Scalable Processing of Call Detail Records for Community Detection

Md. Nasim (0905018), Sharowar Md Shahriar Khan (0905053)

Problem Definition:

- A large volume of call detail records (CDR) is generated everyday from millions of phone calls
- These CDR data can be used to reveal vital information about an individual or a community
- Processing this huge volume of data for extracting any specific information, in particular, detection of different communities is a challenging task

In this project, we provide a scalable approach to process a huge volume of CDRs and identify different communities.

Background and Motivation:

- **Applications of community detection**
  - Provide insight into how users in a mobile network interact with each other
  - Recommend suitable packages to users
  - Identify possible churns in the network
- **Analyzing call detail records**
  - Each day's call log contain around **32 million** records
  - For one month, size of the call log is around **76 GB**
- **Limitations of existing approaches**
  - Do not exploit the full potential of existing big data frameworks [1-3]

Our Approach:

- Split the data file containing call detail records of one month into smaller files based on the date of call i.e., one file for each day
- Generate graphs for each day's call logs and store them in a distributed graph database
  - the graph contains 2 types of nodes- User and Call
  - From each call node, 2 edges exist toward the associated pair of users

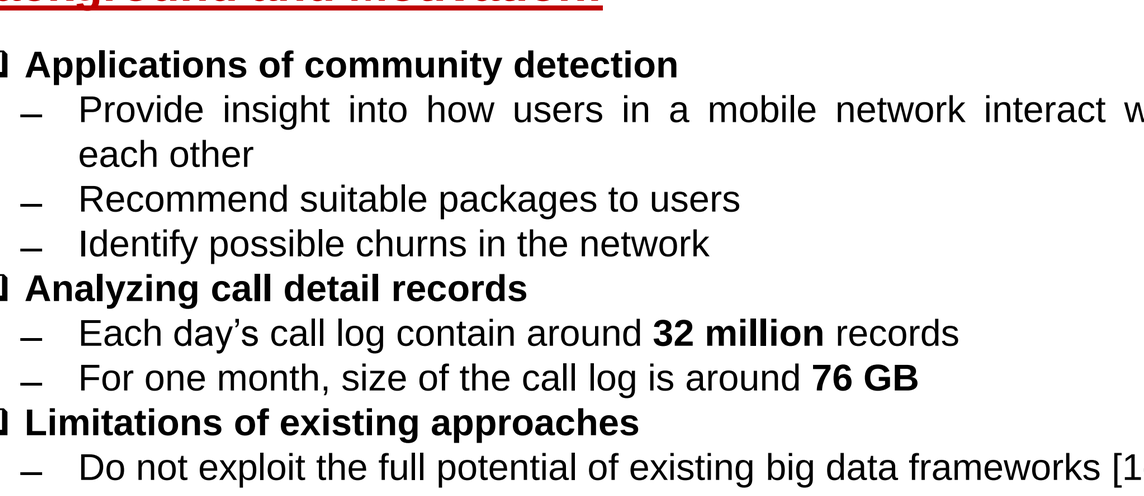


Figure 1: A sample of the generated graph

- Process the generated graph for community detection

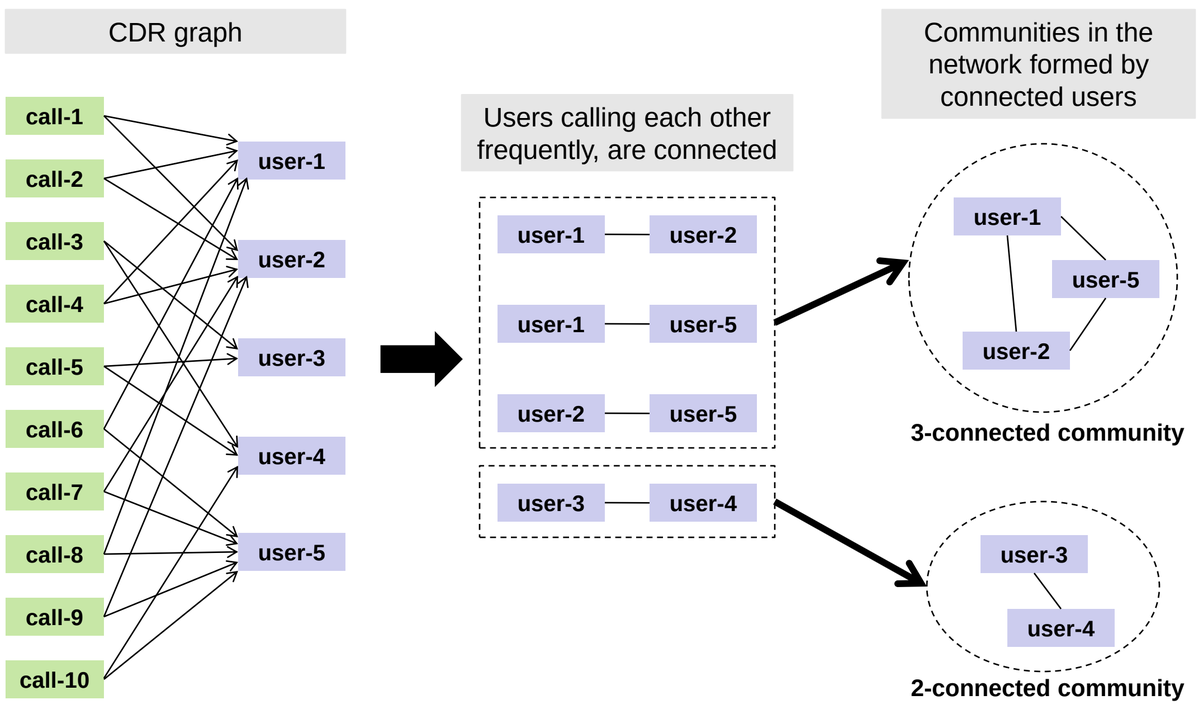


Figure 2: Community detection from call detail record graph

Tools Used:

- **Java**
  - To split the large data file into smaller ones based on time
  - To preprocess data records, before inserting into database
- **Titan Graph Database**
  - To store call detail records as a graph
- **Apache Cassandra**
  - To maintain **distributed storage backend** for Titan
- **Rexster Graph Server**
  - To **visualize** the generated graph
- **Gremlin Graph Traversal Language**
  - To execute graph **queries**

System Description:

- A **distributed graph database** is deployed in a cluster of 4 machines. Each machine is connected to each other through a high-speed (100 Mbps) LAN
- The deployed system is horizontally scalable. As new machines are added, the storage capacity of the system increases

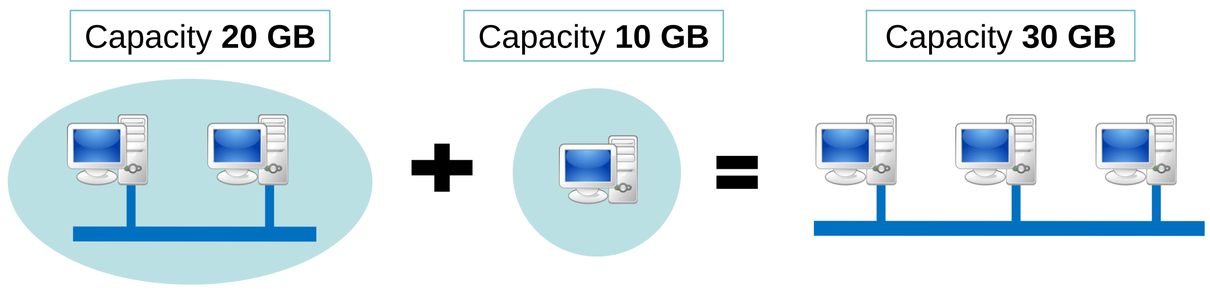
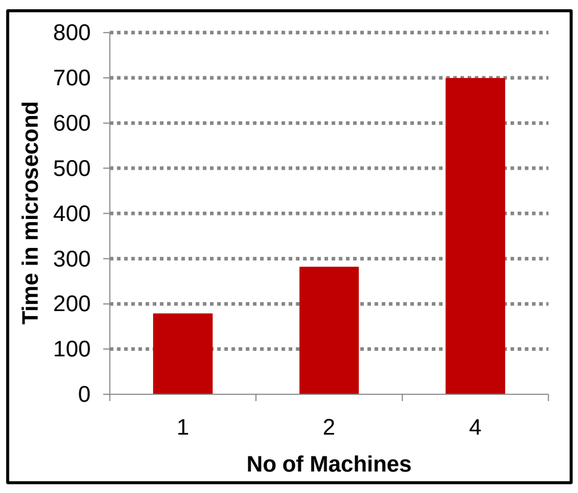
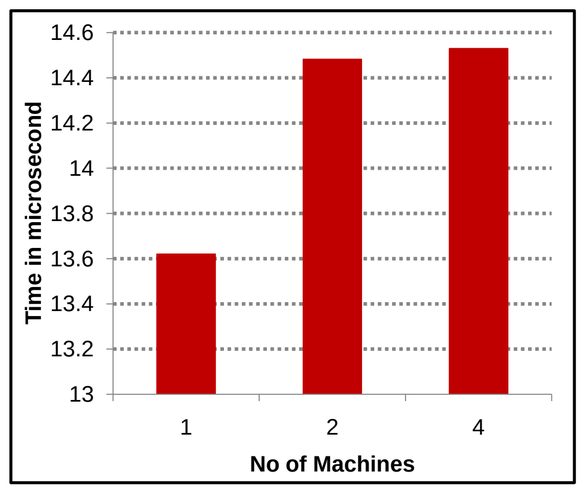


Figure 3: Horizontal scalability of the deployed system

- As more machines are added to the cluster, average query time remains same. However, average insertion time increases



(a) No of machines vs average insertion time.



(b) No of machines vs average query time.

Figure 4: Experimental results for varying no of machines in a cluster

Future Work:

- Optimize the deployed system for reducing the pocessing time
- Develop a novel, scalable algorithm for community detection from the generated CDR graph

References:

1. *Kolda, Tamara G., et al. "Counting triangles in massive graphs with MapReduce." SIAM Journal on Scientific Computing 36.5 (2014): S48-S77.*
2. *Cui, Wen, Guoyong Wang, and Ke Xu. "Parallel Community Mining in Social Network using Map-reduce." International Journal of Advancements in Computing Technology 4.15 (2012): 445-453.*
3. *Simmen, David, et al. "Large-scale graph analytics in aster 6: Bringing context to big data discovery." Proceedings of the VLDB Endowment 7.13 (2014).*

Department of Computer Science and Engineering (CSE), BUET