Study Propagation of Tweets in Twitter

Shreyas Achrekar (50060123) and Pratik Deshpande (50097067)

Abstract

The main aim of this project was to construct a parallel twitter simulator to study the tweet propagation. The twitter runs on the principle of algorithms which are defined by probability calculations. The simulator created by us does probability calculation in parallel which is one of the biggest computation overhead while simulating twitter. After creation of network we have simulated tweet propagation with Depth First Search (DFS) on the master node to understand the working of twitter. We have observed various results with respect to increase in network size, increase in number of worker nodes and increase in communication overhead.

Introduction

Twitter is an online social networking service that enables users to send and read short 140-character messages called "tweets".

Twitter was created in March 2006 by Jack Dorsey, Evan Williams, Biz Stone and Noah Glass and by July 2006 the site was launched.

The service rapidly gained worldwide popularity, with more than 100 million users who in 2012 posted 340 million tweets per day

The service also handles 1.6 billion search queries per day.

In 2013 Twitter was one of the ten most-visited websites, and has been described as "the SMS of the Internet."

As of July 2014, Twitter has more than 500 million users, out of which more than 271 million are active users.

Our goal is to build a simulator which mimics the model of Twitter and then study how tweets propagate through the Twitter network.

Shreyas is working with Professor DimitriosKoutsonikolas and ShambhuUpadhyaya along with Abhishek Desikan on a similar [1]project which is done in sequential.

They faced many problems while doing the study as the simulator was getting impossible to scale since the main algorithm was running in O(n^2) time.

The main deliverable of this project will be the parallel Twitter simulator which will be able to simulate much more number of nodes (>100,000) which is currently very difficult to simulate using sequential simulators.

This simulator can be used for various purposes such as to study Malware propagation or rumor propagation in Twitter. We can also use it to predict how Twitter will grow in the near future.

The rest of paper is divided into related work, current solutions, the model used to define our network, the dataset used to observe result, the implementation, the results and discussions, future scope and references.

Related Work

All of the work regarding this topic was done in sequential. There is a simulator available online called [2]Netlogo which simulates various conditions.

It has a pre-defined model which is known as preferential attachment which twitter follows.

This was used by a UB alumni for a project which uses Netlogo to simulate a simple Twitter graph. This was one of the first attempts to closely follow the Twitter model by dividing it into two networks, namely, social and informational network.

But Netlogo had various shortcomings which didn’t enable it to depict Twitter in a very close fashion.

Our work is more focused and has a greater resemblance to Twitter than the above mentioned project.

Also, this project represented Twitter as a Tree which is not the case in the real world. Graphs shows more properties that correspond to the Twitter model than Trees.

Current solution

The most recent work that is being done on this subject is by Abhishek Desikan with the help of ProfessorDimitriosKoutsonikolas and ShambhuUpadhyaya.

Abhishek is building a Java based simulator which follows the paper mentioned above with a few additions. Since this simulator is built from scratch and in Java, it offers very flexible options and tunable parameters. The simulator that Abhishek has currently built runs for 100,000 nodes at the maximum which is not at all enough for a realistic simulator.

This is the major shortcoming of his work that we are trying to address using the power of parallel processing.

The model

The main algorithm behind Twitter is the [3] "Who to follow" algorithm. This algorithm decides who we are most likely to follow at any given point. We have this option when we join Twitter too. We are made to follow some "celebrities" which help us to get familiarized with Twitter.

These celebrities are also called as Informational nodes. These are the profiles which gives out a lot of information but doesn't get a lot of information from us. We only form a unidirectional link with such informational nodes.

There are also other nodes which are social. These are our friends or family who form a bi-directional link with a node and exchange data between them.

Thus, twitter as whole can be divided into two networks: Social and Informational.

Whenever a new node joins in, it makes a pre-defined number of social links and informational links. This determines who it follows in the Twitter world.

The "Who to follow" algorithm works on the preferential attachment model. This model states that the probability that a new node joins a particular node is directly dependent on the number of followers that node already has.

Thus, the node with maximum followers has a higher chance of a new node following it than a node which has a handful number of followers. This is the main reason that few people on twitter have millions of followers and most of the others hardly have a thousand or less.

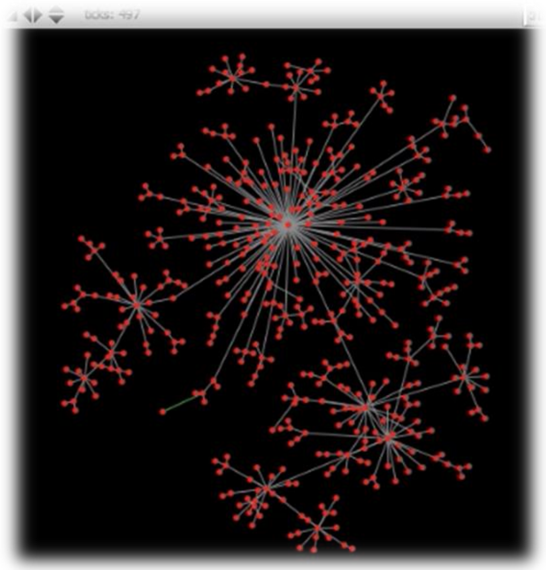

Figure 1: Preferential Attachment

After the network is formed, the propagation of tweet is done using a depth first traversal of the entire network.

At each level, there is a certain probability that a follower will re-tweet the original tweet.

As the depth of the level go on increasing, the probability goes on decreasing. Eventually the tweet becomes stale and nobody re-tweets it.

Data Set

Twitter data was collected over a period of 6 months for this study. We collected more than 10GB of data and over 10 million tweets. The main reason to collect the data was to find out the probability at which a particular user re-tweets a tweet.

We can also see how much it impacts if a celebrity tweets and check if our simulator behaves such a trend.

[4]Twitter4j was used to gather the data. We used to capture streaming tweets and store it in a text file. Twitter4J is an open-sourced, mavenized and Google App Engine safe Java library for the Twitter API which is released under the Apache License 2.0

[5]Twitter API has a rate limit which are divided into 15 minute intervals. There are two initial buckets available for GET requests: 15 calls every 15 minutes, and 180 calls every 15 minutes. Hence, it takes a lot of time to gather significant amount of data.

Then a python script was used to parse the data and get the probabilities.

Implementation of Twitter simulator

1. Sequential

There was a Node class which has properties related to a particular node. It was as follows:

```
class Node {
ntnodeId;                  //Unique node-id
char *userName;//Username of Node
char *handle;                        //Twitter handle
intsocialCount;//Number of social links
intinfoCount           //Number of info links
doubleinfoProb;//Probability of forming
information link
doublesocProb;//Probability of forming social
link
vector<int> following; //Vector of nodes that its
following
vector<int> followers; //Vector of nodes that are
following it
}
```

An initial network was formed which was a fully interconnected network. This helps in making a group which can represent celebrities or important figures on the social media platform.

After this stage, a node is added at every time *t.* When a node joins, we pick up a random number between 0 and 1. Then we check in which node's range that number falls.

The node that is selected is chosen to be followed by the current node if it is already not present in the follower's list. Hence, larger the probability of the nodes, the more range is will cover between 0 and 1.

This allows us to maintain the preferential attachment model. Whenever a new node joins, the probability of the entire network has to be re-calculated.

This creates a huge overhead which is the main cause of this algorithm taking so long to run. Thus this program runs in n^2 time which causes huge delays as the network goes on increasing.

2. Parallel

We are doing two implementations of the parallel model. The first model uses the same algorithm as used in the sequential code but it does so on multiple processors.

Thus we divide the entire network into chucks on network and divide them onto the CCR clusters. Thus, after the network is fully formed, there will be equal number of twitter nodes corresponding to each CCR node.

This helps us in parallelizing the probability calculation since each CCR node will calculate the probability of its cluster simultaneously.

We have a master slave configuration in the parallel implementation in which the master chooses which nodes to be followed by the current incoming node. Then a MPI message is sent to all the processors telling the choice the master has made.

Once each processor receives the message, it checks if it is one of the nodes that is now followed by the new incoming node. If so, it increases its in degree by one. After all the followers are properly mapped, we update the probabilities of all the nodes and send the updated probabilities back to the master so that it can take a decision based

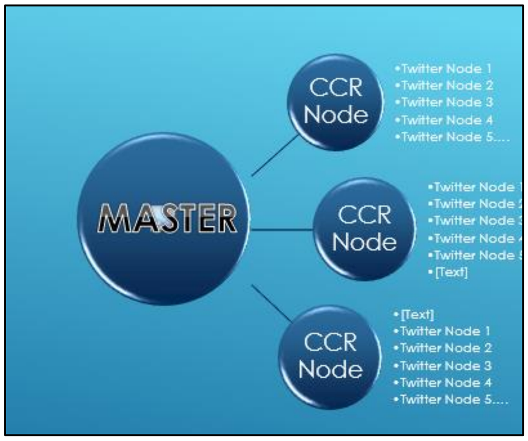

Figure 2: Graphical Depiction of Implementation

on these new probabilities for the next incoming node.

Here the job of the master is to build a map of all the probabilities that it recieves from its slaves. And on the basis of that map, the master chooses a node to be followed for the next incoming node.

Since there is not much work for the master to do, the master does not become a huge bottleneck which is the general problem in a master slave configuration.

But after running this configuration for large values of network size (>50,000) we found out that there is a lor of overhead while doing MPI send and recieves for each and every incoming node. We also found out that the probabilities of the nodes do not change drastically when each node is added.

Keeping these two things in mind, we came up with an alternate implementation in which, we send the updated probabilites to the master after a certain number of nodes are added.

This modification will make the accuracy of the model a little lesser than the actual one but the execution time of the program gets reducesd drastically. Thus, by this model, only when we send the probabilites to the master, we will require a large amount of time to do the communication.

For example, if we do MPI send and recieves after every new node joins, it takes about 8 seconds for each node to join after we reach 30000 nodes. In the new implementation, we take a user input as to how much step size he wants. If the user decides that accuracy is more important than execution time, then he can put step size as one. Or else if we want the execution to be quick after sacrificing a little accuracy, we can put the stepsize to as big as 1000.

The main issue here is to synchronize the master and the slaves as they should send and receive at the exact same time. If synchronization is not done properly then one of them stops execution and blocks itself since MPI_SEND and MPI_RECEIVE are blocking calls.

Once this synchronization is handled, this implementation is similar to the first one.

3. Tweet propagation

Once the network is formed, we move on to tweet propagation. Tweet propagation is done on the basis of the probabilites of re-tweet.

After studying the data, we found out that one out of three followers re-tweet the original tweet in the first level.

As the level goes on increasing, the probability goes on decreasing. Hence to make the calculation easy, we make the probability of re-tweet (0.33^depth) at each level.

Result and Discussion

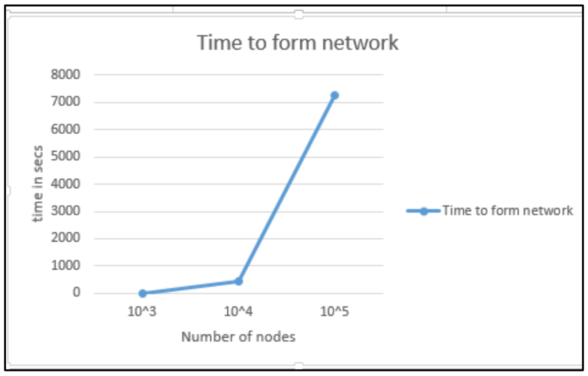

Figure 3: Running time for sequential

As we can see from the above graph, the running time of this algortithm increases exponentially for larger values.

It is almost impossible to create a network of more than 100000 nodes since creating a network of that size itself takes more than two hours. Thus, for running the simulator for higher values of node size, we need a parallel simulator which can scale to atleast a million nodes.

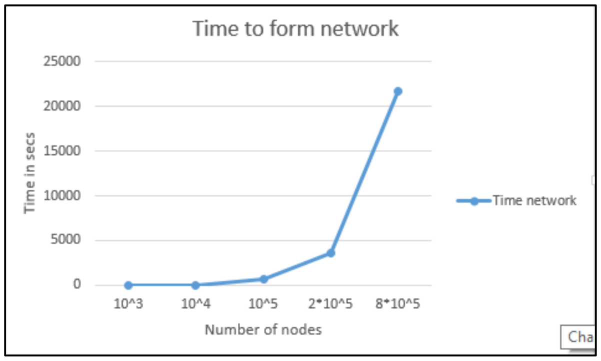

Figure 4: Running time for parallel

As we can see from the above figure, the parallel simulator runs must faster than the sequential one. It takes about 10 minutes for a network size of 100000 and a step size of 100. The step size is an important factor in this implementation. Since the MPI sends and receives are a huge bottleneck, we have to find a sweet spot for the step size.

Also, by looking at the graph, we may think there is a super linear speed up in the parallel implementation. But that is not the case. The reason we are getting such a good speedup is due to the fact that while doing in parallel, we are not making a Node class as we did in the sequential. Each and every thing is done using vectors on the master and each worker has to just deal with its own small divided network.

Thus, even though the underlying algorithm remains the same, the entire implementaion as a whole is very different than that of the serial implementation. Hence, we are able to create such a huge network.

As per our guestimation, we should be able to create a network of a million nodes in about eight hours. And considering that the time it took for sequential program, this simulation does very well to reach million in eight hours.

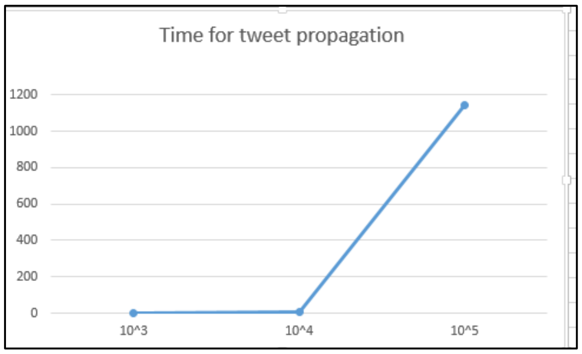

Figure5: Tweet propagation

The above figure shows the time it takes for tweets to propagate throughout the network. As per the figure,we can see that the time increases exponentially when the network size grows. This is a logical result since as the network size increasing, more and more number of users re tweet a particular tweet.

Hence, there are a lot of re tweets of a particular tweet. Also, as observed during the running of these programs, if the initial network size of the network is large, the tweet reaches a lot of users.

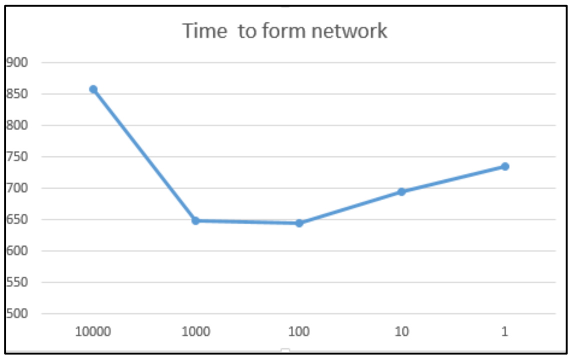

Figure 6: Effect of step size on running time (10^5)

The step size was a very important modification done to the program. As we can see from the figure above, the running time reduces significantly if we make the step size to be 100. There is almost 20% increase in the speedup. This value becomes very significant when the network goes beyond 10^5.

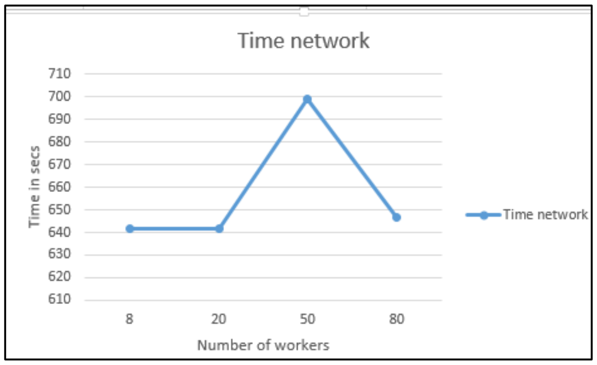

Figure 7: Effect of number of workers (10^5)

Here we can see the effect of increasing the number of workers keeping the size of network same. As we can see, there is not much significant difference in the running times. This happens due to the fact that we are not increasing the problem size. Hence the sequential part of the problem will be similar in all the cases which cause the running time to be the same.

Future Scope

The tweet simulation is currently sequential. We can parallelize that to obtain results in a more twitter like environment. Also the implementation will eventually face problems due to communication overhead as we go on increasing the network size. This is due to the fact that mapping of probabilities is done sequentially by the master. This can be handled by using more than one master and keeping proper synchronization between them.

References

[1]Twitter Structure as a Composition of Two Distinct Networks-Meng Tong, Ameya Sanzgiri, Dimitrios Koutsonikolas and Shambhu Upadhyaya Computer Science and Engineering, University at Buffalo, Buffalo, New York 14260{mengtong, ams76, dimitrio, shambhu}@buffalo.edu.

[2]https://ccl.northwestern.edu/netlogo/

[3]https://blog.twitter.com/2010/discovering-who-follow

[4]http://twitter4j.org/en/index.html

[5]https://dev.twitter.com/streaming/overview