Constructing a Social Network Analysis System for SIGMOD 2014 Programming Contest
VIDA Team: Fernando Chirigati\textsuperscript{1}, Kien Trung Pham\textsuperscript{1}, and Tuan-Anh Hoang-Vu\textsuperscript{1}
Supervised by Huy T. Vo\textsuperscript{1,2}

1. NYU Polytechnic School of Engineering | 2. Center for Urban Science and Progress (CUSP)

**TASK**

- The goal of the contest is to construct a social network analysis system that can execute a set of queries as quickly as possible.
- At total, there are four (4) types of queries that must be handled.
- Queries are given in text as an input to the system.
- Results are printed as standard output.

**Data**

- The datasets (a total of 31) come from the LBBC social network benchmark generator: https://github.com/lbcb/lbcc
- Different social network sizes are tested: small (1K persons), medium (10K persons), large (100K persons), and huge (1,000K persons).

**Evaluation Environment**

- Processor: 2.67 GHz Intel Xeon E5430
- Configuration: 2 processors (8 cores at total)
- L2 Cache Size: 12 MB
- Main Memory: 15 GB
- Operating System: RHE Linux Server 6.5
- Compiler: GCC 4.4.7

**QUERY PRE-PROCESSING**

- Input queries may come in any order.
- System internally sorts queries by query type.
- Repeated queries are detected and processed only once.
- Original order is preserved in the output.

**I/O AND DATA STRUCTURES**

- We use memory mapped files from Boost to improve the I/O performance for large files.
- Whenever possible, static allocation, rather than dynamic allocation, is used.
- Size of a data structure is estimated based on the size of the dataset file to pre-allocate memory.
- Arrays and vectors are preferred over maps and sets.
- Graphs are constructed using a compact adjacency list representation.

**MULTITHREADING STRATEGY**

- We use Boost to support multithreading in the system.
- We focus on using inner multithreading (i.e.: inside each query type) rather than outer multithreading (i.e.: across different query types).
- Initializing data structures for query type 1 is a bottleneck.
- It provides a better use of resources.
- Evaluation environment has 8 cores at total.

**THE MS-BFS TECHNIQUE**

- To speed up BFS computations, we have developed a technique that takes advantage of SIMD instructions and bit operations to efficiently execute multiple BFS concurrently in a single thread - the Multiple-Sources BFS, or simply MS-BFS.
- Given a fixed graph, multiple BFS can be executed concurrently without the need of a locking mechanism or multiple threads.
  - Bit masks and bit operations are used for efficiency.
  - 64-bit mask: 64 concurrent BFS.
  - Time complexity for 64 BFS is O(E|V|) for dense graphs, while the usual algorithm takes O(2E+V|V|) for a single BFS.
- Since queues may overlap, vertices are "sharded" and visited only once for multiple concurrent BFS.

**QUERY TYPE 1**

Shortest Distance over Frequent Communication Paths
Given two person ids p1 and p2, and an integer x, find the minimum number of hops between p1 and p2 in the graph induced by persons who: (i) have made more than x comments in reply to each other's comments, and (ii) know each other.

**QUERY TYPE 2**

Interests with Large Communities
Given an integer k and a birthday d, find the k interest tags with the largest connected component in the graph induced by persons who: (i) have that interest tag, (ii) were born on d or later, and (iii) know each other.

**QUERY TYPE 3**

Socialization Suggestion
Given an integer k, a maximum hop count h, and a place name p, find the top-k pairs of persons based on the number of common interest tags. For each of the k pairs, the two persons must be located in p, or study or work at organizations in p. Furthermore, these two persons must be no more than h hops away from each other.

**QUERY TYPE 4**

Most Central People
Given an integer k and a tag name t, find the k persons who have the highest closeness centrality values in the graph induced by persons who: (i) are members of forums that have tag name t, and (ii) know each other.

**THEORY**

- The basic idea is to run BFS for each query.
- Number of comments between two persons is added as a weight to the corresponding edge in Persons Graph.
- Queries are grouped by x so that the graph can be incrementally reduced, which improves the performance of the BFS.
- BFS threads are used for query type 1, and each thread executes one MS-BFS.
- 512 queries can be executed concurrently.

- The goal of the contest is to construct a social network analysis system that can execute a set of queries as quickly as possible.
- At total, there are four (4) types of queries that must be handled.
- Queries are given in text as an input to the system.
- Results are printed as standard output.

**DATA**

- The datasets (a total of 31) come from the LBBC social network benchmark generator: https://github.com/lbcb/lbcc
- Different social network sizes are tested: small (1K persons), medium (10K persons), large (100K persons), and huge (1,000K persons).

**EVALUATION ENVIRONMENT**

- Processor: 2.67 GHz Intel Xeon E5430
- Configuration: 2 processors (8 cores at total)
- L2 Cache Size: 12 MB
- Main Memory: 15 GB
- Operating System: RHE Linux Server 6.5
- Compiler: GCC 4.4.7

**QUERY PRE-PROCESSING**

- Input queries may come in any order.
- System internally sorts queries by query type.
- Repeated queries are detected and processed only once.
- Original order is preserved in the output.

**I/O AND DATA STRUCTURES**

- We use memory mapped files from Boost to improve the I/O performance for large files.
- Whenever possible, static allocation, rather than dynamic allocation, is used.
- Size of a data structure is estimated based on the size of the dataset file to pre-allocate memory.
- Arrays and vectors are preferred over maps and sets.
- Graphs are constructed using a compact adjacency list representation.

**MULTITHREADING STRATEGY**

- We use Boost to support multithreading in the system.
- We focus on using inner multithreading (i.e.: inside each query type) rather than outer multithreading (i.e.: across different query types).
- Initializing data structures for query type 1 is a bottleneck.
- It provides a better use of resources.
- Evaluation environment has 8 cores at total.