Publish/Subscribe over Geo-Textual Object Stream

Project Report for CS6203

Yi Li
A0120030
a0120030@u.nus.edu

Xianyan Jia
A0123810
jiaxiany@comp.nus.edu.sg

ABSTRACT

In this project, we develop a publish/subscribe application and use it to annotate geo-textual POIs with Tweet streams. We index the POIs as quad-tree and partition the POIs into subsets using Zorder curve. Our system is built in storm system and deployed in distributed clusters. We conduct experiments on real-world dataset to show the efficiency and scalability of our system.

1. INTRODUCTION

Publish/subscribe (pub/sub) is a many-to-many communication model that directs the flow of messages from senders to receivers based on receivers data interests[1]. In the meantime, massive amount of data that contain geographical location information and have rich textual information is being generated in high speed and large scale.

These geo-textual objects come as streams, e.g. twitter stream, each tweet can explicitly be tagged with geographical coordinate or contain the location name in the text. There are many useful applications for publish/subscribe system when considering geo-textual features. For example, in social network like Facebook, some people may only care about news or events that happen near them. So related information that within a area near user will be more attractive. Another example is about the annotations for POIs. POI providers or the owners have interest to annotate each POI with social content like Twitter or Facebook post. These information shows the popularity or fresh news about POI.

[2] is a recent publish/subscribe system, but it is based on centralized index which can not make good use of distributed clusters. In this project, we build a publish/subscribe system that support parallel processing.

2. PROBLEM STATEMENT

Given a set of geo-textual POIs(subscribed contents) Q and a stream of tweets W, the system continuously publish tweets to POIs over time such that tweet text is relevant to POIs, their locations are close to POI location, and they are fresh.

Specifically, for each POI $q \in Q$, $q = \{k, \rho\}$, where $q.k$ is POI text and $q.\rho$ is POI location. For each $w \in W$, $w = \{k, \rho, t\}$, where tweet is composed of tweet text $w.k$, location $w.\rho$ and create time $w.t$.

To determine whether the tweets coming should be pushed to POIs, we need to rank them by considering three factors:

1. **Location proximity**

$$S_p(w.\rho, q.\rho) = 1 - \frac{dist(w.\rho, q.\rho)}{dist_{max}}$$

The spatial similarity score is calculated by the normalized Euclidian distance, where $dist(w.\rho, q.\rho)$ is the Euclidian distance between w and q, and $dist_{max}$ is the maximal possible distance in the spatial area.

2. **Text similarity** We use cosine similarity to measure the text similarity. The general idea is to first convert text to TF-IDF vectors. And then compute the cosine of the angle between them. There are many other ways to measure the text similarity, e.g. hamming distance, edit distance, etc.

3. **Time freshness** We are also concerned about the time freshness of the information. The recency of the object o is calculated by the following exponential decay function:

$$S_t(o.t, t_c) = D^{-(t_c - o.t)}$$

where D is the base number to determine the rate of the recency decay. t_c is the time when the system decay the score. Based on the experimental studies[3], the exponential decay function has been shown to be effective to combine text and time.

Score Function: We integrate the three factors and get the score function to determine whether to publish tweets to POIs:

$$S = [\alpha * S_p + (1 - \alpha) * S_w] * S_t$$

3. FRAMEWORK

The framework of our publish/subscribe system is shown in Figure 1. The system has two data input streams: POIs and Tweets. When a new POI is added, we insert it into the POI index, and initialize the publish content with past tweets. When a new Tweet comes, it is emitted to SearchBolt to update related POIs. In the meantime, the tweet is added to the index. Once the published content for each POI is updated, they are pushed to the target POI. In the next subsections, we will describe each part in detail.

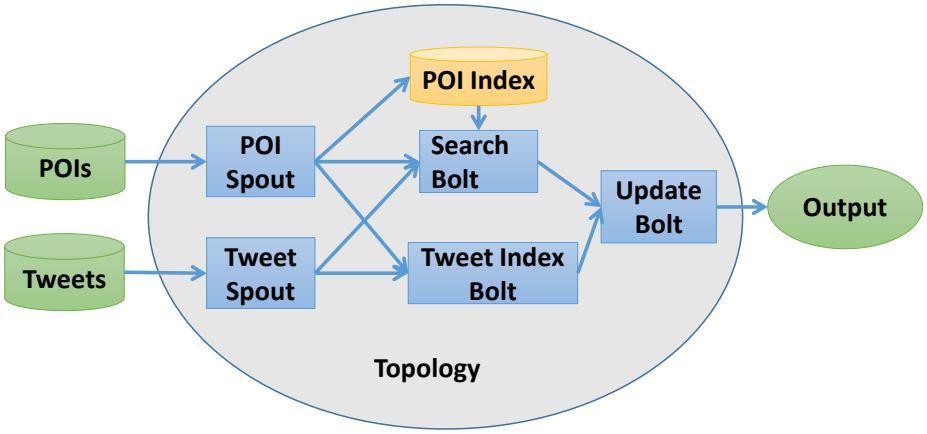

Figure 1: Framework of our publish/subscribe system

3.1 Spout

The system has two types of data sources: POIs and Tweets. Both of them come in the form of stream. To distribute each tuple to corresponding sub-areas, we will first build a bounding box for each geo-object. And calculate the Z-values for the boundary box corners. Because for each sub-area, we keep the min-max z-values, if current z-value is in the range of certain sub-area, this geo-object will be emit to corresponding tasks.

3.2 POI index

The purpose of the POI index is to group POIs that is near in location. When a new tweet come, we fill first find those POIs that are near the tweet. These POIs are the targets that may be updated.

We use quad-tree[4] to index POIs. The reason is that Quad-tree is more suitable for update-intensive application. A quad-tree is a data structure used to divide a 2D region into more manageable parts. It's an extended binary tree, but instead of two child nodes it has four. Another reason is the simplicity of quad-tree. Because it is composed of a set of squares, it is easy to check the overlap area for a boundary box.

Further, instead of building a big centralized quad-tree. We split the tree into a set of sub-trees. The basic idea is that we partition the whole area into a set of small areas based on the distribution of POI points. And then build a quad-tree for each sub-area. There are several advantages for area partition:

- We can process the tweets in a parallel way
- The tree depth is shortened due to clustering of location points, as a result, the search time is expected to be reduced

To partition the area into a set of sub-areas, we adopt Zorder curve[5] to solve this problem. Zorder is a function which maps multidimensional data to one dimension while preserving locality of the data points. The z-value of a point in multi-dimensions is simply calculated by interleaving the binary representations of its coordinate values. Once the data are sorted into this ordering, any one-dimensional data structure can be used such as binary search trees. The resulting ordering can equivalently be described as the order one would get from a depth-first traversal of a quad-tree.

In our system, all POI points are first converted to Z-values. These Z-values are sorted and then equally partitioned. Points in each partition are built as a quad-tree. For each tweet in the stream, we construct a boundary box whose center is tweet location point. We convert the boundary points into z-values and find their corresponding matching sub-tree. In this way, we distribute the tweets to their near sub-area only.

3.3 Search Bolt

The search bolt receive the tweets emitted from Tweet Spout. The number of tasks in search blot is the same as the number of area partitions. So in this way, each partition can be processed independently and in parallel. When preparing the tasks in search bolt, we build the POI index for each sub-area.
When the task receives a tweet, it will search the near POIs for this tweet. But both tweet location and POI location are point in the map, how can we filter and find near locations for them? The solution here is to use a influence distance and build a boundary box for each tweet location point. So to find the candidate POIs, we start from the root of the quad-tree, go down to find its overlapping leaf nodes. All POI points that are within the boundary box will become the candidate targets to be checked and updated. These found POIs will be sent to Update Bolt for further checking.

3.4 Update Bolt

The update bolt keep the top-k result windows for each POI. It receives tuples from two sources: search bolt and Tweet index bolt.

If the tuple is from Search Bolt, it will compute the similarity score for POIs and tweets. If the similarity score is larger than k^{th} result in result windows, then the score of tweets in current windows will be decayed, and new tweet is inserted into the top-k windows. If the new result is not inserted, there may be some reasons: the text similarity is too small or the current results are still really fresh and we don't want

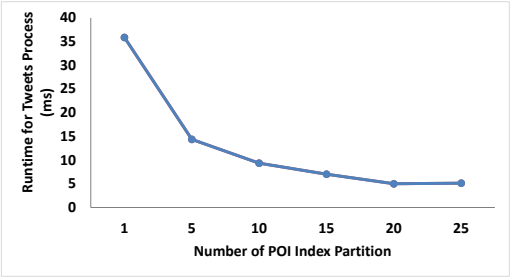

Figure 2: The effect of POI partition# to tweet process time

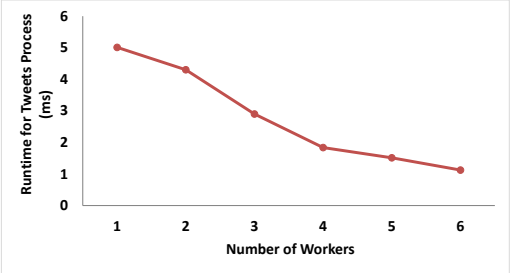

Figure 3: The effect of worker number to tweet process time

too frequent update.
If the tuple is from Tweet Index Bolt, this means this is a new inserted POI. For these new POIs, we will query the Tweet Index Bolt for cold-start information. And the top-k result windows is initialized with indexed recent relative near tweets. The tweet index structure used here is from [6].

4. EXPERIMENT

4.1 Dataset

Our experiments are conducted on real-life dataset collected from Twitter and Foursquare. POI data contains 1.1 million worldwide POIs with both location and text. Table 4.1 shows the example POI data. And Tweet data contains 40 million tweets with geo-location. Table 4.1 shows the example format of tweet data.

4.2 System Configuration

The system is deployed in the distributed clusters "epic". The system details for the server machine is shown in Table 3. And the storm configuration detail is shown in Sec. 6.

Setting	Detail
Server	epic.d1.comp.nus.edu.sg
System	Ubuntu 14.04.2 LTS
Memory	8G
Storage	183.21GB
Storm	Version 0.9.5

Table 3: Configuration

4.3 Result and Analysis

In this section, we are going to show the experimental results. Each method runs for 4000 seconds. We set the decaying scale at 0.5. The arrival rate of geo-tagged tweets in

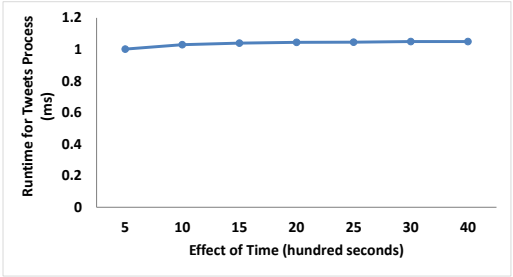

Figure 4: The effect of time to average tweet process time

Twitter is around 100 tweets/second. We report the average runtime for processing a tweet. The total process time is the maximum process time among all tasks.

4.3.1 *The effect of POI index partition*

Here we want to test the effect of number of POI index partition to the Tweets stream process time. As we can see from the Figure 2, with the increase of area partition number, the average tweets process time first decrease and then tend to be steady. The reason is that, each partition corresponds to one task in the storm topology. If we have more partitions, then we can have more tasks being processed in parallel. But there is a trade-off, if we have more partitions, more tasks are performed in parallel and may be limited by the capacity of CPU.

4.3.2 *The effect of worker number*

The Figure 3 shows the effect of worker number to tweet process time. And as expected the runtime for tweets process decrease every time a worker is added into the system. It is clear that the increase of the workers will add more labor to process the tasks in Parallel. So the run time for tweets process will be reduced owing to the effect of parallelism.

4.3.3 *The effect of running time*

Here we observe the process performance of our system overtime by different running time and the result is shown in Figure 4. As we can see, the run time for tweets process is steady as the time passed by.

4.3.4 *The effect of max items kept in Quad-Tree leaf node*

We also perform experiment to see the effect of changing the maximum item numbers kept in leaf nodes. Figure 5 shows the change of running time and Figure 6 shows the memory usage. As all actual data items are stored in the leaf nodes, the larger items number can reduce the tree depth. The less depth of the Quad Tree can reduce tree traversal time as well as memory usage.

Note that, in Figure 5, the runtime for tweets process has been changed to the upward trend with the increase of the Max num items in Quad Tree.

There may be a reasons result in the phenomenon. With the increasing of the Max items in Quad Tree, the number of leaf nodes wiil also be increased. So the worker process should take more time to filter unrelated nodes all, which affects the runtime for tweets process more than the decreasing of

Tweet	Time	Keywords	latitude	longitude
1	2012-08-25 23:00:33	Chenault,PRO,Defense,Martial,Art,Academy,Decatur,AL	34.56164	-87.00443
2	2012-08-25 23:00:33	RT,Team,Follow,Follow,RT,s,RT,Gain,Follower	32.44804916	-92.05424605
3	2012-08-25 23:00:33	pair,beat,ear,bud	41.44897592	-73.47118861
4	2012-08-25 23:00:33	ousted,Nicole,mayor,Enterprise,Car,Rental	29.88726498	-97.91910196

Table 1: Tweet Dataset Example

ID	latitude	longitude	Keywords
1	-2.925304016850944	104.72089733865361	Komp.patra,permai,Medical Center,Medical
2	39.78670863359689	-86.07844593144316	Jady Chinese House,Home (private),Home
3	19.41832179187541	-99.1587585838822	Cafebrera El Pndulo,Bookstore,Bookstore
4	-23.42619686603531	-46.48093342781067	Asa C,Airport Terminal,Terminal

Table 2: POIs Dataset Example

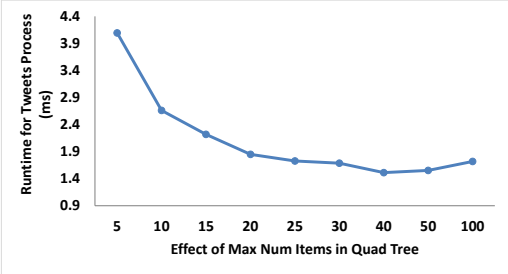

Figure 5: The effect max items in quad tree to tweet process time

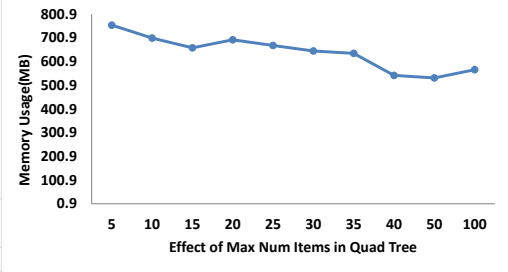

Figure 6: The effect max items in quad tree to index memory usage

the depth of the tree.

5. CONCLUSION

In this project, we develop a publish/subscribe application and use it to annotate geo-textual POIs with Tweet streams. We index the POIs as quad-tree and partition the POIs into subsets using Zorder curve. Our system is built in storm and deployed in distributed clusters. We conduct experiment on real-world dataset to show the efficiency and scalability.

6. APPENDIX

In this section, we will describe how to deploy the storm in clusters[1]. We also present our project in the storm clusters. We deploy the storm on the epic server.

[1]http://storm.apache.org/documentation/Setting-up-a-Storm-cluster.html

6.1 Storm Cluster Deployment

6.1.1 *Node Deployment*

For this experiment, we provisioned 1 physical machine. In the physical machine, the number of Nimbus, Zookeeper, Workers and tasksfor the project in the topology is listed below:

Node	Number
Nimbus	1
ZooKeeper	1
Worker	1-6
Tasks	20

Table 4: Node Deployment

6.1.2 *ZooKeeper*

Zookeeper is used by storm system for coordinating the various actors (e.g. Nimbus, Supervisor daemons) win the cluster. In the experiment, we choose one available node epic to play the role of the zookeeper.

1. Key setting in zoo.cfg
   (a) tickTime=2000
   (b) dataDir=/tmp/
   (c) clientPort=2181
   (d) server.1=epic:2888:3888
2. Save the myid as server.id in the same path as the dataDir.
3. Add the external classpath in the local path ".bashrc"

It is strongly recommended to run ZooKeeper under process supervision because Zookeeper is fail-fast and will exit the process if it encounters any error case. On the plus side a ZooKeeper cluster is self healing: once restarted the failed server will automatically rejoin the cluster without any manual intervention.

6.1.3 *Storm*

Next, download a Storm release and extract the zip file somewhere on Nimbus and each of the worker machines. In our experiment, we set the nimbus node to be the same

node as zookeeper node "epic".
Each worker in a node uses a single port for receiving messages, and this setting defines which ports are open for use. In order to provide enough workers in the experiment, we open 6 ports per node, which means six workers will run on this machine.

1. Key setting in storm.yaml
   - storm.zookeeper.servers:
     - "epic"
   - nimbus.host: "epic"
   - storm.local.dir: "/tmp/storm"
   - server.1=epic:2888:3888
   - supervisor.slots.ports:
     - 6700
     - 6701
     - 6702
     - 6703
     - 6704
     - 6705
2. Add the external classpath in the local path ".bashrc"

6.1.4 Files

Then, the whole project should be unpacked to get the topology jar file. Upload the topology into the project folder. Moreover, the datasets, the dependencies needed in the project and the config file should also be uploaded in the the project folder.

6.2 Storm Setup

The last step is to launch all the Storm daemons as well as the project. Storm is a fail-fast system which means the processes will halt whenever an unexpected error is encountered. All the daemons will be run in the background and the logs will be recorded in the corresponding folder. Here are the instructions to run the Storm daemons:

6.2.1 Zookeeper

Before running all the tasks, it is necessary to run the zookeeper in the epic node firstly. So the zookeeper can monitor and keep track of the woker nodes.

- zkServer.sh start

6.2.2 Nimbus

As the Nimbus and Zookeeper running in the same node in our experiment, the nimbus should be run under supervision in the epic to track the status of the topology.

- storm nimbus r nimbus_boot.log 2 &1 &

The Nimbus will be run in the background as its logs will be recorded in the storm folder.

6.2.3 Supervisor

The supervisor and the worker should be started under supervision on each worker machine.

- storm supervisor supervisor_boot.log 2 &1 &

The supervisor daemon is responsible for starting and stopping worker processes on that machine.

6.2.4 UI

Storm system own providing the webpage for the users to monitor the status of the nimbus, zookeeper, worker nodes and the topology. The UI for the storm can be accessed only in the master node, which is epic in our experiment.

- storm ui ui_boot.log 2 &1 &

The UI can be accessed by navigating your web browser to http://{nimbus host}:8080.

6.2.5 Topology

Now, it's time to submit our topology for top-k keyword searching. The topology should be submit to the Nimbus node "epic".

- storm jar KeywordSerach.jar ks

So the topology can be stored in Zookeeper through the Nimbus. And the source code will be stored in the local disk of Nimbus.

6.2.6 Result

If the processs of the topology is done, the keyword search results can be accessed in the data folder. And the logs for whole nodes: nimbus, supervisor, workers can be viewed in the logs folder in the storm.

7. REFERENCES

[1] Yanlei Diao and Michael J Franklin. Publish/subscribe over streams. In *Encyclopedia of Database Systems*, pages 2211–2216. Springer, 2009.

[2] Lisi Chen, Gao Cong, Xin Cao, and Kian-Lee Tan. Temporal spatial-keyword top-k publish/subscribe. In *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, pages 255–266. IEEE, 2015.

[3] Miles Efron and Gene Golovchinsky. Estimation methods for ranking recent information. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 495–504. ACM, 2011.

[4] Hanan Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys (CSUR)*, 16(2):187–260, 1984.

[5] Guy M Morton. *A computer oriented geodetic data base and a new technique in file sequencing*. International Business Machines Company, 1966.

[6] Dongxiang Zhang, Kian-Lee Tan, and Anthony KH Tung. Scalable top-k spatial keyword search. In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 359–370. ACM, 2013.