

Systems and Algorithms for Large-scale Graph Analytics

Edited by

Eiko Yoneki¹, Amitabha Roy², and Derek Murray³

¹ University of Cambridge, GB, eiko.yoneki@cl.cam.ac.uk

² EPFL – Lausanne, CH, amitabha.roy@epfl.ch

³ San Francisco, CA, US

Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 14462 “Systems and Algorithms for Large-scale Graph Analytics”. The seminar was a successful gathering of computer scientists from the domains of systems, algorithms, architecture and databases all of whom are interested in graph processing.

Seminar November 9–12, 2014 – <http://www.dagstuhl.de/14462>

1998 ACM Subject Classification B.3.3 Performance Analysis and Design Aids, B.5.1 Design, C.1.2 Multiple Data Stream Architectures (Multiprocessors), D.1.3 Concurrent Programming


Keywords and phrases Large-scale graph processing, Graph structured data, Database, Graph algorithms, Parallel I/O, Parallel programming, Storage, Distributed systems, GPU, Solid state drive (SSD)

Digital Object Identifier 10.4230/DagRep.4.11.59

1 Executive Summary

Amitabha Roy

Eiko Yoneki

License  Creative Commons BY 3.0 Unported license
© Amitabha Roy and Eiko Yoneki

Graph analytics, the class of data analysis that deals with data forming networks, is emerging as a huge consumer of computational resources due to its complex, data-hungry algorithms. Social networking, personal medicine, bioinformatics, text/graphics content analysis and search engines are a few examples where Tera-, Peta- or even Exa-scale graph processing is required. Graph algorithms are becoming increasingly important for solving multiple problems in diverse fields. As these problems grow in scale, parallel computing resources are required to meet the computational and memory requirements. Notably, the algorithms, software and hardware that have worked well for developing mainstream parallel applications are not usually effective for massive-scale graphs from the real world, which exhibits more complex and irregular structures than traditional data in scientific computing.

Research into large scale graph processing within the computer science community is currently at an early and fragmented stage. This seminar brought together researchers from systems, computer architecture, algorithms and databases to discuss emerging trends and to identify opportunities for future advancement. Prior to the seminar, we had prepared a range of research questions below.

1. What is the correct algorithmic abstraction for systems handling large graphs? Algorithmic complexity researchers use PRAM and I/O complexity models to characterize the algorithmic complexity of graph processing. On the other hand, systems researchers



Except where otherwise noted, content of this report is licensed under a Creative Commons BY 3.0 Unported license

Systems and Algorithms for Large-scale Graph Analytics, *Dagstuhl Reports*, Vol. 4, Issue 11, pp. 59–77

Editors: Eiko Yoneki, Amitabha Roy, and Derek Murray



Dagstuhl Reports

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- largely build systems that implement scatter-gather and label propagation models of computation. These are different world views rendering theory and practice incompatible with each other. We will begin work towards a formal algorithmic model for existing large scale graph processing systems as part of this seminar with a view to answering this question. This model should accurately describe large-scale graph processing systems built by the systems community as well as be formal enough to enable algorithmic complexity researchers to draw useful conclusions about their scalability with data set size. This will require close co-operation between theoretical computer scientists on the algorithms front to talk to practical systems researchers.
2. What is the taxonomy of applications that graph processing systems should support? Can it be reduced to a set of representative benchmarks that researchers in this area need to care about? We can currently identify two main interest areas. The first is large scale graph traversal, of interest to the high performance computing and web-services community; primarily driven by security applications and from data mining needs. The second is spectral approaches, primarily of interest to the machine learning community, building systems such as Graphlab. The output from this agenda item will be a clear set of well defined applications that the community can agree will serve as objects of study for building high performance graph processing systems.
 3. What is the correct interface to the system that may be assumed when building a graph DSL? DSL researchers are interested in productive and easy ways to specify graph computations. However they have given relatively little thought to interfacing in an efficient way to systems that execute graph computation. The agenda item therefore will be a discussion between programming language researchers and systems, algorithms and database people researchers about the correct level of interface between a DSL and the underlying systems. A good model in this regard is the decoupling of database systems from ways to query them using declarative languages like SQL. The litmus test for success for this agenda item will therefore be a sketch for a DSL that exposes opportunities for optimization, is productive to use and at the same time is oblivious to the underlying system.
 4. What is the design trade-off among different graph processing approaches. For example, (i) the general graph processing system vs. the dedicated approaches specially optimized for specific graph problems, (ii) the running time between pre-processing and graph processing, (iii) performance vs. running expense.

The seminar identified whole graph analytics and point queries on graphs that explore neighborhoods of vertices as distinct application domains, which require separate treatment and systems. All the participants agreed that there was an urgent need to standardize benchmarks and datasets in order to make meaningful progress with graph processing – particularly given the diverse nature of the communities involved. In addition, the seminar identified a number of interesting approaches and trends. There was also considerable participation from industry, which included work in graph databases as well as new systems architectures that will require practitioners to rethink traditional approaches for graph processing.

The seminar consisted of 6 sessions on focused topic presentations and discussions, followed by a joint session with the seminar 14461 on “High-performance Graph Algorithms and applications computational Science”. At the last day of the seminar, the whole morning was dedicated to the discussion on the challenges and future directions of large-scale graph processing (see Section Challenges and Future directions).

2 Table of Contents

Executive Summary

<i>Amitabha Roy and Eiko Yoneki</i>	59
---	----

Overview of Talks

Grappa: A Runtime System for High-Performance Graph Analyses on Commodity Clusters <i>Mark Oskin</i>	63
Issues Encountered in Distributed Graph Processing <i>Sebastian Schelter</i>	63
Graph Analytics with the Galois System <i>Andrew Lenharth</i>	64
An Evaluation of Pregel-like Graph Processing Systems <i>Khuzaima Daudjee</i>	64
Rethinking Graph Processing with the Machine <i>Kimberley Keeton</i>	64
Hyperball: In-core Computation of Geometric Centralities <i>Paolo Boldi</i>	65
Naiad <i>Derek Murray</i>	65
Traversal of Massive Scale-free Graphs on HPC with NVRAM <i>Roger Pearce</i>	66
Graph Processing from Secondary Storage <i>Willy Zwaenepoel</i>	66
SSD Prefetching for Large Graph Traversal <i>Valentin Dalibard</i>	66
In-network processing and NVRAM in Grappa <i>Luis Ceze</i>	67
Gunrock: High-Performance, High-Level Graph Computation on GPUs <i>John Owens</i>	67
Using Integrated GPUs for Accelerated Graph Traversal <i>Karthik Nilakant</i>	68
Graph Languages with Benefits <i>Sungpack Hong</i>	68
Query Languages that Taste like Programming Languages <i>Torsten Grust</i>	69
Flattening-Based Query Compilation <i>Alexander Ulrich</i>	69
Core Graph Processing Primitives within SAP HANA <i>Wolfgang Lehner</i>	69
Graph Database, do we need to reinvent the wheel <i>Sungpack Hong and Hasan Chafi</i>	70

Advancing Graph Query Languages <i>Stefan Plantikow</i>	70
Efficient Large Scale Data Analytics <i>Lei Chen</i>	71
Towards Querying Large Graphs <i>Arijit Khan</i>	71
Distributed Algorithms for Graph Partitioning and Community Detection <i>Sarunas Girdzijauskas</i>	72
Taming Graph Dynamics at Scale <i>Felix Cuadrado</i>	72
An Initial Attempt to Classify Graph Processing Systems and Approaches <i>Tamer Özsu</i>	73
Joint Session	
Challenges for Popularizing Graph Applications <i>Hasan Chafi</i>	73
Graph Analysis Platforms and Evaluations Thereof <i>Terence Kelly</i>	74
Graphbench.org <i>Luis Ceze</i>	74
Challenges, Future Directions	
<i>Eiko Yoneki and Amitabha Roy</i>	74
Participants	77

3 Overview of Talks

Talks are distributed in the following six sessions. The detail of the seminar schedule can be found at <http://www.dagstuhl.de/schedules/14462.pdf>.

1. Graph/Parallel Processing Systems (3.1–3.7)
2. Storage, I/O, Memory, and Data Format (3.8–3.10)
3. Hardware/Software based Acceleration (GPU, APU) (3.11–3.13)
4. Query/Programming Languages and Applications (3.14–3.17)
5. Distributed and Graph Databases (3.18–3.21)
6. Parallel Processing and Partitioning (3.22–3.24)

3.1 Grappa: A Runtime System for High-Performance Graph Analyses on Commodity Clusters

Mark Oskin (University of Washington, US, oskin@cs.washington.edu)

Note by Editors: Grappa is a runtime system that provides a distributed shared memory abstraction for a cluster. At Grappa’s core is co-operative multithreading with very low context switch times (in the order of 40–80 ns) and this allows Grappa to provide fine grained access to memory on a different machine while hiding the latency of access behind parallelism. Grappa also uses delegation by moving computation to the data (instead of vice-versa) and aggregates messages to the same machine to amortize per-message overheads on the network. It provides an easy programming model that often looks very much like HPC code. These features make Grappa a great solution for graph processing – it dispenses with the need for good partitioning across machines and manages to beat Powergraph for all possible partitioning strategies that Powergraph can currently use.

3.2 Issues Encountered in Distributed Graph Processing

Sebastian Schelter (TU Berlin, DE, sebastian.schelter@tu-berlin.de)

Note by Editors: This talk focuses on the challenges faced when using distributed graph processing systems in industrial settings. Although the “think like a vertex” programming model has become popular, pre and post-processing steps usually involve relational operators instead. An interesting observation made is that the vertex centric model can be viewed as a join followed by an aggregation, which is the approach used in systems such as GraphX. Further, more complex operations that can be represented as linear algebra operations are not well suited to the vertex-centric model. An example is linerank: a scalable alternative to betweenness centrality to computes the stationary distribution for a random walk on the line graph. Another example is the factorbird work done at Twitter that involved factorizing very large matrices: a parameter server approach was found to work the best. These relational operators or linear algebra operators did not fit in the vertex centric model. LineRank for example can not be computed on top of BSP (Bulk Synchronous Parallel).

3.3 Graph Analytics with the Galois System

Andrew Lenharth (University of Texas Austin, US, lenharth@ices.utexas.edu)

Note by Editors: Galois makes single machine concurrent graph algorithms easier to write by hiding details of synchronization entirely from the user. As input Galois takes a serial program which consists of an algorithm and data structure. It transforms it into a parallel program consisting of operators, a schedule and a parallel data structure. Priority scheduling is at the heart of Galois and enables great performance on algorithms like SSSP or pagerank, something not provided by vertex-centric systems. The ability to use the correct data structure also helps in the case of graphs with large diameter, connected components being an example where standard label propagation algorithms do not work well. Galois provides an order of magnitude improvement over systems such as Powergraph. Priority scheduling allows the use of information from the application layer for the scheduling and processing vertices with a high importance makes the computation faster.

3.4 An Evaluation of Pregel-like Graph Processing Systems

Khuzaima Daudjee (University of Waterloo, CA, kdaudjee@uwaterloo.ca)

Note by Editors: Graph partitioning is an expensive problem and therefore it makes sense to do incremental re-partitioning. We use 1-hop and 2-hop neighbor queries as the workload to optimize for, when re-partitioning, assuming that remote traversals are expensive. The work led to a distributed Neo4j implementation. Another direction of research is to compare and contrast different Pregel-like graph processing systems [1]. Each system features its own optimizations that have been assessed in the evaluation.

References

- 1 An Experimental Comparison of Pregel-like Graph Processing Systems. VLDB 2014.


3.5 Rethinking Graph Processing with the Machine

Kimberley Keeton (HP Labs – Palo Alto, US, kimberley.keeton@hp.com)

Note by Editors: The “machine” is a radical new systems architecture currently in development that leverages memristor technology and silicon photonics. The motivator for the machine is that DRAM scaling is hitting a wall, and it is unlikely we can store large amount of data in it. At the same time secondary storage speeds are unlikely to keep up with the needs of analytics applications that need sufficiently low latency access to data. The machine provides vast amounts of persistent low latency storage via memristor technologies with performance much closer to DRAM than NAND flash. Photonics technology is intended to serve as a communication firehose for memristor stores. These provide low cost, compact form factors via vertical cavity emitting lasers and reach speeds on the the order of a terabit a second. The machine will change the way we approach problems such as graph processing. Partitioning graphs across distributed memory in a cluster – a known hard problem – simply disappears since a single machine would have enough capacity to hold the entire graph. Another interesting application is maintaining multiple indexes on the same graph.

3.6 Hyperball: In-core Computation of Geometric Centralities

Paolo Boldi (University of Milan, IT, boldi@di.unimi.it)

License  Creative Commons BY 3.0 Unported license
© Paolo Boldi

Given a social network, which of its nodes are more central? This question has been asked many times in sociology, psychology and computer science, and a whole plethora of centrality measures (a.k.a. centrality indices, or rankings) were proposed to account for the importance of the nodes of a network. In this paper, we approach the problem of computing geometric centralities, such as closeness and harmonic centrality, on very large graphs; traditionally this task requires an all-pairs shortest-path computation in the exact case, or a number of breadth-first traversals for approximated computations, but these techniques yield very weak statistical guarantees on highly disconnected graphs. We rather assume that the graph is accessed in a semi-streaming fashion, that is, that adjacency lists are scanned almost sequentially, and that a very small amount of memory (in the order of a dozen bytes) per node is available in core memory. We leverage the newly discovered algorithms based on HyperLogLog counters, making it possible to approximate a number of geometric centralities at a very high speed and with high accuracy. While the application of similar algorithms for the approximation of closeness was attempted in the MapReduce framework, our exploitation of HyperLogLog counters reduces exponentially the memory footprint, paving the way for in-core processing of networks with a hundred billion nodes using just 2TiB of RAM. Moreover, the computations we describe are inherently parallelizable, and scale linearly with the number of available cores.

Note by Editors: Hyperball is a new approach to computing the centrality of vertices in large graphs [1]. Normally, this is a computationally intensive approach that is prohibitively expensive to compute for every vertex in a large graph. Hyperball proposes using the notion of geometric centrality of a vertex x in a graph $G = (V, E)$: $\sum_{y \in V} \frac{1}{\text{dist}(y, x)}$. Hyperball computes geometric centralities efficiently using HyperLogLog counters and sequential passes over the graph in a manner very similar to HyperANF. Although Hyperball does not admit a tight closed form bound on the error, in practise it returns centrality measures with very low error, while being efficient enough to allow centrality measures to be estimated on graphs with a hundred billion vertices and beyond.

References

- 1 <http://arxiv.org/abs/1308.2144>

3.7 Naiad

Derek Murray (Google, derek.murray@gmail.com)

Note by Editors: Naiad is a distributed system for executing data-parallel cyclic dataflow programs. One of its most interesting applications is to combine incremental processing with interactive queries. For example, one might want to build connected components of Twitter users from an incoming stream of messages and then obtain the most popular hashtag in each component. Such applications are easy to write on Naiad, which in turn provides great performance without the user having to worry about the underlying details. At its core Naiad attaches an ordering timestamp to messages flowing through the system. Prioritizing

the processing of messages is key to obtaining good performance from the system. Another key idea is to do any necessary joins incrementally. Naiad also applies batching in order to better apply the priority based selection of messages to process. Naiad's timely dataflow API provides a good basis to implement other systems – including graph processing systems.

3.8 Traversal of Massive Scale-free Graphs on HPC with NVRAM

Roger Pearce (Lawrence Livermore National Labs, US, rpearce@llnl.gov)

Note by Editors: Traversing massive graphs with HPC is an important application area as it facilitates processing of massive real-world graphs. The key problem is to tolerate data latencies and the adopted solution is to use parallel asynchronous techniques. An important building block is the visitor abstraction where a visit to a vertex results in traversal of graph edges, which in turn queues further visits to target vertices. Visitor execution scheduler orders visitors to exploit page-level reuse. Another important aspect of the work is partitioning vertices across cluster nodes to queue visitors at. The solution was to split high-degree hub nodes and delegate communication to the copies. The solution also incorporates a message aggregation layer (motivated in much the same way as Grappa) although message aggregation happens both at source and at intermediate software routers. It also motivates a future with NVRAM as MTEPS decreases only by about 10% when moving from DRAM to NVRAM.

3.9 Graph Processing from Secondary Storage

Willy Zwaenepoel (EPFL, CH, willy.zwaenepoel@epfl.ch)

Note by Editors: Graph processing from secondary storage takes an entirely different approach from the traditional one of storing the graph in the main memory of a single machine or many different machines. The aim is to exploit large amounts of secondary storage (including magnetic disks) to process very large graphs using a fraction of the resources normally required. To this end, the talk covers two systems. X-Stream is a system for processing graphs from secondary storage on a single machine. It recasts graph computation to use edge-centric computing and partitions graphs into streaming partitions. This results in purely sequential access to secondary storage. The second system, SlipStream, extends the ideas in X-Stream to work with distributed storage in a cluster. It does so using a combination of flat storage to avoid graph partitioning problems, and work stealing to avoid load imbalance problems.

3.10 SSD Prefetching for Large Graph Traversal

Valentin Dalibard (University of Cambridge, GB, valentin.dalibard@cl.cam.ac.uk)

Note by Editors: Mining large graphs has now become an important aspect of multiple diverse applications and a number of computer systems have been proposed to provide runtime support. Recent interest in this area has led to the construction of single machine graph computation systems that use solid state drives (SSDs) to store the graph. This approach reduces the cost and simplifies the implementation of graph algorithms, making

computations on large graphs available to the average user. However, SSDs are slower than main memory, and making full use of their bandwidth is crucial for executing graph algorithms in a reasonable amount of time. In this paper, we present PrefEdge, a prefetcher for graph algorithms that parallelizes requests to derive maximum throughput from SSDs. PrefEdge combines a judicious distribution of graph state between main memory and SSDs with an innovative read-ahead algorithm to prefetch needed data in parallel. This is in contrast to existing approaches that depend on multi-threading the graph algorithms to saturate available bandwidth. Our experiments on graph algorithms using random access show that PrefEdge not only is capable of maximizing the throughput from SSDs but is also able to almost hide the effect of I/O latency. The improvements in runtime for graph algorithms is up to 14 x when compared to a single threaded baseline. When compared to multi-threaded implementations, PrefEdge performs up to 80% faster without the program complexity and the programmer effort needed for multi-threaded graph algorithms.

3.11 In-network processing and NVRAM in Grappa

Luis Ceze (University of Washington, US, luisceze@cs.washington.edu)

Note by Editors: Systems like Grappa spend a lot of time aggregating messages. However aggregating messages at the sender is limited in view and perhaps a better place to do aggregation is in the network. Moving forward, DRAM capacity per-core is shrinking due to the end of Dennard scaling also affecting DRAM. At the same time, network injection rates are getting better – beginning to rival DRAM. This suggests that it might also be beneficial to do some amount of compute in addition to aggregation in network middle boxes. Another interesting idea is to use Grappa to also tolerate increased latencies to main memory when DRAM is replaced with NVRAM.

3.12 Gunrock: High-Performance, High-Level Graph Computation on GPUs

John Owens (UC Davis, US, jowens@ece.ucdavis.edu)

License © Creative Commons BY 3.0 Unported license
© John Owens

We are building Gunrock, a graph framework for GPUs that delivers high performance on iterative, bulk-synchronous graph primitives with high-level programmability. Gunrock supports efficient traversal, filtering, and computation operators. In my talk I outline challenges facing Gunrock and by extension other GPU graph frameworks: load-balanced, high-performance traversal; scalability beyond the memory footprint of one GPU; addressing higher-level operations beyond graph primitives; and parallel-friendly mutable graph data structures.

Note by Editors: Executing graph algorithms on GPUs requires efficient parallel algorithms from the GPU perspective, coding which traditionally requires bridging a “ninja” programming gap. Gunrock aims to bridge this programming skills gap with the right interface, while enabling good use of GPU resources. It uses a bulk synchronous programming model. Traversals maintain an active frontier of active vertices and edges. Each traversal step does

compute and generates a new frontier. Gunrock uses the CSR representation internally as that works best. It also uses careful scheduling of units of work (corresponding to blocks of edges) to GPU units such as cores, threads and warps in order to achieve the best performance. A future area of interest with GPUs and graphs is to look at graphs that change as a result of computation.

3.13 Using Integrated GPUs for Accelerated Graph Traversal

Karthik Nilakant (University of Cambridge, GB, karthik.nilakant@cl.cam.ac.uk)

License  Creative Commons BY 3.0 Unported license
© Karthik Nilakant

General-purpose GPU computing has been used to improve the performance of a variety of applications. However graph processing is a problem domain that is difficult to adapt to the GPGPU computing model, due to irregularity and poor locality of access. There are aspects of graph processing that can benefit from the additional parallelism provided by the GPU platform, but the need to ship data between system RAM and discrete graphics memory negates the potential performance improvement. Integrated graphics processors (or accelerated processing units) provide a way to avoid this problem, by adopting a unified memory model. In this talk, I show that by adopting a current-generation APU platform.

Note by Editors: Systems where CPUs and GPUs are integrated on the same die are becoming commonplace. This represents an opportunity as both see the same shared memory with the same access latency. A graph algorithm can therefore be run more efficiently by partitioning work across the CPU cores and GPUs. However due to their different characteristics, discrimination is needed when assigning work. The key idea is to assign more homogeneous low-degree vertices to GPUs. This leads to substantial speedups over simply assigning all work to CPUs or GPUs.

3.14 Graph Languages with Benefits

Sungpack Hong (Oracle – Belmont, US, sungpack.hong@oracle.com)

License  Creative Commons BY 3.0 Unported license
© Sungpack Hong


There are two types of graph processing workloads – computational analytic and pattern matching. We first discuss the benefit of having a high-level imperative language for computational graph analytic. We give examples of simplicity, performance and portability benefits. Then we discuss (existing) declarative languages for pattern matching workloads. We close the talk by asking how to combine these two languages.

Note by Editors: There is a benefit to designing programming languages specifically for graphs. This language should be easy to use both for computational graph analytics like pagerank as well as for declarative queries such as graph pattern matching. Even for computational problems it is useful to raise the level of abstraction above that of “think like a vertex”, a user typically does not want to be aware of or bother with the computational model. At the same time, a higher level of abstraction enables system specific optimization to be brought into play. A good example is ordering vertices by degree to reduce the number

of neighbor list comparisons needed in triangle counting. Oracle now has a proprietary graph processing engine (PGX.dist) that incorporates this philosophy by exposing a DSL which is then compiled down through IR all the way to system level details such as ghost vertices.

3.15 Query Languages that Taste like Programming Languages

Torsten Grust (Universitat Tübingen, DE, torsten.grust@uni-tuebingen.de)

License  Creative Commons BY 3.0 Unported license
© Torsten Grust

We study database query languages whose design mimics those of general-purpose programming languages. In these languages, (1) clauses may be composed freely and nest to arbitrary depth (as long as typing rules are obeyed) and (2) data may be modeled using a rich type system (that admits lists, arrays, dictionaries). This is in contrast to the common design of query languages (see SQL or SPARQL) in which users need to fit their queries into rather rigid templates that query flat data structures.

The compilation of such languages and their execution on common database backends poses a challenge. We borrow insights and adapt techniques invented by the programming language community and the 1990s efforts of implementing nested data parallelism, in particular to let query languages taste more like programming languages.

Note by Editors: Combining SQL style query languages with C is a hard problem and often requires bi-lingual developers who communicate using narrow channels from the C heap to the database. The proposed solution is database supported program execution. The program is compiled down to native code and SQL queries that are instantiated automatically on the program heap. This is a direct counterargument to DSLs and suggests that richer language level abstractions are a better way to deal with graphs and other forms of data rather than inventing domain specific languages.

3.16 Flattening-Based Query Compilation

Alexander Ulrich (Universitat Tübingen, DE, alexander.ulrich@uni-tuebingen.de)

Note by Editors: Combining databases and data models such as graphs with programming languages leads to the interesting question of how best to query data in such settings. The proposed solution involves trading iteration for lifted operators and un-nesting comprehensions. A key question is then whether a column backend is better suited for vector operations.

3.17 Core Graph Processing Primitives within SAP HANA

Wolfgang Lehner (TU Dresden, DE, wolfgang.lehner@tu-dresden.de)

License  Creative Commons BY 3.0 Unported license
© Wolfgang Lehner


SAP HANA internally exhibits core primitives for processing graph data stored within the property graph model. In this talk, I will touch the SAP proprietary “Graph Extraction

and Manipulation” language (GEM) and discuss the graph traversal operator as one of the most interesting graph operations. The talk outlines a corresponding plan operator which is embedded into the SAP HANA optimizer and execution runtime and gives insights into two different strategies traversing graphs based on a columnar storage representation.

Note by Editors: SAP HANA use cases that involve graphs are traceability track and trace, supply chain planning and optimization, and medical insight for health care. A motivator was that one cannot take data out of the system so why not embed graph analytics into a main memory centric/column store system? HANA allows one to plug in different engines and languages. The stack takes a language, compiles down through an optimizer and plan generator and passes the result to in-memory processing engines including graph engines. The GEM language is used for graph exploration and manipulation. The language allows loading, inserting and deleting objects with possibly irregular structure. The engine can efficiently traverse links between data objects. It uses a threshold that switches between brute force scans of columns to index lookups.

3.18 Graph Database, do we need to reinvent the wheel

Sungpack Hong and Hasan Chafi (Oracle – Belmont, US)

License  Creative Commons BY 3.0 Unported license
© Sungpack Hong and Hasan Chafi

Graph is a great data modeling tool that enables powerful data analysis. However, we do not believe that we need a completely new database system only for maintaining graph data. Instead, we propose an system where the graph data is mapped into conventional relational database which provides consistent data management. For the sake of graph analysis, the database is tightly integrated with a fast in-memory/distributed analytic engine. We also open up a discussion about creating graphs directly from relational tables.

Note by Editors: Graphs are a relatively new data model that capture arbitrary relationships between entities. They enable newer applications such as collaborative filtering. On the other hand traditional databases provide a great platform for ACID semantics. Combining the two is therefore a good idea. The insight is that one can execute long running analytics on snapshots of the graph in memory. Each snapshot is consistent. This is the new PGX system that is being built at Oracle. It scales out by partitioning the in-memory snapshot and using ghost vertices.

3.19 Advancing Graph Query Languages

Stefan Plantikow (Neo Technology Germany, DE, stefan.plantikow@gmail.com)

Note by Editors: Forrester estimates that over 25% of enterprises will be using graph databases by 2017. Neo4j is a graph database meant for all sources of graph data – cloud, social, mobile and information (wikipedia). Neo4j is an OLTP graph database with use cases in routing, navigation, logistics, network impact analytics, social recommendations, access control, fraud detection and others. It uses a labeled property graph data model with highly connected data. It also employs master-slave replication with query sharding. Cypher is Neo4j’s declarative graph language, addressing the shortcomings of SQL for graph queries.

Cypher handled path sets, functional expressions, and optional matches (outer join). The query model starts from vertices, finds instances of a pattern such that each relationship is uniquely bound and aggregates results.

3.20 Efficient Large Scale Data Analytics

Lei Chen (Hong Kong University of Science and Technology, HK leichen@cse.ust.hk)

Note by Editors: A number of fundamental graph problems such as community detection, centrality computations, frequent subgraph mining needs a scalable and efficient graph analysis toolkit. Most existing toolkits are either designed for HPC or focused on machine learning. An example of a problem not handled well by these toolkits is subgraph detection and listing. Our solution [1] uses a dynamic partitioning algorithm and traversal on an in-memory graph. It uses independence properties on the expansion tree to parallelize subgraph listing.

References

- 1 Parallel subgraph listing in a large-scale graph. SIGMOD 2014.

3.21 Towards Querying Large Graphs

Arijit Khan (ETH, CH, arijit.khan@inf.ethz.ch)


License © Creative Commons BY 3.0 Unported license
© Arijit Khan

Given a massive network of interconnected entities such as the Google knowledge graph, how can a user search for an activity at a child-friendly tourist-attraction site inside the New York City that is also close to an Asian restaurant? Freebase that powers Google's knowledge graph alone has over 22 million entities and 350 million relationships in about 5428 domains. Before users can query anything meaningful over this data, they are often overwhelmed by the daunting task of attempting to even digest and understand it. Without knowing the exact structure of the data and the semantics of the entity labels and their relationships, can we still query them and obtain the relevant results? In this talk, I shall give an overview of our user-friendly and scalable techniques for querying of large-scale networks, including heterogeneous networks, uncertain and stream graphs.

Note by Editors: Graph queries usually involve a complex combination of structure and content. For example, one might query from a mobile device for good restaurants recommended by friends on the way to the bank. Queries also often involve some amount of fuzz and users might want the best answer rather than an exact answer. One way to solve this problem is to make a vector representing the neighborhood of a vertex and propagate this vector along the edges. A string distance metric can then be used to rank matches. The algorithm starts from low-degree nodes first to avoid explosion in dimensions with high-degree nodes. Compared to existing algorithms, this results in shorter runtimes but can lead to false positives.

3.22 Distributed Algorithms for Graph Partitioning and Community Detection

Sarunas Girdzijauskas (Royal Institute of Technology, SE, sarunas@sics.se)

License  Creative Commons BY 3.0 Unported license
© Sarunas Girdzijauskas

Balanced graph partitioning is a well known NP-complete problem with a wide range of applications. These applications include many large-scale distributed problems including the optimal storage of large sets of graph-structured data over several hosts – a key problem in today’s Cloud infrastructure. However, in very large-scale distributed scenarios, state-of-the-art algorithms are not directly applicable, because they typically involve frequent global operations over the entire graph. In this talk, we introduce a fully distributed algorithm, called JA-BE-JA, that uses local search and simulated annealing techniques for graph partitioning. The algorithm is massively parallel: there is no central coordination, each node is processed independently, and only the direct neighbors of the node, and a small subset of random nodes in the graph need to be known locally. Strict synchronization is not required. These features allow JA-BE-JA to be easily adapted to any distributed graph-processing system from data centers to fully distributed networks. We perform a thorough experimental analysis, which shows that the minimal edge-cut value achieved by JA-BE-JA is comparable to state-of-the-art centralized algorithms such as METIS. In particular, on large social networks JA-BE-JA outperforms METIS, which makes JA-BE-JA, a bottom-up, self-organizing algorithm, a highly competitive practical solution for graph partitioning. We also present a highly parallel solution for cross-document coreference resolution, which can deal with billions of documents that exist in the current web. At the core of our solution lies a novel algorithm for community detection in large scale graphs. We operate on graphs which we construct by representing documents’ keywords as nodes and the co-location of those keywords in a document as edges. We then exploit the particular nature of such graphs where coreferent words are topologically clustered and can be efficiently discovered by our community detection algorithm. The accuracy of our technique is considerably higher than that of the state of the art, while the convergence time is by far shorter. In particular, we increase the accuracy for a baseline dataset by more than 15% compared to the best reported result so far.

Note by Editors: JA-BE-JA is a new gossip based algorithm to partition graphs. Each node has access only to its immediate state and that of its neighbors. Nodes communicate only through messages. Each node starts with a random color. Each node attempts to change its color to the most dominant color among its neighbors. An improved version requires nodes to exchange colors for pairwise benefit with simulated annealing. An interesting application of this algorithms was in taking a set of documents, treating the words vertices in a graph and connecting vertices (words) by an edge if they appear in the same document.

3.23 Taming Graph Dynamics at Scale


Felix Cuadrado (Queen Mary University of London, GB, felix.cuadrado@qmul.ac.uk)

Note by Editors: Real world graphs are dynamic. We looked at distributed graph processing systems that offer a Pregel-style interface but support arbitrary changes to the graph. Partition quality is a determinant of graph performance but the quality of the partition can

start to degrade as changes are made to the graph. Therefore we implemented migration based rebalancing. Vertices migrate to where their neighbors are with a migration quota. Oscillation problems are tackled with a stickiness factor. We experimented with a biomedical simulation of heartbeat stem cells – 3TB memory footprint and 64 compute nodes. A key lesson was that BSP greatly aids system optimizations.

3.24 An Initial Attempt to Classify Graph Processing Systems and Approaches

Tamer Özsu (University of Waterloo, CA, tamer.ozsu@uwaterloo.ca)

License  Creative Commons BY 3.0 Unported license
© Tamer Özsu

There are many graph processing, graph data management systems. This talk represents our first attempt at understanding the design landscape in this area and identifying design points.

Note by Editors: This is an initial attempt to classify graph processing systems and approaches. One dimension for classification is by whether the graphs being tackled are static or dynamic. Included in this dimension is discrimination based on whether the graph access is restricted to streaming. The second dimension is along algorithms: this includes offline, online, streaming and incremental. The third dimension is along whether the workload types being targeted are online queries or analytics over the whole graph. Open questions remain about how to distinguish based on storage (disk/memory), architecture, computing paradigm and the language being used to interface with the system.

4 Joint Session

Two talks from Seminar “14461 High-performance Graph Algorithms and Application in Computational Science” are presented:

1. Peter Sanders (KIT – Karlsruhe Institute for Technology, DE): Parallel multicriteria shortest paths.
2. Andrew Lumsdaine (Indiana University, US): Distributed control for scalable parallel algorithms.

Three talks from Seminar “14462 Systems and Algorithms for Large-scale Graph Analytics” are presented as follows.

4.1 Challenges for Popularizing Graph Applications

Hasan Chafi (Oracle – Belmont, US, hasan.chafi@oracle.com)

Note by Editors: There is a need to cross the chasm between research and industry in terms of ‘graph solutions’. An important question is, who is going to use graph processing platforms? Startups such as Graphlab have rebranded themselves as machine learning toolkits. It is therefore possible that graph analytics as we see it today is early in the hype cycle. One way to move from hype to practical reality is to talk to industry to find use cases and industry specific graph models.

4.2 Graph Analysis Platforms and Evaluations Thereof

Terence Kelly (HP – Palo Alto, US, terence.p.kelly@hp.com)

Note by Editors: Graph analysis platforms today suffer from two problems. The first is an overkill on interface. They often make it harder to write graph algorithms that with a simple text editor and compiler. The second is an overkill on implementation. They often incorporate complex solutions to non-problems. For example, the problem of scalability on inputs that actually don't need it (the graphs are too small).

Both these issues stem from the lack of a good baseline to compare these systems against. Simple single threaded baseline implementations of popular graph algorithms are necessary to properly shed light on these problems. The community also suffers from the lack of large enough datasets. An effort to share large graphs or at least their characteristics would be extremely helpful towards moving forward the state of the art.

4.3 Graphbench.org

Luis Ceze (University of Washington, US luisceze@cs.washington.edu)

Note by Editors: Graphbench is an effort to capture a representative set of graph benchmarks including surrounding systems and environments to enable *standardized* comparisons between systems. Graphbench is an ongoing effort and comments or contributions are most welcome. Some questions and comments were received during the talk itself. An important point was to clearly delineate between different communities when building the benchmark. Should the benchmark be targeted towards the database, systems or HPC community – or all of them? Another important distinction is between graph kernels such as BFS and applications that use those kernels, such as betweenness centrality. Also, should the benchmark be industry focused or mostly aimed at supporting the research community? Finally the dataset itself can have a big impact and therefore decisions such as degree distributions of the input graph will have to be taken as part of the benchmark.

5 Challenges, Future Directions

Eiko Yoneki and Amitabha Roy

License © Creative Commons BY 3.0 Unported license
© Eiko Yoneki and Amitabha Roy

This section highlights the discussion on the challenges and future directions of large-scale graph processing, where we focused on mainly benchmarking and programming model.

Benchmark applications

Current research often focuses on fundamental graph processing algorithms such as traversal, shortest paths and centrality measurement. However benchmarking with more complex applications may be more useful for building production-ready systems. This will also help to motivate adoption by industry. An issue is that customers do not have a good grasp of how

to map their data / business logic into graph data and associated analytical processes. As a result, relying on end users to provide examples of important benchmarking applications may not be feasible. Instead, we may look to broad topics of interest among customers in this field and build benchmark applications that fit those topics. Influence analysis and recommendation systems are examples of topics that currently have a lot of interest in the potential user community. The onus is on graph processing researchers to design benchmarks that fit these use cases.

Tracking the convergence of analytical graph processing systems (i.e. graph databases), it is important to build benchmark suites that accommodate both styles of computation. In particular, graph pattern matching is an area that has gained popularity but lacks a standard set of benchmarks. Related to this, there needs to be more of a focus on “fuzzy” pattern matching (even beyond isomorphism / homomorphism), since this is where the real-world applications and customer interest will focus. “Local density” is another important topic (e.g. triangle counting), insofar as such algorithms often exhibit computation patterns that existing systems have difficulty in accommodating.

A promising effort that aligns with these aims is the LDBC (Linked Data Benchmark Council) [5]. This is an EU project with good participation by industry, but possibly needs more academic involvement to discuss issues similar to those outlined here. The LDBC “GraphBench” benchmark suite has two different types of workloads (query-oriented and analysis-oriented). There are also scalable mechanisms of generating data for the benchmark. GraphBench focuses on “chokepoints” or “known bad” query characteristics that are used to stress systems. One of the reasons for SPEC’s success is its stability, which has set clear expectations for developers, and the aim is for LDBC to provide a similarly comprehensive, scalable standard.

Benchmark datasets

In addition to finding a suite of applications for standardized benchmarking, there is a need to use more varied datasets at the right scale. There has been a tendency in recent graph systems research to test large-scale systems with relatively small-scale data. A major issue is the lack of access to large real-world graph datasets such as social networks. One approach may be to consolidate available datasets into a publicly-accessible repository. A number of repositories are currently available, but need to act as portals (due to a lack of storage). The Milan Laboratory for Web Algorithmic (LAW) [4] is one such portal, the Stanford SNAP database [7] and the Koblenz Network Collection (KONECT) [6] are other examples [1, 3].

With a lack of sufficiently large real-world datasets, an alternative is to synthesize large-scale graph data. However, an issue with graphs is that it is actually a complex problem to synthesize graphs with all the characteristics that have been observed in real networks. There have been efforts (such as the Facebook LinkBench tool) [2] that attempt to at least match some of the characteristics that they have found in smaller datasets into synthesized larger datasets.

Anonymization is a key issue that blocks access to large-scale social data. Social networks are not able to release their data due to legal issues, without some guarantee of privacy. Currently, efforts in this area are largely undertaken by a separate community of security researchers. There needs to be work in this space in order to open up access to large-scale data.

Programming models

There is a general lack of consensus on the “right” programming model for graph systems, and this is an area that warrants further investigation. It may be that a unified programming model is inappropriate, but there may still be potential for consolidating and validating a class of approaches. A theme that is emerging out of the work by Oracle and SAP is that different programming models may be appropriate at different layers of the system, such as the runtime / scheduler, algorithmic level, and query layer. Another important consideration when building scalable systems is also to consider the base case; designing large-scale systems that impose so much overhead that a sequential standalone program can exhibit better performance seems counterproductive. Conversely, this kind of comparison should form an essential part of the evaluation of any new framework.

Follow-up plan

There is an opportunity to consolidate the discussion from this into a survey-style paper. Tamer Ozsu’s group at Waterloo is currently composing a taxonomy of existing systems, and this could be augmented with elements of the discussion from this workshop. There is also an opportunity to share the insights on benchmarking with the LDBC committee, particularly highlighting the need for more datasets. The LDBC needs more types of participants in that forum – users as opposed to just industrial developers.

Following on from this, it may be prudent to launch a new workshop based on use cases, benchmarks and workloads in the graph domain. This could be a joint effort with LDBC. Existing workshops such as GRADES at SIGMOD (Workshop on graph data management experiences and systems) provide a platform from paper-publishing, however there appears to be scope for a more discussion-oriented event.

References

- 1 DIMACS <http://dimacs.rutgers.edu/Challenges/>.
- 2 Facebook LinkBench <http://github.com/facebook/linkbench>.
- 3 Graph500 <http://www.graph500.org/>.
- 4 Laboratory for Web Algorithmics <http://law.di.unimi.it/>.
- 5 LDBC <http://www.ldbc.eu/>.
- 6 Koblenz Network Collection <http://konect.uni-koblenz.de/>.
- 7 Stanford Large Network Dataset Collection <http://snap.stanford.edu/data/index.html>.

Acknowledgement. We thank Derek Murray for his great contribution to the organization of the workshop and participation via Skype as he could not travel because of the sudden set down of Microsoft Research Silicon Valley. We also thank Karthik Nilakant and Valentin Dalibard for their help with editing the report.

Participants

- Paolo Boldi
University of Milan, IT
- Luis Ceze
University of Washington –
Seattle, US
- Hassan Chafi
Oracle Labs – Belmont, US
- Lei Chen
HKUST – Kowloon, HK
- Felix Cuadrado
Queen Mary University of
London, GB
- Valentin Dalibard
University of Cambridge, GB
- Khuzaima Daudjee
University of Waterloo, CA
- Šarunas Girdzijauskas
KTH Royal Institute of
Technology, SE
- Torsten Grust
Universität Tübingen, DE
- Sungpack Hong
Oracle Labs – Belmont, US
- Kimberly Keeton
HP Labs – Palo Alto, US
- Terence P. Kelly
HP Labs – Palo Alto, US
- Arijit Khan
ETH Zürich, CH
- Wolfgang Lehner
TU Dresden, DE
- Andrew Lenharth
University of Texas – Austin, US
- Karthik Nilakant
University of Cambridge, GB
- M. Tamer özsü
University of Waterloo, CA
- Mark H. Oskin
University of Washington –
Seattle, US
- John Owens
Univ. of California – Davis, US
- Roger Pearce
LLNL – Livermore, US
- Ella Peltonen
University of Helsinki, FI
- Stefan Plantikow
Neo Technology Germany, DE
- Amitabha Roy
EPFL – Lausanne, CH
- Sebastian Schelter
TU Berlin, DE
- Alexander Ulrich
Universität Tübingen, DE
- Eiko Yoneki
University of Cambridge, GB
- Willy Zwaenepoel
EPFL – Lausanne, CH

