Trace collection

--------Documentation Summer 2020

1. database & cache configuration:

·database: postgresql/mysql/Cassandra (in the virtual machine)

cache: redis (local)

workloads: McGill/Netflix/venues(in java,local，codes in this directory)

* Using corresponding JDBC and Jedis to set up connection to database/cache, we ran 50 threads in parallel to get more flows in between. When the query has appeared before, it’s read from cache directly; if not, the query result is retrieved from database and stored into cache, whose key is the hash code of the query. The value format stored in Redis are json-string, linked list and array list.

1. YCSB & Cassandra cluster

* database: Cassandra cluster in docker / in AWS (sample of Cassandra.yaml is in this dir)

workloads: YCSB workloada/workloadc/workloade

port used: 7000 for inter-cassandra communicateon; 9042 for client connection

commands: (first load then run)----workload example is in this dir:ycsb\_workloada

./bin/ycsb load cassandra-cql -p hosts=HostIP -P workloads/workloada >loada.txt

./bin/ycsb run cassandra-cql -p hosts=HostIP -P workloads/workloada -s –threads 50>loada.txt

* There are 3 cassandra ‘nodes’in both configurations, namely cas1, cas2 and cas3. cas1 and cas2 are in the datacenter1 while cas3 is in the datacenter2. 2 replication rules are used：each node holds one replica(I) or each datacenter has one (half splitted between cas1 & cas2)(II). We only specified cas1’s ip in the YCSB run command(./bin/ycsb run cassandra-cql -p hosts=172.17.0.2 -P workloads/workloada >loada.txt).

✓docker setup:

cas1 and cas2 are in the default docker0 network, with ip address being 172.17.0.2 and 172.17.0.3, while cas3 is in my\_bridge network and its ip is 172.20.0.1. Those two networks are connected through routing table.

✓AWS setup:

cas1 and cas2 are in the same region and availability zone, which is different from the availability zone of cas3.

* observations:

client talks to cas1 and cas2 in replication rule II while only talks to cas1 in rule I. There’s inter-cassandra communication due to write consistency and heartbeat.

1. Spark bench & spark

* database: spark cluster in docker(standalone mode) / in AWS(yarn cluster mode with zookeeper)

workloads: Sparkbench workloads(SparkPi/linear regression/kmeans)

command: ./bin/spark-bench.sh /path/to/your/config/file/ (sample of config file ‘pi.conf’ is in this dir)

* There are three nodes in both configurations, namely master(master.hadoop),worker1(worker1.hadoop),and worker2(worker2.hadoop).

✓standalone mode setup:

|  |  |
| --- | --- |
| Node | Ip address |
| Spark-bench client | 172.21.0.5 |
| master | 172.21.0.2 |
| Worker1 | 172.21.0.3 |
| Worker2 | 172.21.0.4 |

Flows detected:

client - worker1 : 44847 - 40510/40514/40526

                            48808 - 42373(very few)

                            45341 - 56014(very few)

         client - worker2 : 44847 - 34110/34134/34114

                             45341 - 46798(few)

                             40643 - 46072(very few)

         client - master :  57186/56790 - 7077

         master - worker1 : 7077 - 35398

         master - worker2 : 7077 - 53900

In the spark-bench configuration file, we only specified the master ip and path for spark. To increase the flows, we add on more partitions of the task.(pi\_20.cap -> pi\_4000.cap shows that the increment of partitions will give more flows)

✓yarn-cluster mode setup:

◇services on each node on AWS(3 EC instances):

master.hadoop(172.31.76.109): hadoop resource manager + namenode + spark master+ spark-bench client(tried to connect locally to AWS but the latency is too high)+zookeeper;

   worker1.hadoop(172.31.78.246): hadoop datanode + spark worker + spark master + zookeeper;

   worker2.hadoop(172.31.79.12): hadoop datanode + spark worker+zookeeper;

   The master alive is switched from worker1 to master.

◇port used for each service:

2181: zookeeper client connection : for zookeeper, spark Master is the client.

2287 : communication between zookeeper servers

9000 : used by hadoop namenode; heartbeat between namenode and datanode for hadoop

8031 : connection to yarn resource manager

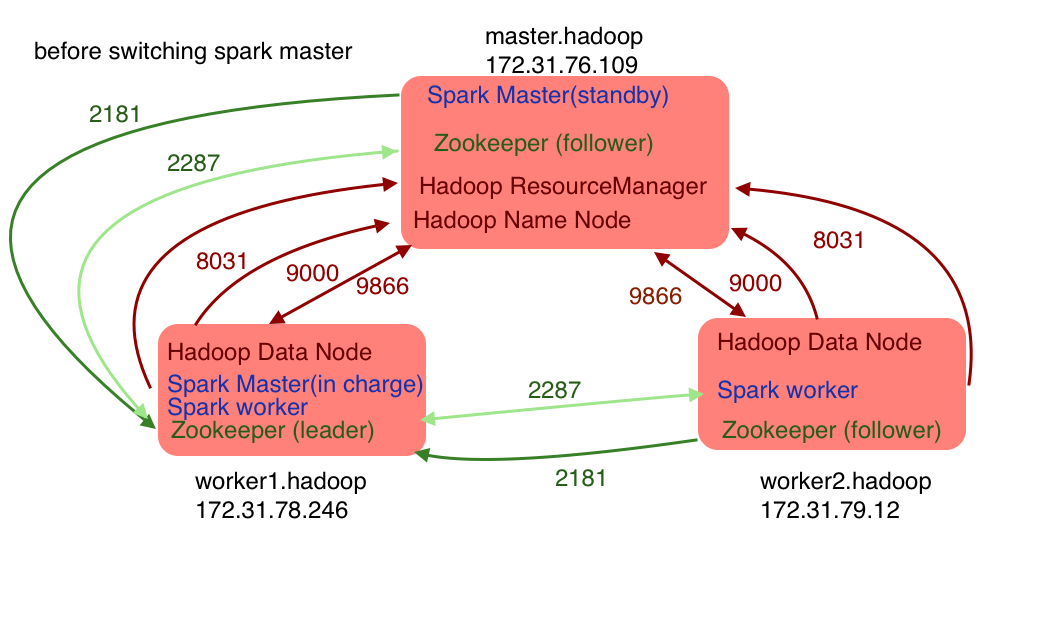
9866 : used by hadoop datanode; data transmission to spark workers

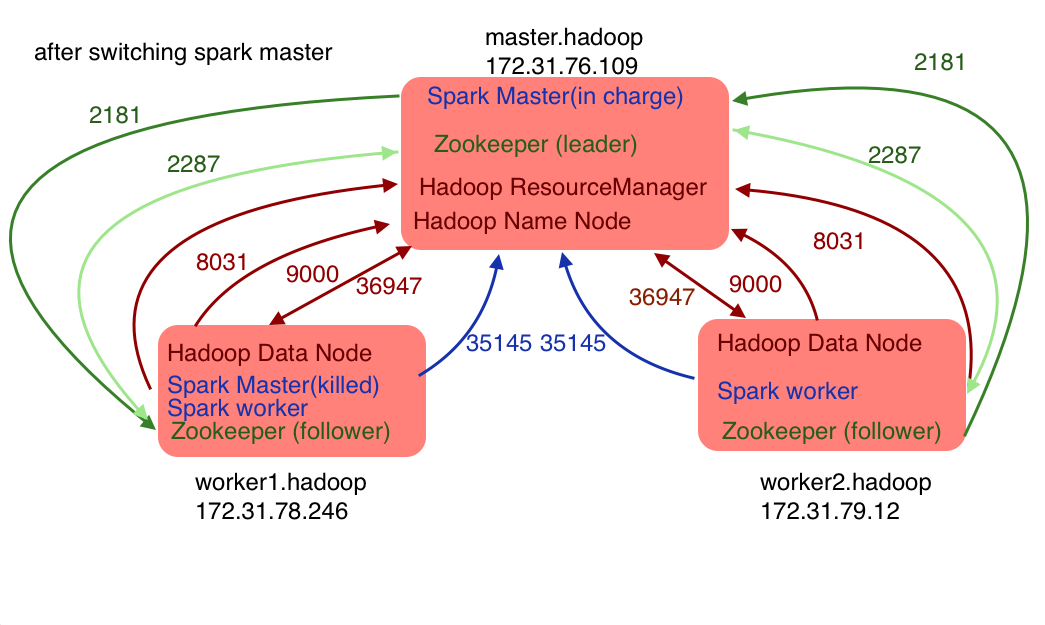
36947: data transmission for master.hadoop(used as 9866 since it’s not a datanode)

35145: Spark cleanup

The diagram before and after switching the master is attached below:

The color of arrows corresponds to the color of service where the ports above are used.





Future work:

1. Traces with inter-cassandra encryption are collected. But as for client-Cassandra encrypted communication(namely between ycsb and cassandra), there’s a parameter ‘cassandra.useSSL’ in YCSB to enable SSL.

2. The reason why Cassandra traces are mistaken as postgresql traces might be that it uses Lz4 compression which has similar pattern as the payload of postgresql traces. We could look into compression issues in the future.

3. More secured traces for HTTPs, MonetDB, DB2, Redis and Memcache

Note:

1. In database & cache configuration, some database may have problem with concurrent queries. The code with \*\_serialize in name is to have serialized queries.

2. In Spark AWS setup, remember to check the leader/follower status before run the application to ensure which is the first master.( It could be different through the trials)

3. In Cassandra AWS setup, use public ip for seeds/listen\_address ip instead of private ip in cassandra.yaml file.

4. There’s a parameter ‘-s –threads 50’ to run 50 threads. If that doesn’t work, you can run 50 processes from the command line.

5. Another interesting observation for Spark AWS setup is that there exist 9866 -> some port communication within worker1 and worker2 separately, which I suspect is the data transformation between hdfs and spark.

6.YCSB doesn’t support MonetDB officially, but I wrote one. It’s in my github repo.

<https://github.com/SeverinaZheng/YCSB-Monetdb-binding>