well to take in inputs from the user that had details about the schedule of clusters.

Week 8

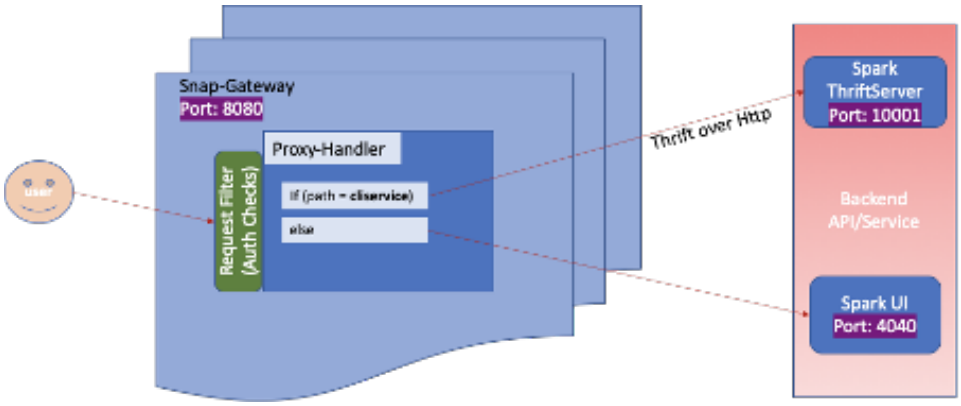
In the final week, I was preparing the presentation for the final demonstration with the larger team. Made a small Java application that resembles the actual Sparkline Service. Presented to my manager, mentors and other members of the team. Off-boarding took place after the final presentation.

Chapter 3

Introduction

3.1 What is Sparkline Gateway?

The Sparkline Gateway is a proxy service responsible for authenticating and authorizing users. In addition to this, it provides a Spark UI and JDBC endpoints on the same connection and allows for zero downtime upgrades. Spark provides an interface for programming clusters with implicit data parallelism and fault tolerance. Java Database Connectivity is an application programming interface for the programming language Java, which defines how a client can access databases. It is a Java-based data access technology used for Java database connectivity. Once authorized, the user is allowed to connect to the backend and execute queries. The gateway accommodates two types of requests - Spark UI requests and JDBC specific requests.



3.2 Need for Connection Pooling/Caching

In the current SNAP gateway service, every time a user makes a query to the data lake, it allocates a backend and creates a new connection each time. If the same user makes another query from another session, we allow a new connection and process the query. There are a few problems with the current execution. They are:

1. **Inefficient use of connections to the SNAP cluster:**

   Let us say our SNAP cluster can accept a maximum of 100 connections. If we allocate three connections to the same user, we have denied connections to 2 potential users.

2. **Large overheads involved in creating connections:**

   While opening a session and making a connection with the SNAP cluster, opening the connection is an expensive step. As a result, if we can reuse an already open connection, we save on time.

Due to the above two problems, we determined that we need to reuse the connections. Connection pooling or connection caching was the way forward to tackle this problem. As a result, we decide to look into connection pooling and connection caching

3.3 Connection Pooling

In server environments involving the JDBC API, establishing JDBC is an expensive process, with respect to time and resources. Performance can be improved remarkably when connection pooling is used in such environments. Connection pooling means that connections are reused rather than created each time a connection is requested. To facilitate connection reuse, a memory cache of database connections, called a connection pool, is maintained by a connection pooling module as a layer on top of any standard JDBC driver product. A connection is selected from this pool whenever the user makes a query, and then we execute it. After it completes, the connection returns to the pool for future use.

3.4 Connection Caching

Connection caching involves creating a connection and caching it for a certain duration. If the same connection is required, we can reuse it without worrying about creating a new connection and the overhead involved.

Chapter 4

Strategy

4.1 Methodology

4.1.1 Connection Pooling

The concept of connection pooling is that we create an extensive collection of connections. A connection is selected from this pool whenever the user makes a query, and then we execute it. After it completes, the connection returns to the pool for future use. There are quite a few frameworks available that use connection pooling. A few of them are:

1. Apache Commons DBCP
2. HikariCP
3. C3P0

All three are very similar and have the following advantages:

- easy to use
- takes care of all the connection pooling logic (we do not need to reinvent the wheel)
- lots of examples and resources, good documentation

4.1.2 Connection Caching

As discussed earlier, connection caching involves creating a connection and caching it for a certain duration. If the same connection is required, we can reuse it without worrying about creating a new connection and the overhead involved. The spark connection is independent of the request. It only depends on username, backend and the headers. Of the three entities, username and headers are the same for a given user. In order to keep track of the users and backends in use, we can map users and backend to the

connection and count of active queries that the connection is handling. In case the same user is redirected to a new backend, it opens a new connection (connection is unique to a user and a backend). To implement connection caching, we explore Guava cache. It is a very robust library that has a ton of valuable features.

- Guava cache checks to see if it already has the corresponding value for the supplied key
- if it does, it simply returns it (assuming it hasn't expired)
- if it doesn't already have the value, it uses a CacheLoader to fetch the value and then it stores the value in the cache and returns it
- this way, the cache is growing as new values are requested (it is an incremental cache)
- set timeout value using the expireAfterAccess method, which tells the cache to throw away values that has not been read (or written) in a certain amount of time
- to update values in the cache, use the put method

The cache will have the key as the user and the backend allotted to the user. The associated value with the key would be the connection and an integer that keeps count of the active requests. After being allotted a backend, the key-value pairs are populated.

There are two main steps involved in the connection caching approach:

1. **Handling New Requests:**
   - whenever a new request is made, we check if the user and the corresponding backend already have an open connection or not.
   - in case it is available, we reuse the connection by calling a certain segment of code (will be discussed below).
   - in case no such key-value pair exists, we follow the existing procedure and add the new connection to the maps. While reusing the connections, we must update the request.
2. **Closing a Transport:**

   To close a transport, the following two criteria must be satisfied:
   - no active request running on the backend
   - the transport has been inactive for a specified duration of time (say, 5 minutes)

4.1.3 Low Level Changes

The low level changes required to implement connection caching:

- guava cache should be a map of ¡Object, transport¿ where Object has user and backend while transport corresponds to the open connection for the particular user and backend
- as soon as sendRequest is called, we update the value of username and backend in the key and the resulting transport is updated as the value for the associated key (put)
- a method exists which deletes the cache elements after a specified duration of inactivity (expire AfterAccess)

4.2 Technologies and Tools Used

4.2.1 Thrift

Apache Thrift is a lightweight, language-independent software stack with an associated code generation mechanism for RPC. It provides abstractions for data transport, data serialization, and processing at an application level. Originally developed by Facebook, it is now an open sourced project by Apache. It allows developers to build RPC clients and servers by just defining the data types and service interfaces in a simple definition file via a set of code-generation tools. Code is generated to build RPC servers and clients that communicate flawlessly across programming languages using this file as an input.

4.2.2 Guava Cache

Guava cache is a library that provides users with a cache which is flexible and powerful. Developers prefer this over others because the it can be used when fast access is required and when values are retrieved on multiple occasions.

4.2.3 Apache Spark

Apache Spark is an open-source unified analytics engine. It is used for large-scale data processing. It provides an interface for programming clusters with data parallelism and increased fault tolerance.

Chapter 5

Results and Analysis

5.1 Connection Pooling

There were plenty of connection pooling frameworks that were discussed in the previous section. Though the frameworks were terrific, they had quite a few disadvantages:

- None of them support dynamic users and connections
- As a result, the users need to be defined earlier and are static
- Thus, it is not an excellent method to implement connection pooling for this particular service since it has users and passwords which are dynamic (not predefined) and cannot be stored statically

Due to these drawbacks, it was determined that using preexisting connection pooling frameworks was not feasible. If we proceed to use a custom connection pooling method, it will not be very efficient due to the dynamic nature of the users. For every user we cannot create connections as per their requests because the pool will get exhaust pretty fast. The alternative is to set a limit on the number of connections permitted per user. However, it becomes a tedious task to keep track of all the active requests in the gateway. In addition to this, the memory overhead is also significant and cannot be ignored.

5.2 Connection Caching

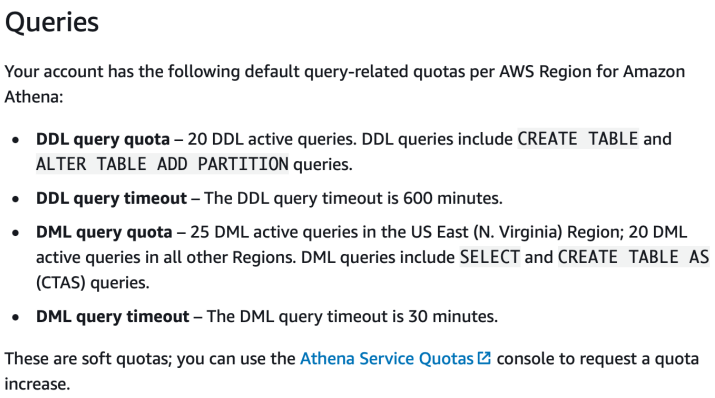
While exploring this approach, we realise that Thrift is not thread-safe. This means that across multiple threads we cannot use the same connection in parallel. As a result, connection caching cannot be implemented. This development in the project resulted in aborting the plan. Alternatives had to be found and evaluated.

Chapter 6

Conclusion

Since both the approaches aren't feasible, alternatives are being explored, the concept of quotas was also taken into consideration. That way, one user cannot exploit more resources than necessary and the service would serve all the users equally.

In addition to this, a decision is made to see how AWS Athena implements efficient usage of connections for its own services. AWS Athena uses the concepts of quotas to execute queries. These quotas limit the load on the servers and ensure that all users are equally catered to.



Queries

Your account has the following default query-related quotas per AWS Region for Amazon Athena:

- **DDL query quota** – 20 DDL active queries. DDL queries include `CREATE TABLE` and `ALTER TABLE ADD PARTITION` queries.
- **DDL query timeout** – The DDL query timeout is 600 minutes.
- **DML query quota** – 25 DML active queries in the US East (N. Virginia) Region; 20 DML active queries in all other Regions. DML queries include `SELECT` and `CREATE TABLE AS` (CTAS) queries.
- **DML query timeout** – The DML query timeout is 30 minutes.

These are soft quotas; you can use the Athena Service Quotas console to request a quota increase.

As discussed above, implementing a custom connection pool in the gateway would be extremely inefficient due to memory overheads and making the code complex at the gateway level. The alternative is to have the quota logic in the Spark backend. These changes are quite significant and would require changes in some of the key components.

References

[1] https://www.baeldung.com/java-connection-pooling

[2] https://www.baeldung.com/guava-cache

[3] https://guava.dev/releases/17.0/api/docs/com/google/common/cache/Cache.html

[4] https://blog.jayway.com/2012/04/16/introducing-the-google-guava-cache/

[5] https://stackoverflow.com/questions/37674466/how-to-make-a-thrift-client-for-multiple-threads

[6] https://docs.aws.amazon.com/athena/latest/ug/service-limits.html

[7] Oracle_Proprietary_Documentation

Summer Internship Project
Report

Scheduling the Start and Stop of Clusters in Sparkline Service

Anjaneya Tripathi
106118010
National Institute of Technology, Tiruchirappalli

Under the guidance of

Priyank Rastogi
Director, Product Development
Data and AI Services

Data and AI Services
Oracle Cloud Infrastructure (OCI)
Bengaluru

Summer Internship 2021

Acknowledgment

The internship opportunity I had with Oracle Cloud was an amazing chance for me to learn and grow professionally as a developer and person. Therefore, I find myself an extremely fortunate individual as I was provided with an opportunity to be a part of this phenomenal organisation. I am also grateful for having a chance to meet so many wonderful people and professionals who led me though this internship period.

I would also like to use this opportunity to show my gratitude to my manager Priyank Rastogi who despite being extremely busy with his work, took time out to hear me, guide me, ensure I was on track and allow me to carry out my project at their esteemed organization.

I express my deepest thanks to Vivek Pathak and Ankur Jain for mentoring me through out the duration of my internship. They were instrumental in giving useful advice and guidance and arranged all facilities to make my 8 weeks pleasurable. I use this moment to acknowledge their contribution gratefully.

This opportunity is a big milestone in my life and career. I strive to use the skills and knowledge gained in the best possible way, and continue to work on the improvements suggested by my team, in order to attain desired career objectives. I wish to continue cooperating with all of you in the near future.

Sincerely,

Anjaneya Tripathi

Chennai, India

27th August, 2021

ORACLE

Oracle India Pvt. Ltd.
India Development Center

Oracle Technology Park
3, Bannerghatta Park
Bangaluru - 560 029

CIN: U74899DL1993PTC051764
Phone +91 80 4107 0000
Fax +91 80 2552 6124

Private & Confidential

Ref: Oracle-Interns- 1455577

26th Aug 2021

TO WHOMSOEVER IT MAY CONCERN

This is to certify that Mr. Anjaneya Tripathi , student of NIT Tiruchirapalli (Pursuing B.Tech (Computer Science and Engineering)). has completed his project with Oracle India Private Limited.

The project was undertaken from 02-Jun-2021 to 27-Jul-2021.He worked on "Implementing Connection Pooling/Caching on SNAP Gateway Service"

We wish him all the best in his future endeavours.

Yours sincerely

For Oracle India Private Limited.

Rambabu Jagatha
Director – HR Operations

Abstract

Sparkline is a new feature of data flow service, which allows users to perform interactive query analytics alongside data flow's batch analytics. Before being able to run their queries, our users need to provision a Sparkline cluster, define their (cube) schema, as well as load (or attach the existing) data for analysis. Sparkline enables fast exploratory data analytics. With improved Apache Druid's smoosh file format and inbuilt caching, it provides high-performance aggregation, OLAP, and multi-dimensional cubing in general. The current service expects the user to manually start and stop the cluster whenever they want. We decided that if we could automate the start and stop of clusters it would be an added feature that the user can utilise. Regarding the implementation, three new input parameters were to be added - isScheduled, startTime, stopTime. All the details of the cluster are stored in Kiev. Kiev is a noSQL database created by Oracle. It stores all the details of the cluster and was used for retrieving the startTime, stopTime etc. via indices in the code. In order to ensure that the retrieval of the clusters is fast, we decided to add an index on the start and stop hours. We set up workers which will be allotted different hours. The workers are responsible for the hours they are allotted and will perform operations in that time span. Once the database is set up, we need to query the clusters and start or stop the necessary clusters based on the current time. Due to data privacy reasons, access couldn't be granted to me for the original Sparkline Service. In order to show how the scheduling was being implemented, I made my own Java application that mimicked the Sparkline Service. Changes had to be made in the API as well. It required addition of 3 inputs from the user - isScheduled, startTime and stopTime. The latter two are CRON strings indicating the time when the cluster is to be started and stopped. Once this was done, tests were also written to ensure the modifications would work without any issues.

Contents

Chapter 1

Introduction about the Industry

Based in Austin, Texas, Oracle is an American multi-national company that specializes in computer technology. Founded in the year 1977 by Larry Ellison, Bob Miner and Ed Oates, the company has come a far way from being a startup in the Silicon Valley to an industry leader today. The company sells technology and software pertaining to databases, enterprise software products — especially its own brands of database management systems, and cloud engineered systems.

Oracle has a mission to help the people perceive data in new ways that weren't available before, gather insights and present infinite possibilities with their data.

Oracle Cloud Infrastructure

Oracle Cloud Infrastructure, often referred to as OCI, is a vast and immersive platform for cloud services. It provides users a secure, scalable, high-performing environment to build and execute a large range of applications. Some of it's salient features are:

- **Autonomous Services**

  OCI exclusively provides Oracle Autonomous Database - a self-optimizing and self-repairing autonomous database. Routine tasks are automated via machine learning. OCI provides better performance, improved security, and higher operational efficiency, and enables developers to build and run enterprise applications.

- **Low Costs and High Performance**

  Oracle Cloud Infrastructure is built for users and enterprises that seek better performance, low costs, and easy cloud migration for pre-existing on-premises applications, and better price-performance for cloud native workloads. This has resulted in substantial reduction in their costs and improving compute platform performance.

- **Easy Migration of Enterprise Apps**

  OCI has made it easier than ever to migrate traditional, on-premises workloads that companies rely on for business to migrate to Oracle Cloud. It has been designed to provide compute services, traffic isolation on the network, and performance. Oracle Cloud enables swift migration and improved time to innovation. Enterprises can enhance the value of migrated applications faster with Autonomous Database, data science and cloud native development tools.

- **Support for Hybrid Architectures**

  OCI enables enterprises to deploy applications and databases with a wide range of options, from public regions to edge devices. Oracle offers cloud computing that works the way you expect with full support for VMware environments in the customer tenancy.

Oracle Applications

Oracle Fusion Cloud Applications provides the world's most connected and connected SaaS suite. It delivers a modern user experience and with non-stop innovation, committed to the success of enterprises with quarterly updates across the entire business: finance, supply chain, manufacturing, human resources, sales, marketing, advertising and customer service.

Chapter 2

Training Schedule

Week 1

The first week started off with the on-boarding process and installation of various software. We were introduced to what the company does and many senior folks of Oracle Cloud Infrastructure also talked to us. Followed by Oracle Cloud Infrastructure Essentials training. Was introduced to my team with which I would be working for the next few weeks.

Week 2

I was shared the code for the Sparkline Gateway Service. Went through the code and got all my doubts cleared. I also went through documentation of various technologies that were being used in the service.

Week 3

In this week I was exploring methods such as connection pooling and connection caching. Was devising ways in which I could implement the same and improve the performance of the gateway service. Various libraries and frameworks were also explored in addition to implementing a workflow that would reduce the latency.

Week 4

After the theoretical aspects were explored and learnt, low level changes were being made. During this process, there was a complication with respect to Thrift, it was not thread safe. As a result, this approach was not feasible. I prepared a detailed report stating why a different method had to be implemented to improve the gateway service. Suggestions were also made on new approaches and how AWS Athena also solves this issue.

Week 5

I was handed a new project which involved the Sparkline Service. The task was to schedule the start and stop of clusters. In this week I went through the code base and clarified doubts.

Week 6

Worked on coming up with an algorithm that would implement scheduling in a clean and concise manner. In addition to this, the algorithm was supposed to be scalable since there were a sizable number of clusters involved. Then proceeded to code the algorithm - the scheduler and the service that would run when the gateway would be started. Also made changes to the schema of the metadata which is being stored in Kiev (NoSQL database).

Week 7

Worked on cleaning the code and fixing bugs. Had to make changes in the API as well to take in inputs from the user that had details about the schedule of clusters.

Week 8

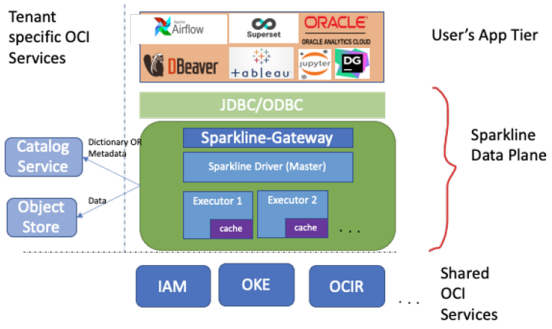
In the final week, I was preparing the presentation for the final demonstration with the larger team. Made a small Java application that resembles the actual Sparkline Service. Presented to my manager, mentors and other members of the team. Off-boarding took place after the final presentation.

Chapter 3

Introduction

3.1 What is Sparkline Service?

Sparkline is a new feature of data flow service, which allows users to perform interactive query analytics alongside data flow's batch analytics. Before being able to run their queries, our users need to provision a Sparkline cluster, define their (cube) schema, as well as load (or attach the existing) data for analysis.



Sparkline enables fast exploratory data analytics. With improved Apache Druid's smoosh file format and inbuilt caching, it provides high-performance aggregation, OLAP, and multi-dimensional cubing in general.

3.2 Need for Scheduling of Clusters

The current service expects the user to manually start and stop the cluster whenever they want. We decided that if we could automate the start and stop of clusters it would be an added feature that the user can utilise.

Chapter 4

Strategy

4.1 Ideation

4.1.1 Why Perform EDA in the first place?

Assuming we group clusters together on the basis of their starting and stopping times, there is a high chance that the data will be skewed. The issue with such skewed data is that in some of the intervals, the amount of data that needs to be scanned through will be significantly larger than another segment. This uneven distribution of data can pose as an issue when it comes to computational speed. Thus, if we can do some initial analysis of the data available at hand and suggest a suitable data structure as well as distribution, we can ensure that while scanning for the start and stop time of clusters the computational speed is not compromised.

4.1.2 How do we perform EDA?

Every time the user creates a new cluster, we will also be given the start and stop times of the corresponding clusters. A separate table can be created with the interactiveClusterId, startTime and stopTime. Once we have the table ready, an analysis can be made and suitable batches can be made and a data structure implemented to ensure faster scanning of the records.

4.1.3 Method 1

Once we have a table of all the clusters that have scheduled start and stop times, we proceed to analyse the same. We can try various methods such as:

- classifying based on hours
- classifying based on minutes (unlikely to be helpful but worth a shot)

Once classified, we can store them in batches. For example, if we decide to store on the basis of hours, we will have a separate table for each hour and while populating the table we will ensure that it is sorted. The population phase will be expensive, but once the table is created, for starting and stopping we will be much faster. There will be a tradeoff here so we may need to perform some tests to compare if sorting is the most optimal way forward. We can have it as an array of arrays or key-value pairs where the values are a list containing the interactiveClusterId, startTime and stopTime.

Thus, every time the scheduler has to check which clusters to start/stop, we just need to go to the associated batch rather than checking all the entries.

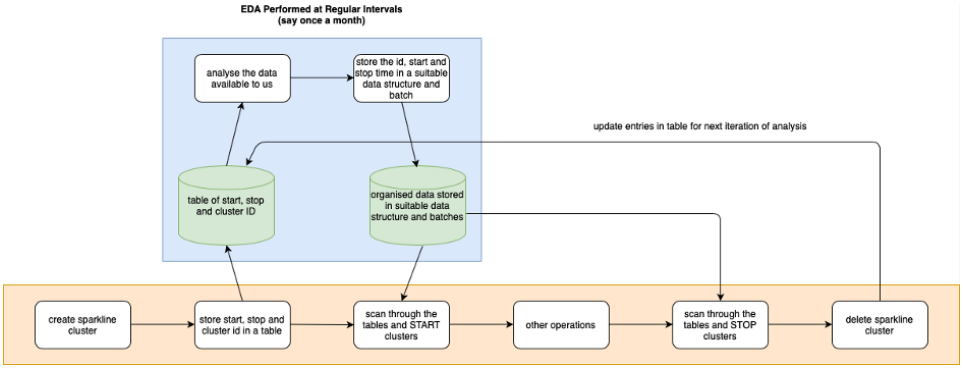
4.1.4 Method 2

Another method that can be employed is creating a hash map for the start and stop times separately. We will require 2 workers, one for checking the start map and the other for the stop map. The worker will run every minute and check the map associated with the time that the worker has started. In case the time for a particular entry takes more than a minute to scan, we can have a memory to keep track till where we have scanned. This backlog will eventually be cleared because the distribution will not be consistent across the whole duration. The same issue will arise in the brute force approach, but at a larger scale.

4.1.5 Method 3

Create batches of equal elements but varying intervals. While scanning, we look at the start times of the particular batch and if it lies in range of starting a cluster, we select that batch and start all the associated clusters.

4.1.6 Pipeline



4.2 Approach

Three new input parameters were to be added - isScheduled, startTime, stopTime.

1. **isScheduled:** a boolean that indicates if the cluster will be scheduled or not
2. **startTime:** a CRON expression indicating when the cluster should be started
3. **stopTime:** a CRON expression indicating when the cluster should be stopped

All the details of the cluster are stored in Kiev. In order to ensure that the retrieval of the clusters is fast, we decided to add an index on the start and stop hours. This resulted in adding two more columns, start Hour and stopHour which would be indexed. The indices assist us to reduce the querying time from Kiev.

We set up workers which will be allotted different hours. The workers are responsible for the hours they are allotted and will perform operations in that time span.

Once the database is set up, we need to query the clusters and start or stop the necessary clusters based on the current time. We have set a buffer which checks for clusters that are starting in a certain time span. This buffer can be adjusted after suitable to determine the time taken to start and stop clusters.

4.3 Making a Mock Sparkline Service Application

Due to data privacy reasons, access couldn't be granted to me for the original Sparkline Service. In order to show how the scheduling was being implemented, I made my own Java application that mimicked the Sparkline Service.

4.4 Technologies and Tools Used

4.4.1 SLF4J

Simple Logging Facade for Java (SLF4J) – behaves as a facade for a variety of logging frameworks. The logging is independent of the actual implementation because it offers a generic API.

4.4.2 Kiev

It is a noSQL database created by Oracle. Kiev stores all the details of the cluster and was used for retrieving the startTime, stopTime etc. via indices in the code.

4.4.3 Drop Wizard

Dropwizard gathers stable libraries from Java ecosystems and combines them into a simple package that lets developers focus on completing the work. It has out-of-the-box support for logging, tools, complex configurations and much more, allowing developers and their team to produce exceptional web services in short spans of time.

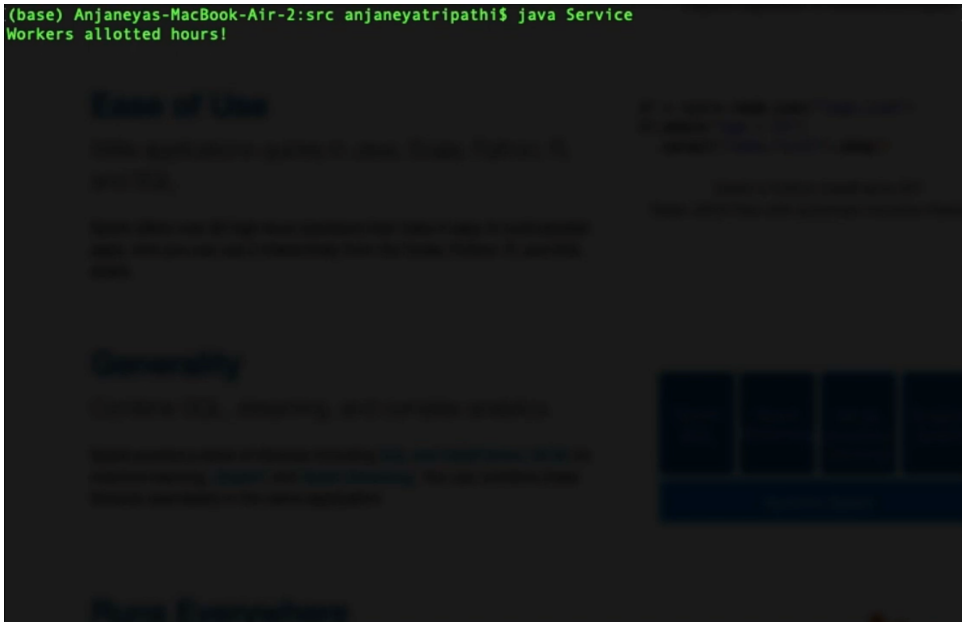
4.4.4 Apache Spark

Apache Spark is an open-source unified analytics engine. It is used for large-scale data processing. It provides an interface for programming clusters with data parallelism and increased fault tolerance.

Chapter 5

Simulation and Analysis

To determine the working of the scheduler and service, a mock Java application was built to enact the actual Sparkline Service. The observation are available in the screenshots below.

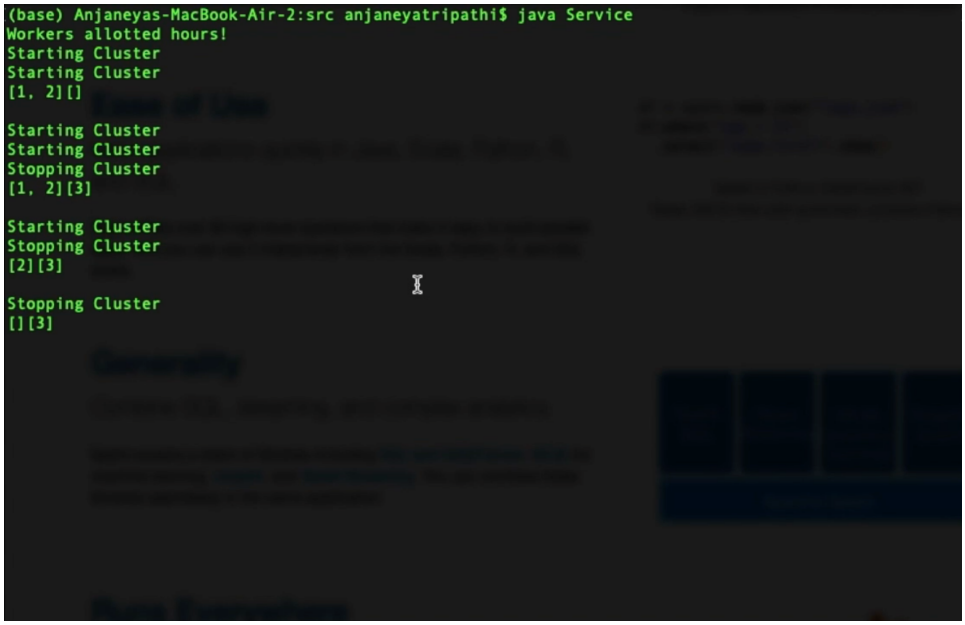


In the above image, the service boots up and allots hours to the various workers. Once allotted, the workers will check if any cluster is scheduled to start or stop.

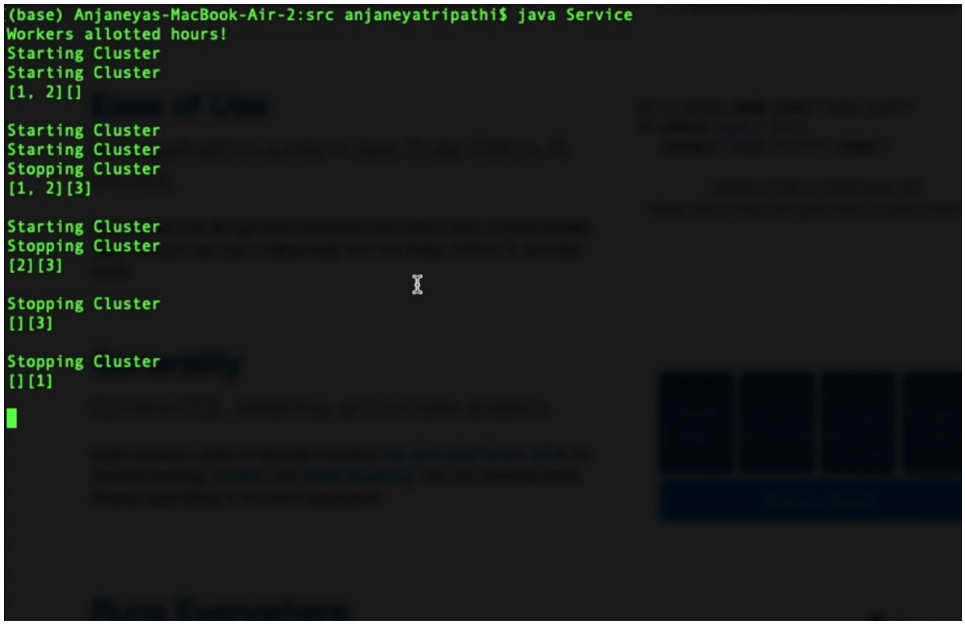
After the allotment, the worker determines that clusters 1 and 2 need to be started. At the moment, no cluster lies in range of the buffer for it to be shut down.

At this instance, cluster 3 needs to stopped as its time has entered into the range of the buffer. Clusters 1 and 2 are still in range, so it indicates that the clusters need to be started if they haven't.

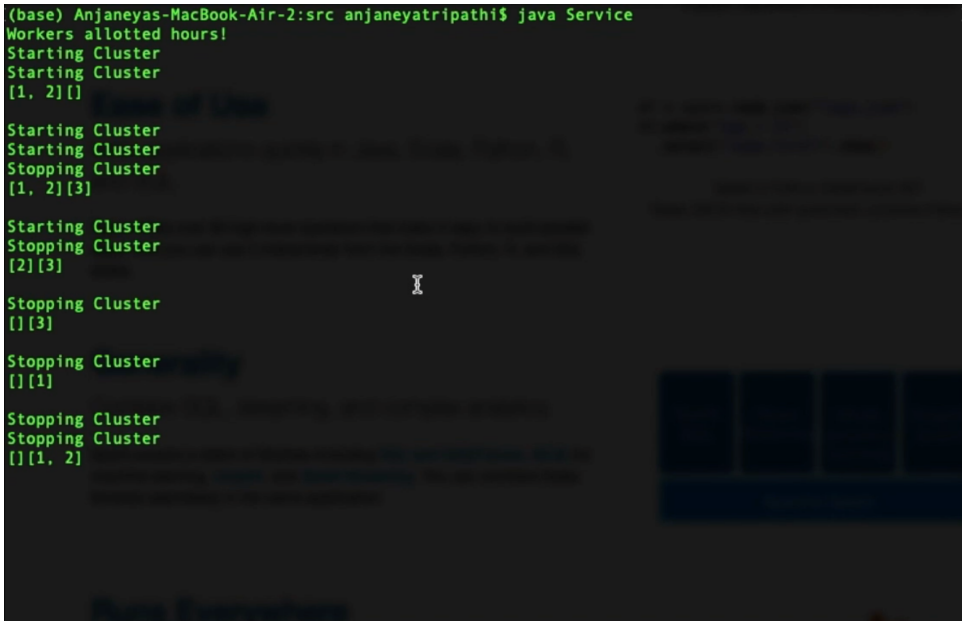
In this case, cluster 1 falls out of range in which it needs to be started, so only cluster 2 is mentioned. Cluster 3 is still in range for it to be stopped.



The only cluster in range to be stopped is cluster 3.



In the above case, cluster 1 is now in the range for it to be stopped. That is why it is the only cluster scheduled to stop.



Finally, cluster 2 also arrives in the range of the buffer, so it is stopped as well.

From this simulation, it is observed that the mock Java app worked successfully and implemented the scheduled start and stop of clusters.

Chapter 6

Conclusion

From the simulation and analysis of the Sparkline Service, it was observed that the scheduler and service worked flawlessly and gave the desired output. Through this, we had successfully implemented the scheduling of clusters.

I got to learn about Kiev, a NoSQL database by Oracle and how it stores and indexes data. In addition to this, I was exposed to various frameworks which let the developers focus on the task at hand rather than worrying about how to integrate different tools and libraries. SLF4J, Drop Wizard are a few to mention. I was also able to learn about the importance of clean code and the importance of documentations. This project shed light on how crucial writing code is. It isn't just code to get the program to work, the code should be efficient, scalable and neat.

This shows that the scheduled start and stop of clusters was successfully implemented.

References

[1] https://www.dropwizard.io/en/latest/

[2] http://www.slf4j.org

[3] https://www.baeldung.com/slf4j-with-log4j2-logback

[4] https://spark.apache.org/docs/latest/

[5] https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html

[6] Oracle_Proprietary_Documentation