On Several Social Network Analysis Problems

© George Chernishev © Vsevolod Sevostyanov © Kirill Smirnov
Saint-Petersburg University, Russia
chernishev@gmail.com vsevost@gmail.com kirill.k.smirnov@math.spbu.ru

© Ilya Shkuratov
Saint-Petersburg University, Russia
shkuratov.ilya@gmail.com

Abstract

In this paper we describe our approach to several problems offered at the ACM SIGMOD Programming Contest 2014. These problems belong to the area of a social network analysis and involve several types of queries to a social graph. The considered graph is modeled by the standard SNB benchmark. We briefly introduce this benchmark, the contest and the problems. Next, we describe our contribution, which is the following: the algorithms for evaluation of these queries and their efficient implementation. Furthermore, we present parallelization techniques for these algorithms and describe overall architecture of our solution.

1 Introduction and Related Work

In this paper we study several problems offered at the ACM SIGMOD Programming Contest 2014 [1], a yearly programming contest focused on a data management topics.

This contest has a number of features, which distinguish it from a well-known ICPC series:

- Participants are offered some science-intensive task, which is usually an unsolved problem of current importance.
- The contest runs for several months and no on-site participation is required.
- Topic specificity — the clear data management focus is present. For example, contests of previous years involved construction of distributed query processing engine (2010), multidimensional index (2012) or document stream filtering system (2013).
- The participation is allowed to both graduate and undergraduate students, without any restriction on a number of attempts.

While this contest is not so well known as the ICPC, it is nevertheless popular. For example, last year there were more than 100 registered teams. The contest is relatively young — it runs for 6^{th} time this year.

In this paper we also describe the contest: the rules, the task, its timeline and required qualifications. Moreover, we present our experiences and provide a solution of the team "GenericPeople" (Ilya Shkuratov and Vsevolod Sevostyanov), which was ranked[1] 17 out of 33 teams on the preliminary (public) tests. While our approach is not the best, it still has merit:

- our solution can serve as an example demonstrating the required qualifications and which may help to assess the required effort and work intensity. These factors may be of interest for a person who is thinking about the participation;
- the solution successfully passed through all available tests (datasets of three different sizes) within the time limits specified by the contest organizers (5 and 10 minutes);
- the proposed algorithms passed all correctness tests;
- parallelization techniques of these algorithms may be of interest;
- the number reported in the leaderboard is the sum over all query types, at the present time we can say nothing regarding their individual performance;
- at last, the number was reported for three datasets; the proposed algorithms may behave differently (better or worse) on another dataset.

Thus, we deem current study as worthy to be presented and of some interest for the reader. Another motivation for this paper is the concise presentation of the

Proceedings of the 16th All-Russian Conference "Digital Libraries: Advanced Methods and Technologies, Digital Collections" – RCDL-2014, Dubna, Russia, October 13-16, 2014.

[1]`http://www.cs.albany.edu/~sigmod14contest/leaders.html`, last accessed 02/05/2014.

solution for the contest problem, which is usually lacking. After the contest all what is left are the posters of the top five performing teams without detailed explanation (it is given orally at the conference). Also, these posters are (or at least were in the past years) not going into the conference proceedings and are kept on a website, which may disappear. Moreover, we present our experiences and describe (at least partially) the way we went through in order to produce a working solution. It is impossible to pass on all these aspects via poster.

This year contest was dedicated to a social network analysis topic. Social network is essentially a graph, whose vertices represent users and edges denote relations between them. An example of such relation may be "know each other", "follow" and so on. Additionally extra information like a place of work or study, geographical information, various tags, images, likes etc. is known.

In the past years massive amounts of such information were made available for analysis, forming a strong incentive for both academy and industry to come with means for its efficient storage and processing. Social data play a significant role in the whole "Big Data" movement.

A lot of analysis tools employ the MapReduce [6] programming model. Industrial examples of such systems are PIG (Yahoo) [13], SCOPE (Microsoft) [4], Hive (Facebook) [19], Dremel (Google) [14]. Academic examples are Starfish [9], HadoopDB [3] and many others[2]. An alternative (which can be considered a poor man's solution) sometimes employed in production environment, is to use scripts written in scripting language like Python to commence the analysis. A data scientist has to analyze the problem and implement all necessary algorithms manually. While it may not favor the rapid development, it may allow to achieve a more efficient processing. Naturally, this approach is more flexible than using a standard tool and allows a fine-tuning of algorithms. However, it requires extensive technical expertise: knowledge of algorithms and data structures, the understanding of the data processing and so on. The tasks of the contest are representative examples of this "manual" approach and can be considered as a training for a data scientist.

Another aspect of the contest task is the graph analysis component. Graph analysis is a mature area of research which studies the efficient storage and processing of graph data. There are several graph database management systems (a special type of DBMS) and graph programming frameworks. These DBMS feature special query languages, query processing algorithms and data storage.

Some examples of the graph DBMS are Neo4j [12], InfiniteGraph [10] and the framework examples are Apache Giraph [2], Signal/Collect [17]. It is necessary to mention that two latter systems also follow the MapReduce model.

The contestants were given the task which consists of the datasets and four types of queries. The social graph was generated using the SNB [16] tool.

The goal was to develop a program which computes the results as fast as possible. The contestants had not only to devise the algorithms for efficient query processing on a large graph, but also to parallelize them. This is a must, given the fact that the evaluation of the resulting implementation was performed on a server-class equipment (8 cores).

Another important aspect was the order of computation for each sub-query. The contestants had to bear in mind the size of intermediate results and the memory bound. In other words, the contestants had to perform the work of a query optimizer: gather needed statistics, assess selectivities and develop an optimal processing strategy for each query type. Also, given the hardware multi-core capability, efficient inter-query type orders are also of interest.

The contribution of this paper is the following:

- The description of the ACM SIGMOD Programming Contest 2014 and its task;
- The contest from the participant's point of view: our experiences;
- The algorithms to handle the problems offered at the contest;
- A parallelization techniques for each of these algorithms;
- A general system architecture: subquery computation orders, inter-query type orders and chunk-based data loading.

Now, we are going to describe our experience. The SNB description and its data schema is presented in the appendix section. Detailed description of our approach and data statistics can be found in the report [5].

2 Contest description and experiences

Let's describe this year contest from the participants' point of view. We have already briefly described the contest and its specifics in the introduction section. You can find detailed information regarding the ACM SIGMOD Programming Contest series in the reference [18].

Our research group is a frequent participant of this contest; we had achieved good results twice in the past: in the 2010[3] (team "spbu") and 2013[4] (team "Rota Fortunae") year. Both times our teams achieved 3^{rd} place in the final ranking.

2.1 General information

This year contest followed the general scheme described in the reference [18]. However, there were several notable divergences:

[2]A list can be found in `http://dl.acm.org/citation.cfm?id=1454166`, last accessed 22/07/2014.

[3]`http://dbweb.enst.fr/events/sigmod10contest/results/#winner`, last accessed 22/07/2014.

[4]`http://sigmod.kaust.edu.sa/finalists.html`, last accessed 22/07/2014.

1. The contest started noticeable later compared to previous years;

2. There were no 2^{nd} round, unlike early years. This change happened in 2013;

3. The absence of the dedicated correctness testing phase during the evaluation (it was performed concurrently with the performance evaluation);

4. There was a series of datasets which were progressively disclosed by the organizers, as the performance of the submissions improved;

5. The task did not explicitly required parallelization or concurrency support, but instead, implied it. It was possible to submit purely sequential implementation;

6. It was possible to submit only the executable, without source code during the preliminary evaluation. The final evaluation required source code and this led to some compatibility difficulties;

7. Contestants were allowed to choose programming languages other than C++.

The provided task was a science-oriented problem related to social network analysis. The problem was to execute a number of queries to a graph representing some social network. The goal was to produce a correct answer and minimize the overall processing time. The graph and queries are fully described in the next section.

Below you can see the timeline of the contest.

- January 25, 2014 — Contest announced.
- February 1, 2014 — Detailed specification of the requirements and test data available.
- February 16, 2014 — A medium data set (10k people) with query workload and answers are available on the Task page. New query workload and answers for the small data set (1k people) are available on the Task page.
- March 1, 2014 — Team registration begins. Leaderboard available.
- March 11, 2014 — Workloads on a medium data set (10k people) have been added to the evaluation system.
- March 17, 2014 — Workloads on a large data set (100k people) have been added to the evaluation system.
- April 15, 2014 — Final submission deadline.
- May 15, 2014 — Finalists announcement.
- June 22-27, 2014 — Conference: announcement of the winner and the poster presentations.

In the overall the contest run for two and a half months. Also you can see that several datasets were progressively added to the evaluation pool. These datasets were progressively disclosed by the organizers as the performance of submissions improved. This is a rather new model of evaluation (appeared in 2013 contest) and it was employed in the following way. As soon as the several submissions were achieving some performance level, where it was hard to discern their quality due to inaccurate measurements (thread scheduling effects, for example), a new, larger dataset was added.

2.2 Communication with the contest organizers

Information about the order and rules of the contest were provided on a special web page [1], which was the main mean of communication between the organizers and the contestants. It also describes test data sets, the task and an evaluation environment. Later opportunities to register a team and submit solutions were added.

The organizers also created a Google Group in order to discuss any technical issues (e.g. code page problems) and to provide additional information that might be of interest to all of the contestants: test data-sets publication dates, disk space availability, size of data set for the final evaluation and so on.

2.3 Required skills and our experiences

Since the organizers of the contest considers Linux as its target platform, we decided to use C++ programming language as it looks to us an highly-optimizable one. Those who want to take part in the contest are advised to learn Linux development utilities such as gcc, make, valgrind (especially callgrind might be useful), gdb, etc. Also two bash scripts were required: one should build the solution and the other — run it with certain parameters.

You also may encounter restriction on size of submitted solution. It was 8 MB this year, thereby it was helpful for us to learn a couple of gcc flags. The first one is *-s*. It removes unneeded symbols from an executable, thus reducing its size without the loss of performance. The second flag may be useful, if you use external libraries: *-MM* instructs the compiler to generate source files dependencies. This helped us to familiarize with boost headers dependencies, strip boost from unneeded header files and further reduce submitted archive size.

Understanding compiler optimization methods may be of use as well. It allowed us to cope with the gcc optimizer bug, namely incorrect copy propagation after global common subexpression elimination pass. It leads to usage of the original pointer to the buffer instead of its copy, which cause segmentation fault on an attempt to free this buffer. The workaround is to add a dummy use of the original pointer after working with the buffer.

Another important skill is an ability to find necessary information on the subjects of the competition, i.e. the ability to work with digital libraries. Usually the task of the competition (or one of the tasks) is an unsolved scientific problem. Thus one may find useful information about methods have been tried or perspective approaches. These gave us several hints for the given task.

2.4 Tools

Aside from the usual requirements this year contest posed an additional one: knowledge of some scripting language or a tool for data analysis. This language can be used for data mining: to detect hidden dependencies in the source data and to collect necessary statistics. We used Python programming language; other examples include R and Octave tools.

2.5 Data

The schema for the data used in the task formulation is presented on Figure 3. Data were stored as a set of CSV files. It is worthy to mention that not all of the files were needed for the query processing. Also, organizers had provided data only for two datasets — the one containing thousand and the one containing ten thousand of persons. These datasets are sufficient for the debug purposes, but they are not enough to tune algorithms for the final evaluation, which involved a graph of million of persons. The benchmark generation parameters were kept in secret and it was impossible to generate that graph by ourselves.

3 Problems

The contest offered [1] the following problems (we fully provide them here for the better understanding of the reader and in case of the original website outage):

1. **Query Type 1 (Shortest Distance Over Frequent Communication Paths).**

   Given two integer person ids $p1$ and $p2$, and another integer x, find the minimum number of hops between $p1$ and $p2$ in the graph induced by persons who:

   (a) have made more than x comments in reply to each other's comments (see comment_hasCreator_person and comment_replyOf_comment);

   (b) know each other (see person_knows_person, which presents undirected friendships between persons; a friendship relationship between persons x and y is represented by pairs $x|y$ and $y|x$).

2. **Query Type 2 (Interests with Large Communities).**

   Given an integer k and a birthday d, find the k interest tags with the largest range, where the range of an interest tag is defined as the size of the largest connected component in the graph induced by persons who:

   (a) have that interest (see tag, person_hasInterest_tag);

   (b) were born on d or later;

   (c) know each other (see person_knows_person, which presents undirected friendships between persons; a friendship relationship between persons x and y is represented by pairs $x|y$ and $y|x$).

3. **Query Type 3 (Socialization Suggestion).** Given an integer k, an integer maximum hop count h, and a string place name p, find the top-k similar pairs of persons based on the number of common interest tags (see person_hasInterest_tag). For each of the k pairs mentioned above, the two persons must be located in p (see person_isLocatedIn_place, place, and place_isPartOf_place) or study or work at organizations in p (see person_studyAt_organization, person_workAt_organization, organisation_isLocatedIn_place, place, and place_isPartOf_place). Furthermore, these two persons must be no more than h hops away from each other in the graph induced by persons and person_knows_person.

4. **Query Type 4 (Most Central People).** Given an integer k and a string tag name t, find the k persons who have the highest closeness centrality values in the graph induced by persons who:

   (a) are members of forums that have tag name t (see tag, forum_hasTag_tag, and forum_hasMember_person);

   (b) know each other (see person_knows_person, which presents undirected friendships between persons; a friendship relationship between persons x and y is represented by pairs $x|y$ and $y|x$).

   Here, the closeness centrality of a person p is:

   $$\frac{(r(p)-1)\cdot(r(p)-1)}{(n-1)\cdot s(p)},$$

   where $r(p)$ is the number of vertices reachable from p (inclusive), $s(p)$ is the sum of geodesic distances to all other reachable persons from p, and n is the number of vertices in the induced graph. When either multiplicand of the divisor is 0, the centrality is 0.

4 Algorithms

In this section we describe algorithms for the tasks of the contest. Due to the space constraints they are presented in a brief, a detailed version featuring algorithm listings can be found in the report [5].

In the rest of this paper we refer to the graph induced by "know each other" relation as $graph$, and to the breadth-first search of that graph as BFS. This graph is used in every query type and BFS (as we show further) plays the key role in all of them. Thus, a shorthand notation would be useful.

4.1 Query Type 1 (Shortest Distance Over Frequent Communication Paths)

4.1.1 Algorithm description

An obvious strategy for evaluation of such query would be the following:

1. Run BFS from person $p1$ to person $p2$ and return hops count;
2. During the BFS traversal one needs to check the replies condition. For each edge, considered on a given BFS step, one has to calculate the number of mutual replies for the corresponding persons. If it is less than k, then the transition is not possible — the edge does not exist.

This "naive" approach needs no preparation and can be ran just after the $graph$ construction. For each pair of adjacent persons it is necessary to calculate the number of replies and this may take some time. Thus, the described BFS has the complexity $O(m \cdot n \cdot (|V| + |E|))$ where n denotes a cardinality of "comment is reply of comment" relation and m — cardinality of "comment has creator person".

Therefore, we propose a pretreatment phase that will compute number of replies once, which effectively eliminates the repeated calculations. Our goal is to find persons that made *not less than* k comments replying to each other. For each pair of persons connected by an edge e in the $graph$ we will determine the number of mutual replies k_e and attribute it to e. In this way, BFS on each step compares two numbers: given k and pre-calculated k_e.

4.2 Query Type 2 (Interests with Large Communities)

In order to reduce the overhead related to connected component size estimation one needs to take into account restrictions which are specified by the query. To tackle the first restriction (the common tag requirement) we built a "tag-person" index. It allows to search persons which are interested in a given tag. We employ the resulting list during the node traversal. It allows us to avoid visiting nodes (persons) which are not interested in a given tag. Also we avoid expenses related to probing person interest list for a given tag.

The second restriction which we have to take into account — the birthdate restriction. This restriction can be tackled by projecting our graph to a given time interval. By doing so, we avoid excessive comparisons related to birthdate which take place during the query processing. In this case the comparisons are moved to the preprocessing phase, thus providing us no benefit. However, this approach may be beneficial, if used differently. The idea is to produce a decomposition of the whole time interval into disjoint several time slices. During the query processing we can use the projection corresponding to an interval d, specified by the query. These projections are constructed during the preprocessing phase. Thus, we can avoid some excessive comparisons during the query processing phase.

Thereby, the estimation of the connected component size for a single tag is essentially a BFS, performed on a graph whose time slice conforms to the date specified by the query. This algorithm can be easily parallelized. For example, one can divide tag set between threads equally and then construct a final result by joining results for the individual tags.

4.3 Query Type 3 (Socialization Suggestion)

4.3.1 Algorithm description.

The common sense may provide the following idea of the straightforward evaluation:

1. for each vertex v in the $graph$ perform BFS while keeping in mind the given hops count h;
2. upon completion BFS returns the list of reached people rp;
3. for v and each person v_r from rp check information about their work places, study places and location for correlation with p;
4. if one of the places where both v and v_r are involved is p or its subplace, then calculate the number of common interests ci;
5. store (sorted by ci) the resulting pairs (v, v_r);
6. return the top-k pairs as a result.

This algorithm requires examination of all the persons returned by BFS. Since $graph$ is a social its edge count follows power law, therefore there are some hubs and connectors with large degree and many vertices with only a few incident edges [11]. Hubs and connectors shorten the paths between persons and thus, the size of rp may be significant. The time complexity of this algorithm is

$$O(|V| \cdot (|V| + |E| + |rp| \cdot |person.places| \\ + |person.interests|)).$$

It is desirable to reduce the number of persons to examine without the loss of result correctness. In order to do that we suggest to group persons by some of place types. SNB provides three place types: $city$, $country$ and $continent$. The type $country$ seems to be a good choice (see [5] for the explanation).

Using the proposed partitioning we suggest a following improvement: use the type of p to determine which country c to process and then perform BFS for each person v from c bearing in mind the given hops count h. That way only persons from c are stored in rp, which reduces its size and allows us to reach our goal.

Described approach time complexity is

$O(|persons\ in\ p| \cdot (|V| + |E| + |rp| \cdot |person.interests|))$.

4.4 Query Type 4 (Most Central People)

4.4.1 The calculation of closeness centrality metric

First of all, we should note, that our graph is an undirected graph, therefore **r(p)** can be calculated once for each connected component. Thus, the problem is how to compute **s(p)**.

An algorithm selection. Given the fact that our graphs is an undirected one and the edges are of unit weights, a simple BFS modification would suffice for the evaluation of **s(p)**. For this purpose we can label each visited vertex with the distance to the initial one. In this approach we do not increase asymptotic complexity of BFS and do not use additional memory. We would require $O(|V| + |E|)$ time and $O(|V| + |E|)$ memory. This estimation is better than estimation for many classical algorithms oriented for general cases of problem "minimal distance from one vertex to all other". For example, Dijkstra algorithm [7] for graphs with non-negative weights, based on Fibonacci heap [8] uses $O(|V| + |E|)$ memory and $O(|V| \cdot \log |V| + |E|)$ time. Moreover, our approach is easily parallelizable: we can compute **s(p)** in parallel for different vertices.

The cut-off heuristic. One can note that *closeness centrality* is inversely proportional to **s(p)** within a connected component. Thus, we can propose a criterion for a vertex to enter the *top-k* of a given connected component which uses it's **s(p)**. Let's define a threshold:

$$\Theta = \max_{p \in current\_top\_k} s(p) .$$

Now, we can interrupt the computation of **s(p)**, if the current value had exceeded the threshold Θ.

Despite the simplicity of this cut-off heuristics it drastically decreased the evaluation time for the fourth query type. Unfortunately, we do not know the number and parameters of queries of this type during the final evaluation. But the implementation of this heuristic allowed to decrease the evaluation time for more than 380 seconds on a graph containing 100 thousand persons. The resulting time was 220 seconds.

We also construct a special index structure for this type of query. More details can be found in the report [5].

Other approaches. In the last few days of the contest we found the solution that fits almost perfectly into the described problem [15]. It is developed for directed graphs with non-negative weights and reuses the CCV of a single vertex in order to estimate CCV for other vertices and reduce the further computations. Authors also use estimates in order to produce the cut-off of vertices which not to get into *top-k*. That method could be modified to take into account the memory restrictions. The experiments described by authors show that this approach may be particularly efficient for unweighted, undirected graph of a large size. It can reduce the amount of computations for a majority of vertices or even avoid their processing at all.

5 System architecture

Graph structure. Considering the graph structure we bear in mind the following: **(i)** the cardinality of vertices may run up to a million, **(ii)** BFS is crucial for the evaluation of every query type. Therefore, our approach must have low memory footprint and provide efficient BFS evaluation. In order to satisfy these requirements we use representation similar to adjacency lists, but with arrays instead, that is, each vertex contains a pointer to an array of adjacent vertices. It allows us to meet the memory constrains and avoid unnecessary comparisons in the BFS implementation.

Layers. Three layers may be distinguished in our implementation: **(i)** file loading, **(ii)** structure initialization and preparation, **(iii)** query evaluation.

This layered structure is rather natural to the task and allows some flexibility in the setting up the order of query evaluation. That is a rather important feature for the performance improvement. The use of the first layer is to provide the interface to chunk-based file loading. It copes with the problem of big files which can be up several gigabytes in size. The use of the second layer is to parse loaded files and to build indexes and other structures required for the query evaluation. The last layer is responsible for the final results formation.

6 Experiments

In this paper we present some experiments illustrating the performance of our approach. Unfortunately, we could not provide detailed experimental data from the contest due to several reasons: (i) we do not have access to the final benchmarks (they are not yet released to public); (ii) we no more have access to the hardware used for the evaluation by the organizers (it was a server-class one); (iii) the two largest benchmarking query sets are unavailable too (we used the largest available dataset — the medium dataset, containing 10k persons).

Thus, we had to perform experiments on our own. The hardware and software setup was the following: i7-4930K CPU (6 cores), P9 X79WS motherboard, 4GB RAM; Ubuntu 14.04, kernel 3.13.0-24, x86_64.

The first series of experiments is presented on Figure 1. They illustrate the basic approach when we sequentially evaluate queries of the same type. The results show the contribution of each query type to the overall processing time. In this series we vary the number of threads. Eventually we get a *U-shaped* graph, which shows that it's not useful to employ more than four threads for the processing in this scenario. It is the result of the algorithm parallelization imperfection (not all algorithms use all cores all the time) and of the synchronization overheads. This leads us to the idea of pre-treatment phase which will allow us to balance the load. The load balancing will be done by grouping tasks together into stages and reordering of query types.

To examine our idea, we had split the query evaluation into the following stages (the stages are described in the [5]): **(i)** Q3 evaluation and Q1 preparation part 1, **(ii)** Q1 preparation part 2, Q2 preparation and Q4 preparation, **(iii)** Q1 evaluation, **(iv)** Q2 evaluation, **(v)** Q4 evaluation. Tasks belonging to one stage are executed in parallel. Figure 2 shows the results for this kind of processing. Despite that in fact we used our idea in the first two stages only, the performance boost of the evaluation with six threads is about 28% (compared to the best performance from Figure 1) and 56% comparing the performance with the six threads. This may be considered a good result for the medium dataset, which we use for testing. Efficiency of such task grouping is determined by the "closeness" of the tasks executed in parallel in terms of time. The closer times of execution, the more efficiently we use the processor. We can perform the load balancing in two ways: by varying the number of threads for one task and by varying the number of tasks. Hence we can use this approach to tune performance further. However, effects of the load balancing may vary with the dataset. Taking such variation into account is rather difficult and requires a more detailed study of the data structures and the algorithms involved.

7 Conclusions

In this paper we described the ACM SIGMOD Contest 2014, its tasks, timeline and our experiences. Also we presented our approach to the offered problems and described the advantages over the naive processing. We discussed algorithms as well as parallelization techniques and presented the general system architecture. Its key points are the following: query type intermixing, query type reordering, continuous query processing and block file loading techniques.

References

[1] ACM SIGMOD 2014 Programming Contest website. http://www.cs.albany.edu/~sigmod14 contest. Accessed 23/05/14.

[2] Apache Giraph website. https://giraph.apache.org/. Accessed 23/05/2014.

[3] Azza Abouzeid, Kamil Bajda-Pawlikowski, Daniel Abadi, Avi Silberschatz, and Alexander Rasin. 2009. HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads. Proc. VLDB Endow. 2, 1 (August 2009), 922–933.

[4] Ronnie Chaiken, Bob Jenkins, Per-Åke Larson, Bill Ramsey, Darren Shakib, Simon Weaver, and Jingren Zhou. 2008. SCOPE: easy and efficient parallel processing of massive data sets. Proc. VLDB Endow. 1, 2 (August 2008), 1265–1276.

[5] On Several Social Network Analysis Problems: a Report. George Chernishev, Vsevolod Sevostyanov, Kirill Smirnov, Ilya Shkuratov. Technical report. http://www.math.spbu.ru/user/chernishev /papers/sigmod2014contest-report.pdf

[6] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. Commun. ACM 51, 1, 107–113.

[7] E. Dijkstra. 1959. A Note on Two Problems in Connexion with Graphs", Numerische mathematik, vol. 1, no. 1, 269–271.

[8] M. L. Fredman and R. E. Tarjan. 1984. Fibonacci Heaps And Their Uses In Improved Network Optimization Algorithms. In Proceedings of the 25th Annual Symposium on Foundations of Computer Science, 1984 (SFCS '84). IEEE Computer Society, Washington, DC, USA, 338–346.

[9] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. Cetin, and S. Babu. Starfish: A Self-tuning System for Big Data Analytics. In Proc. of 5th Conf. on Innovative Data Systems Research (CIDR), 2011.

[10] InfiniteGraph: The Distributed Graph Database. Whitepaper. http://www.objectivity.com/wp-content/uploads/Objectivity_WP_IG_Distr_ Benchmark.pdf. Accessed 23/05/2014.

[11] LDBC SocialNet Benchmark: Data Generation. https://github.com/ldbc/ldbc_socialnet_bm /wiki/Data-Generation#graph-generation. Accessed 23/05/2014.

[12] The Neo Database — A Technology Introduction (20061123). http://dist.neo4j.org/neo-tech nology-introduction.pdf. Accessed 23/05/2014.

[13] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. 2008. Pig latin: a not-so-foreign language for data processing. In Proceedings of the 2008 ACM SIGMOD international conference on Management of data (SIGMOD '08). ACM, New York, NY, USA, 1099–1110.

[14] Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, and Theo Vassilakis. 2010. Dremel: interactive analysis of web-scale datasets. Proc. VLDB Endow. 3, 1–2 (September 2010), 330–339.

[15] Paul W. Olsen, Alan G. Labouseur, Jeong-Hyon Hwang. "Efficient Top-k Closeness Centrality

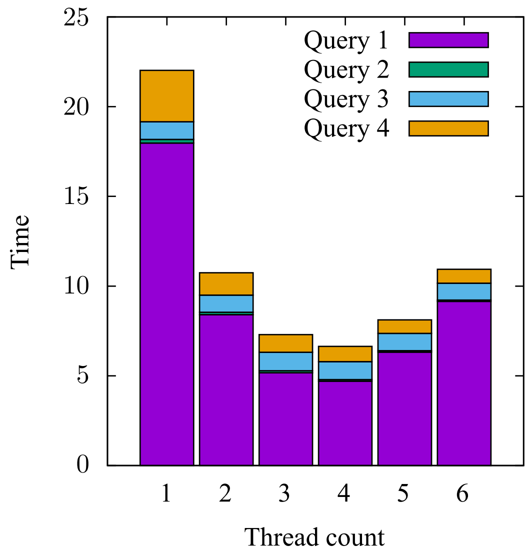

Figure 1: Performance scalability (without pre-treatment phase).

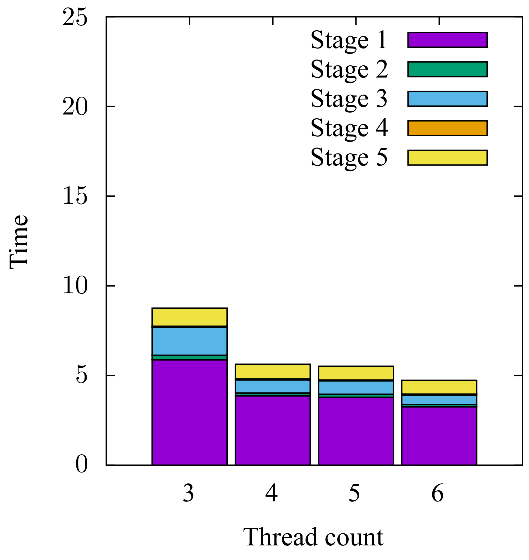

Figure 2: Performance scalability and effects of query reordering (pre-treatment phase).

Search". In Proceedings of the Data Engineering (ICDE), 2014 IEEE 30th International Conference, p 197-207, Chicago, IL, USA, 2014.

[16] Social Network Benchmark (SNB) Task Force Progress Report http://www.ldbc.eu:8090/download/attachments /4325436 /LDBC_SNB_Report_Nov2013.pdf. Accessed 23/05/14.

[17] Signal/Collect Documentation (website). http://uzh.github.io/signal-collect/documenta tion.html. Accessed 23/05/14.

[18] ACM SIGMOD Programming Contest: an opportunity to study distinguished aspects of database systems and software engineering. Kirill K. Smirnov, Georgiy A. Chernishev. 2012. Компьютерные инструменты в образовании, 6(2012), 22–25, ISSN: 2071-2340, url:http://ipo.spb.ru/journal/index.php?article /1541/ (in Russian).

[19] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. 2009. Hive: a warehousing solution over a map-reduce framework. Proc. VLDB Endow. 2, 2 (August 2009), 1626–1629.

8 Appendix: SNB Description

Let's briefly survey the SNB benchmark which was used during the contest and in the experimental section of this paper.

The purpose. In order to provide efficient evaluation for a variety of algorithms, tools, frameworks for social network data management tasks, a standard benchmark, called Social Network Benchmark (SNB) [16] was developed. This benchmark allows not only efficient, but also a repeatable evaluation for a variety of scenarios: online transactions, business intelligence and graph analytics. Authors of the benchmark tried to make it as realistic as possible.

Covered systems. This benchmark covers several types of systems: graph DBMS and graph programming frameworks, RDF database systems, relational and NoSQL database systems.

Data schema. The general data schema of the benchmark is presented on Figure 3 (illustration taken from [16]). It is called Social Intelligence Benchmark Data Schema. The schema uses UML notation to describe entities, attributes and their relationships of different cardinalities. The schema defines the result of the benchmark's data generator. Essentially it is a set of tables linked via primary-foreign key relationships.

The schema defines some social network and its most characteristic features:

1. users and their personal details, tags and likes;
2. relations between users (follows and knows);
3. textual content: posts and comment trees.

Generator and its output: technical details. This benchmark is essentially a synthetic data generator, which is implemented using MapReduce programming model. The generator is dictionary-based and is capable of generating correlated values. The result of the generator is the set CSV files, where each file contains records of the corresponding table.

The benchmark and the contest. The organizers of the contest used only the dataset generator, but not

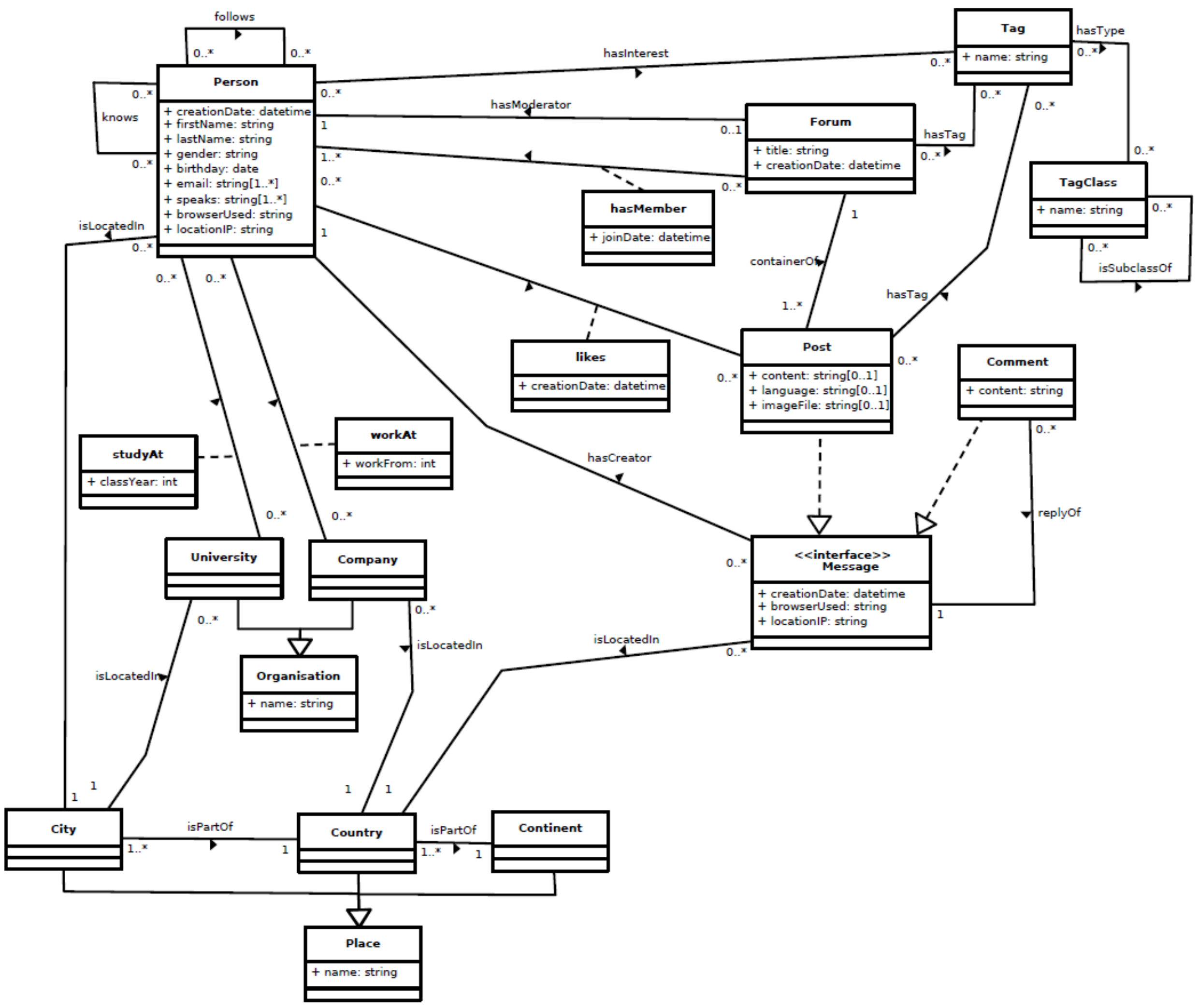

Figure 3: Social Intelligence Benchmark Data Schema

queries. Instead, they proposed four stand-alone types of queries, which we described earlier.

The dataset generator provided four types of graph workloads: small (1k vertices), medium (10k vertices), large (100k vertices) and huge (1M vertices). The last one would be used for the final evaluation by the contest organizers.

Unfortunately, only the first two datasets were fully released to the public. The third one was discussed in the mailing list, where some of the generator parameters for this dataset were disclosed. However, no queries are known. In this paper we use the largest available (on the current date) dataset — the medium one for the experimental evaluation. All of the queries are known at the start of the processing, contestants are not required to process them in a specific order.