Project Report

Harald Ng
Chuan Su

October 27, 2019

In our project, we have implemented friends recommendataion and community detection on given directed graph based on `Pregel` algorithm. The intermediate result calculated from `GraphX-Pregel` are sent to `Kafka`. On the consumer side we used `spark-stream` to process the messages and write final result to `Cassandra`.

1 Community Detection

For the community detection we used clique-based methods. Cliques are subgraphs in which every node is connected to every node in the clique. In our `Pregel` approach, each vertex in the graph maintains a `Map` data structure as its internal state in which the keys are its neighbor Ids and the value is an array of vertex id associated with its neighbors. Thus, each vertex in the graph maintains a state that not only contains its own neighbors but neighbors of its neighbors. On each super step, each vertex sends out a message containing all of its neighbor Ids extracted from its state - `map.keyset`. Upon receiving incoming messages each vertex will update its state and calculate the cliques it belongs to with `Bron-Kerbosh` algorithm. Vertices `vote to halt` when the number of neighbors remains unchanged comparing to previous superstep.

2 Friend Recommendations

The aim of this feature was to find vertices that are not adjacent but connected and at most `X` hops from each vertex. Using the `Pregel API`, a forwarding algorithm was used. Each vertex would have a state of a set of friend recommendations, a message queue and a set of neighbors. Each vertex would in the first superstep send a message of `(selfId, 0)` to its neighbors. The second element of the tuple represent how many hops this message has been passed. In the rest of the supersteps, each vertex would check if the number of hops of the message is 0. If so, the id of that vertex is stored as a neighbor. If not 0 and also not already in the `friend-recommendations` set, then put it it there. All messages that a vertex received would be forwarded to its neighbors with the hops element incremented with 1. To restrict the recommendations to be within `X` hops, the program would only run for `X+1` supersteps.

3 Kafka, Spark Stream and Cassandra

In each super step, the resolved friends recommendation and cliques are published to Kafka broker. On the consumer side `Spark Stream` was used to process messages based on topics. For the community topic, spark stream maintains its state of the cliques count and the clique that has maxium peers. For the friend recommendations topic, the average number of hops and the number of friend recommendations would be stored along with the vertex ids. The data would be then written to the corresponding Cassandra table.

4 Dataset

The dataset we used describes Twitter followership and can be found under the project directory `dataset/twitter`.

5 Run the program

Our program is composed of two parts, one is the `GraphX Pregel` application acting as message producer to Kafka and the other one is `Spark Stream` application acting as topic consumer. In order to run the program, you need to have `Spark` installed and `Kafka` and `Cassandra` up and running.

1. Start Zookeeper, Kafka and Cassandra

```
// start zookeeper
$KAFKA_HOME/bin/zookeeper-server-start.sh $KAFKA_HOME/config/zookeeper.properties

// start kafka server
$KAFKA_HOME/bin/kafka-server-start.sh $KAFKA_HOME/config/server.properties

// create topics friend_recommend and community
$KAFKA_HOME/bin/kafka-topics.sh --create --zookeeper localhost:2181
--replication-factor 1 --partitions 1 --topic friend_recommend

$KAFKA_HOME/bin/kafka-topics.sh --create --zookeeper localhost:2181
--replication-factor 1 --partitions 1 --topic community

// start cassandra
$CASSANDRA_HOME/bin/cassandra -f
```

2. Run the consumer application

```
cd sparkstream
sbt clean package
sbt run
```

```
// you have to select one of the main class to run
[1] id2221.CommunityConsumer
[2] id2221.FriendRecommendConsumer
// press 1 or 2 and then [Enter] to run the application
```

3. Run the producer application

```
cd sparkstream
sbt clean package

// start friends recommendation Producer
$SPARK_HOME/bin/spark-submit --class "id2221.FriendRecommend" \
target/scala-2.11/graphx-pregel_2.11-0.1.jar

// start the community detection Producer
$SPARK_HOME/bin/spark-submit --class "id2221.CommunityDetect" \
target/scala-2.11/graphx-pregel_2.11-0.1.jar
```

   note: Running the producers require adding the paths to the `kafka` and `kafkaclient` jar files using the flags `spark.driver.extraClassPath` and `spark.executor.extraClassPath`

4. Check the results in Cassandra

```
// start the cqlsh prompt
$CASSANDRA_HOME/bin/cqlsh

use id2221_space;
select * from friend_recommend;
select * from community;
```

   The results should look like the following:

vertexid	avg	count	recommendations
2	1	4	5, 11, 4, 6,
3	1	6	1, 7, 10, 6, 6, 6,
7	1	6	1, 3, 10, 5, 4, 11,
4	1.33333	6	2, 11, 7, 11, 1, 10,
10	1.66667	6	3, 7, 5, 11, 4, 6,
11	1.25	8	5, 4, 7, 2, 5, 4, 1, 10,
5	1.33333	6	2, 11, 7, 11, 1, 10,
6	1.33333	6	3, 3, 2, 3, 1, 10,
1	1.66667	6	3, 7, 5, 11, 4, 6,

Figure 1: Friend Recommendation Result

vertex_id	count	max	peers
1683	1	1	6941
21197	1	4	2871,17657,15369,15371
151	1	2	153,156
1529	1	2	543,544
20423	1	1	20429
4517	1	1	6418
11435	1	2	1409,1410
18845	1	1	18889
2473	1	1	2476
4103	1	1	4104
6161	1	1	6440
4885	1	1	3873
17769	1	1	17779
10877	1	4	10917,7609,10735,1326
22491	1	1	22498
19185	1	1	19186
17975	1	1	17979
17657	1	3	15369,2871,15375
613	1	1	640
16061	1	1	16062
2515	1	1	2135
19011	1	2	20870,20860
12807	1	1	12821

Figure 2: Community Detection Result.