VIEW POINT

LOAD BALANCING SPARK THRIFT: SCALABLE APPROACH ON HA SERVICES AND LOAD BALANCING ON SPARK THRIFT SERVICE USING NGNIX

Abstract

Enterprises need to ensure that the technology stack which caters to the business critical applications to be operated in a fault tolerant manner. This becomes even critical in the analytics domain within the Big data landscape. The technology stack components needs to be deployed in distributed fashion which can deliver a dependable and steadfast level of service availability with huge and unpredictable data traffic. In the Spark world, the most common problem that we face is the load balancing of the spark thrift services for enterprises to consume it in various layers including reporting and visualization tier.

Keeping the above in mind, we wanted to integrate the capability of NGNIX as a load balancer to spawn multiple thrift services to satisfy the growth demand. In this implementation, we use NGINX open source web server software to balance the load across multiple spark-thrift instances running across the cluster. This technique is common for optimizing the resource utilization, maximizing throughput, reducing latency, and ensuring fault-tolerant configurations.

Infosys®
Navigate your next

Introduction

In order to achieve load balancing among multiple spark thrift services, we deploy NGINX as a standalone load balancer with round-robin as default model in routing the data traffic. This document covers the architecture and configuration steps that needs to be taken care to achieve this requirement.

NGNIX

In a typical web architecture world, NGINX can be deployed in a variety of scenarios as a very efficient HTTP load balancer. To proceed with the configuration steps, we will require multiple spark thrift service instances to be up and running in the cluster. Within NGINX configuration, this will be defined as a group with the upstream directive. This directive comes under the 'http' context within NGINX.

Available Load Balancing Models in NGINX

1. Round-robin – Requests are distributed evenly across the servers with server weights taken into consideration. This method is used by default (there is no directive for enabling it)
2. least_conn – A request is sent to the server with the least number of active connections with server weights taken into consideration
3. ip_hash – The server to which a request is sent is determined from the client IP address. In this case, either the first three octets of IPv4 address or the whole IPv6 address are used to calculate the hash value. The method guarantees that requests from the same address get to the same server unless it is not available
4. Generic hash – The server to which a request is sent is determined from a user-defined key which may be a text, variable, or their combination. For example, the key may be a source IP and port, or URI
5. least_time (NGINX Plus) – For each request, NGINX Plus selects the server with the lowest average latency and the least number of active connections, where the lowest average latency is calculated based on which of the following parameters is included on the least_time directive

Spark Thrift

Spark thrift is a variant of HiveServer2 implementation which allows JDBC and ODBC clients to run Spark SQL queries on Spark cluster.

Spark thrift server is very much similar to hiveserver2 thrift implementation. HiveServer2 will interpret and submit the SQL query constructs as Hive MR job whereas Spark thrift service will use Spark SQL context which utilizes the complete spark capabilities to get and manipulate the data-frame. The consumption tier like Tableau, PowerBI or R can get connected to spark thrift server using an ODBC/ JDBC driver just like how we do with a hiveserver2 and access the hive or spark temporary tables registered within the hive context.

Spark Thrift Server features and configuration details

By enabling user impersonation to run SQL queries under the identity of the user who originated the query. By default, queries will run under the user associated with the Spark Thrift server.

Challenges in Integrating Spark Thrift and NGNIX Server

Reporting / BI Tools use the Spark Thrift Server to convert ODBC/JDBC calls with spark layer for achieving distributed and highly-parallel data processing capabilities in an efficient manner. This is commonly done using tables registered inside the spark context either in HDFS or S3 etc. that are common populated for short periods of time.

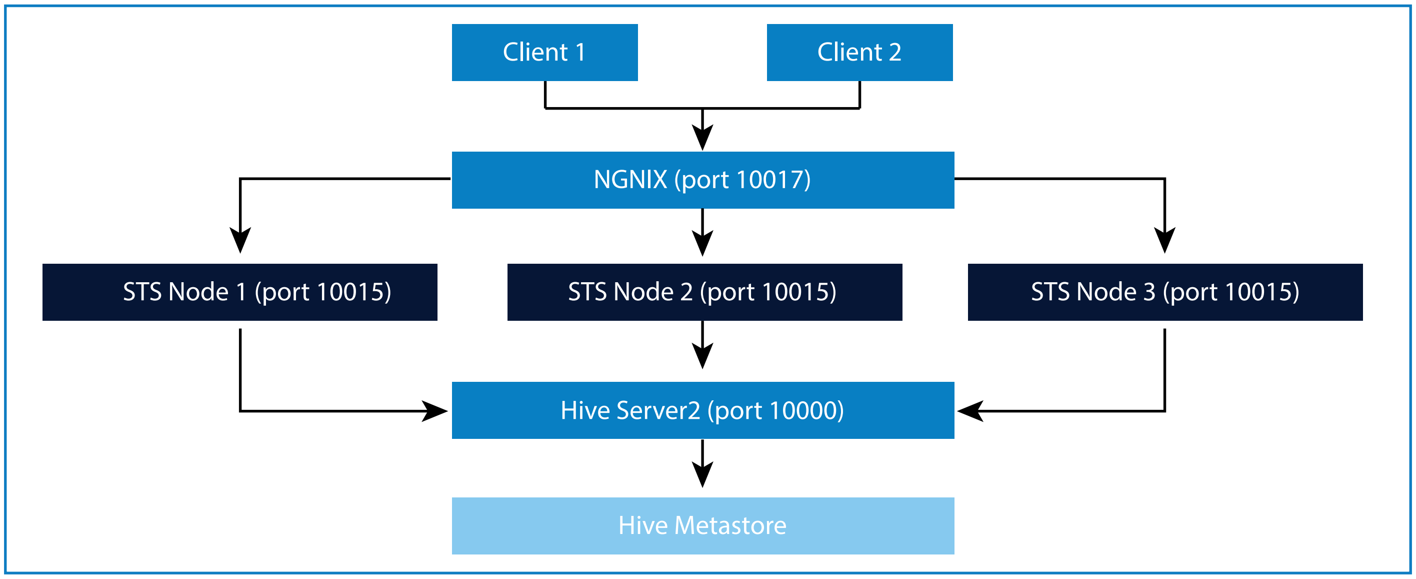
Apache Spark Thrift Server is based on the Apache HiveServer2 which was created to allow JDBC/ODBC clients to execute SQL queries using a Spark Cluster. Load balancer in front of Spark Thrift Service - STS when the cluster with or without kerberized and not available directly from the spark. It appears this functionality is currently being added to the Apache-Spark (so we will have to wait a bit longer for it be included in distributions such as HDP or Cloudera). Refer the below JIRA tickets,

https://issues.apache.org/jira/browse/SPARK-11100

https://github.com/apache/spark/pull/9113

For now, there is no load balancing for STS for kerberized environment is available, So for now we have two options in front of us either to go with Apache based - haproxy, httpd +mod_jk or NGNIX based load balancer for non kerberized environment. To achieve this, there are couple of configuration changes to be done at various service environment tiers. Let's understand the deployment architecture.

Multi STS – NGNIX Deployment architecture



Prerequisite and Configuration for NGNIX for Load balancing

NGNIX requires a set of prerequisite and configuration changes, the following description will detail on each of these. It is expected that the following lib are available before the NGNIX installation.

- gzip module requires zlib library
- rewrite module requires pcre library
- ssl support requires openssl library
- Configure Spark Thrift Server (STS) transport mode to listen in HTTP mode. Propagate the configuration changes across all the nodes in the cluster.
- Start STS in multiple nodes within the cluster nodes

It as work with TCP module of NGNIX, you get further information git://github.com/yaoweibin/nginx_tcp_proxy_module.

Once we build the TCP module based NGNIX, next step is to configure the NGNIX. The configuration changes to be applied at the ngnix.conf file.

```
worker_processes 1;
events {
        worker_connections 1024;
}
tcp   {
        upstream cluster {
                server 12.122.23.107:10015;
                server 12.189.53.100:10015;
                check interval=3000 rise=2 fall=5 timeout=1000;
        }
        server {
                listen 10017;
                proxy_pass cluster;
        }
}
```

Make sure that Port number "10015" is the port on which spark-thrift is running on different nodes and port "10017" is the port on which NGNIX listen to client request . After the configuration a restart of the service required and under /usr/local/nginx/logs/tcp_access.log to view which spark thrift connection logs

Example with Down Stream Systems

Downstream applications such as Tableau, PowerBI or R code can access the data from STS using ODBC/JDBC connection with SASL to NGINX server port.

References

https://www.nginx.com/resources/admin-guide/load-balancer/

https://community.hortonworks.com/questions/33715/why-do-we-need-to-setup-spark-thrift-server.html

https://community.hortonworks.com/questions/116074/run-spark-thrift-server-on-multiple-nodes.html

About the authors

Vishnu Sankar

Architect

Big Data Architect at Infosys with experience in research and development in Big Data & Analytics space with product designing, development & integration. Also a subject matter expert to all teams involved in the POC ,Projects including business and architectural analysis, technical design and development and architecture using big data solutions.

Santhosh Devarajan

Big data consultant

Santhosh Devarajan is a big data consultant and developer with more than 8 years of experience in IT.He started his career with Cognizant Technology Solutions in 2009 with data warehousing and business Intelligence as his core domain. He now works with Infosys Limited in the big data and analytics domain with specialization on Apache Spark, Hive, workflow tools such as Azkaban and Airflow. He is a member of the BigData Center of Excellence within Infosys and regularly works on enterprise related problems in the big data and analytics space.

Parav Kumar Sanjeev Kumar Patel

Senior Systems Engineer

Senior Systems Engineer in Infosys Big Data COE with passion about learning and exploring various technologies and having working experience in Big data technologies like Hadoop, Spark, Hive, Java etc. Being a part of Big Data COE I have worked on many solutions and POC.As a part of learning and exploring new technologies in the world of Big Data I do believe that if you want to keep succeeding in this field you have to continuously keep learning and exploring daily under all odd and even circumstances.