Systems for Big-Graphs

Arijit Khan[*] Sameh Elnikety[†]

[*]*Systems Group, ETH Zurich, Switzerland* [†]*Microsoft Research, Redmond, WA, USA*

arijit.khan@inf.ethz.ch samehe@microsoft.com

ABSTRACT

Graphs have become increasingly important to represent highly-interconnected structures and schema-less data including the World Wide Web, social networks, knowledge graphs, genome and scientific databases, medical and government records. The massive scale of graph data easily overwhelms the main memory and computation resources on commodity servers. In these cases, achieving low latency and high throughput requires partitioning the graph and processing the graph data in parallel across a cluster of servers. However, the software and and hardware advances that have worked well for developing parallel databases and scientific applications are not necessarily effective for big-graph problems. Graph processing poses interesting system challenges: graphs represent relationships which are usually irregular and unstructured; and therefore, the computation and data access patterns have poor locality. Hence, the last few years has seen an unprecedented interest in building systems for big-graphs by various communities including databases, systems, semantic web, machine learning, and operations research. In this tutorial, we discuss the design of the emerging systems for processing of big-graphs, key features of distributed graph algorithms, as well as graph partitioning and workload balancing techniques. We emphasize the current challenges and highlight some future research directions.

1. INTRODUCTION

Querying and mining of graph data are essential for a wide range of emerging applications [4]. As graph problems grow larger in input size and complexity, they easily overwhelm the computation and memory capacities of a single commodity server. However, graph processing also generates a unique workload [8] as follows:

- **Poor Locality.** Graphs represent relationships which can be irregular and unstructured; and therefore, graph algorithms often exhibit poor locality of memory access.

- **I/O Intensive Operations.** Graph algorithms usually have high data-access-to-computation ratio — the runtime could be dominated by waits for memory fetches.

- **Difficult to Parallelize.** Due to the interconnected nature of graph data, graph computations are irregular: It is difficult to extract parallelism by partitioning. Unbalanced computational workload resulting from poor partitioning and synchronization overheads reduces scalability.

- **Large Intermediate Results.** While executing a graph computation such as parallel graph isomorphism, the intermediate results could be large, with a memory footprint larger than the underlying graph.

In this three-hour tutorial, we first illustrate why state-of-the-art distributed frameworks are not suitable for big-graph computations. For example, MapReduce performs well when the algorithm is *embarrassingly parallel* [7]. Due to the linked structure of graph datasets and the iterative nature of many graph algorithms, it is difficult to represent graph algorithms using the MapReduce abstraction. Several graph-indexing methods were also proposed for specific graph operations. Unfortunately, these indices cannot provide the level of efficient random access required by graph computation and general graph exploration during query processing. For graph operations, keeping data in the main memory is critical as it enables fast random access and the reuse of intermediate results for iterative graph algorithms [17].

In this tutorial, we discuss distributed/shared in-memory big-graph processing systems as well as a few disk-based single-server systems that provide comparable performance. We classify these systems into two broad categories based on their application support: (1) systems for offline graph analytic and (2) systems for online graph querying. We teach the key features of these two types of query workloads, and why we need different systems for them. Finally, we conclude by highlighting major open problems such as dynamic graph partitioning and workload balancing.

2. TUTORIAL OUTLINE

Our tutorial consists of five parts, which are given below:

1. Introduction: We introduce the unique workload characteristics of big-graphs processing, and why state-of-the-art distributed frameworks do not perform well with big-graph computations.

2. Systems for Offline Graph Analytic: Offline graph analytic systems perform an iterative, batch processing over the entire graph dataset until the computation satisfies a fixed-point or stopping criterion. In this section, we first introduce graph algorithms which require iterative, batch processing, e.g., PageRank computation, recursive relational queries, clustering, social network analysis, and machine learning/ data mining algorithms (e.g., belief propagation, gaussian non-negative matrix factorization).

We discuss in details how two offline analytic algorithms, namely PageRank computation and belief propagation can be efficiently implemented in Pregel [9], which is a vertex-centric computation model and follows the original Bulk Synchronous Parallel (BSP) model. Next, we show how the same two algorithms can be implemented even more efficiently in GraphLab [7], which follows *asynchronous* vertex-centric computation, and thereby significantly accelerates the convergence of iterative machine-learning and graph algorithms. We conclude this section by discussing GraphChi [6] and X-Stream [12], which store the graph data in the disk on a single server, yet they provide comparable performance to the existing distributed/ shared in-memory big-graph systems for offline graph analytic algorithms.

3. Systems for Online Graph Querying: Online graph queries, e.g., reachability query, finding the shortest path, sub-graph matching, and SPARQL queries require very fast response time, and these queries often explore only a small fraction of the entire graph dataset, as opposed to the iterative, batch processing over the entire graph dataset. We discuss one domain-specific language (DSL) for online graph querying: Horton [14], and two systems that support online graph traversal: G-SPARQL [13] and Trinity [17]. We further demonstrate why in-memory graph-traversal-based operations are often more effective in answering SPARQL and subgraph matching queries as compared to performing multiple join operations using the traditional database management systems.

4. Dynamic Graph Partitioning: In order to reduce the inter-machine communication, it is important to partition the underlying graph data effectively across multiple servers. However, real-world graphs often exhibit a skewed *power-law* degree distribution; and hence, it is difficult to partition and represent such graphs in a distributed environment. We first show how PowerGraph [2] reduces this problem by performing a balanced vertex-cut of the input graph and by keeping mirrors of cut-vertices at multiple servers. We also discuss SEDGE [16] — a complementary partitioning scheme for reducing the inter-machine communication. Dynamic graph partitioning, on the other hand, is critical for online graph queries, since workloads on the vertices change frequently for online queries. We illustrate dynamic graph partitioning strategies in Mizan [5] and SEDGE [16], and the overlapping partitioning scheme in [10], which updates its partitions dynamically based on the past read/write patterns.

5. Major Open Problems: We conclude by discussing the current challenges and some interesting future research directions.

a. Which one is a better design choice for queries on big-graphs — "scaling out" on cheap, commodity clusters (distributed memory) vs. "scaling up" with more cores and more memory (shared memory)? Perhaps, for online graph queries, scaling up is a better option due to their lower communication cost.

b. Do we need to vary the partitioning and re-partioning strategy based on the graph data, algorithms, and systems?

c. Should we decouple query processors from graph storage so that we can scale up both the layers independently?

d. What will be the roles of modern hardware in accelerating big-graphs processing?

e. Do we need stand-alone systems only for graph processing, such as Trinity and GraphLab; or can they be integrated with the existing big-data and dataflow systems, e.g., GraphX [15], Naidad [11], and epiC [3]?

What we shall not cover in this tutorial. We shall not discuss existing graph databases, such as Neo4j and HyperGraphDB, as they usually cannot manage graphs that are distributed among multiple servers. We do not focus on SPARQL engines and RDF data-stores as they are covered in other tutorials [1]. Finally, we do not discuss specialty hardware systems for big-graphs processing, e.g., Eldorado (shared-memory) and BlueGene/L (distributed-memory), as we focus on software techniques.

3. BIOGRAPHICAL SKETCHES

Arijit Khan is a post-doctorate researcher in the Systems group at ETH Zurich. His research interests span in the area of big-data, big-graphs, and graph systems. He received his PhD from University of California, Santa Barbara. Arijit is the recipient of the prestigious IBM PhD Fellowship in 2012-13. He co-presented a tutorial on emerging queries over linked data at ICDE 2012.

Sameh Elnikety is a researcher at Microsoft Research in Redmond, Washington. He received his Ph.D. from the Swiss Federal Institute of Technology (EPFL) in Lausanne, Switzerland , and M.S. from Rice University in Houston, Texas. His research interests include distributed server systems, and database systems. Sameh's work on database replication received the best paper award at Eurosys 2007.

4. REFERENCES

[1] P. Cudr-Mauroux and S. Elnikety. Graph Data Management Systems for New Application Domains. In *VLDB*, 2011.

[2] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin. PowerGraph: Distributed Graph-parallel Computation on Natural Graphs. In *OSDI*, 2012.

[3] D. Jiang, G. Chen, B. C. Ooi, K.-L. Tan, and S. Wu. epiC: an Extensible and Scalable System for Processing Big Data. In *VLDB*, 2014.

[4] A. Khan, Y. Wu, and X. Yan. Emerging Graph Queries in Linked Data. In *ICDE*, 2012.

[5] Z. Khayyat, K. Awara, A. Alonazi, H. Jamjoom, D. Williams, and P. Kalnis. Mizan: A System for Dynamic Load Balancing in Large-scale Graph Processing. In *EuroSys*, 2013.

[6] A. Kyrola, G. Blelloch, and C. Guestrin. GraphChi: Large-scale Graph Computation on Just a PC. In *OSDI*, 2012.

[7] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein. Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud. In *VLDB*, 2012.

[8] A. Lumsdaine, D. Gregor, B. Hendrickson, and J. W. Berry. Challenges in Parallel Graph Processing. *Parallel Processing Letters*, 17(1):5–20, 2007.

[9] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: A System for Large-scale Graph Processing. In *SIGMOD*, 2010.

[10] J. Mondal and A. Deshpande. Managing Large Dynamic Graphs Efficiently. In *SIGMOD*, 2012.

[11] D. G. Murray, F. McSherry, R. Isaacs, M. Isard, P. Barham, and M. Abadi. Naiad: a Timely Dataflow System. In *SOSP*, 2013.

[12] A. Roy, I. Mihailovic, and W. Zwaenepoel. X-Stream: Edge-centric Graph Processing Using Streaming Partitions. In *SOSP*, 2013.

[13] S. Sakr, S. Elnikety, and Y. He. G-SPARQL: a Hybrid Engine for Querying Large Attributed Graphs. In *CIKM*, 2012.

[14] M. Sarwat, S. Elnikety, Y. He, and M. F. Mokbel. Horton+: A Distributed System for Processing Declarative Reachability Queries over Partitioned Graphs. In *VLDB*, 2013.

[15] R. S. Xin, D. Crankshaw, A. Dave, J. E. Gonzalez, M. J. Franklin, and I. Stoica. GraphX: Unifying Data-Parallel and Graph-Parallel Analytics. *CoRR*, abs/1402.2394, 2014.

[16] S. Yang, X. Yan, B. Zong, and A. Khan. Towards Effective Partition Management for Large Graphs. In *SIGMOD*, 2012.

[17] K. Zeng, J. Yang, H. Wang, B. Shao, and Z. Wang. A Distributed Graph Engine for Web Scale RDF Data. In *VLDB*, 2013.