Graph Processing: A Panoramic View and Some Open Problems

M. Tamer Özsu

University of Waterloo
David R. Cheriton School of Computer Science
https://cs.uwaterloo.ca/~tozsu
Graph Research is Dispersed

Knowledge graphs
Semantic Web

Graph Databases

Graph Analytics
Graph Research is Dispersed

- Graph Theory
- Graph Algorithms
- Graph Systems
Graph Research is Dispersed
Graph Research is Dispersed

Knowledge graphs
Semantic Web

Graph Databases

Graph Analytics

Graph Theory

Graph Algorithms

Graph Systems

Database

Data Mining

Social Computing

AI/ML
My objectives

Three things...

1. Discuss a way to coherently position work in the various communities;
2. A tour across different communities to provide a panoramic view of the research;
3. Highlight some problems that interest me!
My objectives

Three things...

1. Discuss a way to coherently position work in the various communities;
2. A tour across different communities to provide a panoramic view of the research;
3. Highlight some problems that interest me!

Knowledge graphs
Semantic web
GraphDBMSs
GraphAnalyticsSystems
RDF graph
Property graph
So yesterday I gave my lecture. Despite a lack of preparation, I spoke quite well and without any hesitation, which I ascribe to the cocaine I had taken beforehand. I told about my discoveries in brain anatomy, all very difficult things that the audience certainly didn’t understand, but all that matters is that they get the impression that I understand it. . . . It was good company: Billroth, Nothnagel, Breuer, etc., etc.—rare people who haven’t taken
In the beginning...

... there was IMS

- By IBM (along with Rockwell & Caterpillar)
- For the Apollo program
- First deployed in 1968
- Managing Bill of Materials (BOM) of Saturn rocket
- Hierarchical model because BOM is hierarchical
In the beginning...

... and IDS

- By GE
- To control their manufacturing processes
- First deployed in 1964
- Manufacturing processes (with scheduling constraints) form a graph
- Network model
- Led to CODASYL standard

Source: https://dba.stackexchange.com/questions/119380/er-vs-database-schema-diagrams
CODASYL Language

- FIND with key
- Navigate within the set, within elements of the same record type, etc

Network DBMSs

CODASYL Language

- FIND with key
- Navigate within the set, within elements of the same record type, etc

Network models were also used in

- Object DBMSs
- XML

Network DBMSs

Network (CODASYL) Data Model

1. Products and all transactions which were done by this company.
2. Computers and all transactions which were done by these customers.
3. Representatives who sell the products.
4. Sales transactions.

Data Sets might look as follows:
1. Products and all transactions which were done by this company.
2. Computers and all transactions which were done by these customers.
3. Representatives who sell the products.
4. Sales transactions.

The main updating facilities of a concrete DML can be also divided into two parts:
(i) data manipulation facilities - each data manipulation language (DML) also includes particular data manipulation facilities of a concrete DML.
(ii) data retrieve functions.

Putting a new record occurrence into a database:
1. To store a new record occurrence into a database;
2. To modify a current state of the database;
3. To add a new occurrence of the record type, etc.
4. To insert existing occurrences of the record type.
5. To remove an existing record occurrence.

Example:
• Consider a company that produces and sells personal computers.
• The company has a representative who sells personal computers.
• The company sells personal computers.
• Sales transactions are done between a company and a customer.

Network models were also used in
- Object DBMSs
- XML

Modern graphs are different and diverse

Internet

Social networks

Trade volumes and connections

Biological networks

Linked data

Road network
Graph Usage Study

[Sahu et al., 2017, 2019]

Objectives

1. What kind of graph data, computations, software, and major challenges real users have in practice?
2. What types of graph data, computations, software, and major challenges researchers target in publications?

Methodology

- **Online survey**
  - 89 participants: 36 researchers; 53 industry
  - 22 graph software products

- **Review of academic publications**
  - 7 conferences, 3 yrs for each
  - 252 papers

- **Review of emails, bug reports, and feature requests**
  - over 6000 emails and issues

- **Personal interviews**
  - 4 interviews with survey participants
  - 4 additional in-person interviews: 2 developers and 2 users

- **Applications from white papers**
  - 4 graph DBMSs + 4 RDF engines
  - 12 applications from graph DBMSs + 5 from RDF engines (with overlap)
1. Graphs are indeed everywhere!
Major Findings

1. Graphs are indeed everywhere!

   Q1. Which real world entities do your graphs represent?

<table>
<thead>
<tr>
<th>Entity</th>
<th>#Part.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Humans (e.g., customers, friends)</td>
<td>44</td>
</tr>
<tr>
<td>Non-human entities (e.g., web, products, files)</td>
<td>61</td>
</tr>
<tr>
<td>RDF</td>
<td>23</td>
</tr>
<tr>
<td>Scientific (e.g., proteins, molecules, bonds)</td>
<td>15</td>
</tr>
</tbody>
</table>

[Sahu et al., 2017, 2019]
Major Findings

1. Graphs are indeed everywhere!
   Q1. Which real world entities do your graphs represent?
   Q2. Which non-human entities do your graphs represent?

<table>
<thead>
<tr>
<th>Entity</th>
<th>#Part.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web</td>
<td>4</td>
</tr>
<tr>
<td>Bus. &amp; Financial</td>
<td>8</td>
</tr>
<tr>
<td>Product</td>
<td>13</td>
</tr>
<tr>
<td>Geo</td>
<td>7</td>
</tr>
<tr>
<td>Infrastructure</td>
<td>9</td>
</tr>
<tr>
<td>Digital</td>
<td>5</td>
</tr>
<tr>
<td>Knowledge</td>
<td>11</td>
</tr>
<tr>
<td>Humans (e.g., customers, friends)</td>
<td>44</td>
</tr>
<tr>
<td>Non-human entities (e.g., web, products, files)</td>
<td>61</td>
</tr>
<tr>
<td>RDF</td>
<td>23</td>
</tr>
<tr>
<td>Scientific (e.g., proteins, molecules, bonds)</td>
<td>15</td>
</tr>
</tbody>
</table>
Major Findings

1. Graphs are indeed everywhere!

   Q1. Which real world entities do your graphs represent?
   Q2. Which non-human entities do your graphs represent?

<table>
<thead>
<tr>
<th>Entity</th>
<th>#Part.</th>
<th>#Papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web</td>
<td>4</td>
<td>30</td>
</tr>
<tr>
<td>Bus. &amp; Financial</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Product</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>Geo</td>
<td>7</td>
<td>11</td>
</tr>
<tr>
<td>Infrastructure</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>Digital</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>Knowledge</td>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>Humans</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(e.g., customers, friends)</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>Non-human entities</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(e.g., web, products, files)</td>
<td>61</td>
<td></td>
</tr>
<tr>
<td>RDF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scientific</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(e.g., proteins, molecules, bonds)</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>
Major Findings

1. Graphs are indeed everywhere!
   Q1. Which real world entities do your graphs represent?
   Q2. Which non-human entities do your graphs represent?

2. Graphs are indeed very large!

<table>
<thead>
<tr>
<th>E</th>
<th>#Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;10K</td>
<td>23</td>
</tr>
<tr>
<td>10K–100K</td>
<td>22</td>
</tr>
<tr>
<td>100K–1M</td>
<td>13</td>
</tr>
<tr>
<td>1M–10M</td>
<td>9</td>
</tr>
<tr>
<td>10M–100M</td>
<td>21</td>
</tr>
<tr>
<td>100M–1B</td>
<td>21</td>
</tr>
<tr>
<td>&gt;1B</td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#Bytes</th>
<th>#Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;10MB</td>
<td>23</td>
</tr>
<tr>
<td>10MB–100MB</td>
<td>22</td>
</tr>
<tr>
<td>100MB–1GB</td>
<td>13</td>
</tr>
<tr>
<td>1GB–10GB</td>
<td>9</td>
</tr>
<tr>
<td>10GB–100GB</td>
<td>21</td>
</tr>
<tr>
<td>100GB–1TB</td>
<td>21</td>
</tr>
<tr>
<td>&gt;1TB</td>
<td>20</td>
</tr>
</tbody>
</table>
Major Findings

[Sahu et al., 2017, 2019]

1. Graphs are indeed everywhere!
   Q1. Which real world entities do your graphs represent?
   Q2. Which non-human entities do your graphs represent?

2. Graphs are indeed very large!

3. ML on graphs is very popular!
   - At least 68% of respondents use ML workload
1. Graphs are indeed everywhere!
   Q1. Which real world entities do your graphs represent?
   Q2. Which non-human entities do your graphs represent?

2. Graphs are indeed very large!

3. ML on graphs is very popular!
   - At least 68% of respondents use ML workload

4. Scalability is the most pressing challenge!
   - Followed by visualization & query languages
Major Findings

[1] Graphs are indeed everywhere!
   Q1. Which real world entities do your graphs represent?
   Q2. Which non-human entities do your graphs represent?

[2] Graphs are indeed very large!

[3] ML on graphs is very popular!
   - At least 68% of respondents use ML workload

[4] Scalability is the most pressing challenge!
   - Followed by visualization & query languages

[5] RDBMS still play an important role!
The Ubiquity of Large Graphs and Surprising Challenges of Graph Processing

Siddhartha Sahu, Amine Mhedhbi, Semih Salihoglu, Jimmy Lin, M. Tamer Özsu
David R. Cheriton School of Computer Science
University of Waterloo
{s3sahu,amine.mhedhbi,semih.salihoglu,jimmylin,tamer.ozsu}@uwaterloo.ca

ABSTRACT

Graph processing is becoming increasingly prevalent across many application domains. In spite of this prevalence, there is little research about how graphs are actually used in practice. We conducted an online survey aimed at understanding: (i) the types of graphs users have; (ii) the graph computations users run; (iii) the types of graph software users use; and (iv) the major challenges users face when processing their graphs. We describe the participants’ responses to our questions highlighting common patterns and challenges. We further reviewed user feedback in the mailing lists, bug reports, and feature requests in the source repositories of a large suite of software products for processing graphs. Through our review, we were able to answer some new questions that were raised by participants’ responses and identify specific challenges that users face when using different classes of graph software. The participants’ responses and data we obtained revealed surprising facts about graph processing in practice. In particular, real-world graphs represent a very diverse range of entities and are often very large, and scalability and visualization are undeniably the most pressing challenges faced by participants. We hope these findings can guide future research.

PVLDB Reference Format:
DOI: https://doi.org/10.14778/3164135.3164139

1. INTRODUCTION

Graph data representing connected entities and their relationships appear in many application domains, most naturally in social networks, the web, the semantic web, road maps, communication networks, biology, and finance, just to name a few examples. There has been a noticeable increase in the prevalence of work on graph processing both in research and in practice, evidenced by the surge in the number of different commercial and research software for managing and processing graphs. Examples include graph database systems [5, 14, 54, 55], RDF engines [38, 64, 67], linear algebra software [6, 40], visualization software [13, 16], query languages [28]. Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Articles from this volume were invited to present their results at The 46th International Conference on Very Large Data Bases, August 2016, Rio de Janeiro, Brazil. Proceedings of the VLDB Endowment. Vol. 11, No. 4
Copyright 2017 VLDB Endowment 2150-8097/17/02. 10. 00. DOI: https://doi.org/10.14778/3164135.3164139

S2, 55], and distributed graph processing systems [17, 21, 27]. In the academic literature, a large number of publications that study numerous topics related to graph processing regularly appear across a wide spectrum of research venues. Despite their prevalence, there is little research on how graph data is actually used in practice. We performed an extensive study that consisted of an online survey of 89 users, a review of the mailing lists, source repositories, and white papers of a large suite of graph software products, and in-person interviews with 6 users and 2 developers of these products. Our online survey aimed at understanding: (i) the types of graphs users have; (ii) the graph computations users run; (iii) the types of graph software users use; and (iv) the major challenges users face when processing their graphs. We describe the participants’ responses to our questions highlighting common patterns and challenges. Based on our interviews and survey of the rest of our sources, we were able to answer some new questions that were raised by participants’ responses to our online survey and understand the specific applications that use graph data and software. Our study revealed surprising facts about graph processing in practice. In particular, real-world graphs represent a very diverse range of entities and are often very large, scalability and visualization are undeniably the most pressing challenges faced by participants, and data integration, recommendations, and fraud detection are very popular applications supported by existing graph software. We hope these findings can guide future research.

Keywords User survey · Graph processing · Graph databases · RDF systems

1 Introduction

Graph data representing connected entities and their relationships appear in many application domains, most naturally in social networks, the web, the Semantic Web, road maps, communication networks, biology, and finance, just to name a few examples. There has been a noticeable increase in the prevalence of work on graph processing both in research and in practice, evidenced by the surge in the number of different commercial and research software for managing and processing graphs. Examples include graph database systems [13, 20, 26, 49, 65, 73], RDF engines [52, 60], linear algebra software [17, 63], visualization software [25, 29], query languages [41, 72, 78], and distributed graph processing systems [30, 34, 40]. In the academic literature, a large number of publications that study numerous topics related to graph processing regularly appear across a wide spectrum of research venues. Despite their prevalence, there is little research on how graph data are actually used in practice and the major challenges facing users of graph data, both in industry and research. In April 2017, we conducted an online survey across 89 users of 22 different software products, with the goal of answering 4 high-level questions:

(i) What types of graph data do users have?
(ii) What computations do users run on their graphs?
(iii) Which software do users use to perform their computations?
(iv) What are the major challenges users face when processing their graph data?

Our major findings are as follows:

1. Variety: Graphs in practice represent a very wide variety of entities, many of which are not naturally thought of as vertices and edges. Most surprisingly, traditional data comprised of products, orders, and transactions, which are typically seen as the perfect fit for relational systems, appear to be a very common form of data represented in participants’ graphs.
2. Ubiquity: Many graphs in practice are very large, often containing over a billion edges. These large graphs represent a very wide range of entities and belong to organizations at all scales from very small enterprises to very large ones. This refurbishes the sometimes heard assumption that large graphs are a problem for only a few large organizations.
3. Scalability: Scalability is unequivocally the most pressing challenge faced by participants. The ability to process very large graphs efficiently seems to be the biggest limitation of existing software.
4. Visualization: Visualization is a very popular and central task in participants’ graph processing pipelines. After scalability, participants indicated visualization as their second most pressing challenge, tied with challenges in graph query languages.
5. Prevalence of RDBMSes: Relational databases still play an important role in managing and processing graphs.

Our survey also highlights other interesting facts, such as the prevalence of machine learning on graph data, e.g., for clustering vertices, predicting links, and finding influential vertices. We further reviewed user feedback in the mailing lists, bug reports, and feature requests in the source code repositories of 22 software products between January and September of 2017 with two goals: (i) to answer several new questions that the participants’ responses raised, and (ii) to identify more specific challenges in different classes of graph technologies than the ones we could identify.
How I Think of This Domain!

Graph Types
- RDF Graphs
- Property Graphs

Graph Dynamism
- Static Graphs
- Dynamic Graphs
- Streaming Graphs

Algorithm Types
- Offline
- Online
- Dynamic

Workload Types
- Online Queries
- Analytics Workloads

Graph Characteristics
- Degree distribution; max degree
- Diameter
- Global/local density
- Connectedness
- Directed/undirected
- Weighted/unweighted
- Homogeneous/heterogeneous
- Simple/multi/hyper
How I Think of This Domain!

Graph Types
- RDF Graphs
- Property Graphs

Graph Dynamism
- Static Graphs
- Dynamic Graphs
- Streaming Graphs

Algorithm Types
- Offline Algorithms
- Online Algorithms
- Dynamic Algorithms

Workload Types
- Online Queries
- Analytics Workloads

Graph Characteristics
- Degree distribution; max degree
- Diameter
- Global/local density
- Connectedness
- Directed/undirected
- Weighted/unweighted
- Homogeneous/heterogeneous
- Simple/multi/hyper

© M. Tamer Özsu
VLDB 2019 (2019/08/27)
How I Think of This Domain!

Graph Types
- RDF Graphs
- Property Graphs
- Static Graphs
- Dynamic Graphs
- Streaming Graphs

Graph Dynamism
- Static Graphs
- Dynamic Graphs
- Streaming Graphs

Algorithm Types
- Offline Algorithms
- Online Algorithms
- Dynamic Algorithms

Workload Types
- Online Queries
- Analytics Workloads

Graph Characteristics
- Degree distribution; max degree
- Diameter
- Global/local density
- Connectedness
- Directed/undirected
- Weighted/unweighted
- Homogeneous/heterogeneous
- Simple/multi/hyper
How I Think of This Domain!

Graph Types
- RDF Graphs
- Property Graphs
- Static Graphs
- Dynamic Graphs
- Streaming Graphs

Graph Dynamism
- Static Graphs
- Dynamic Graphs
- Streaming Graphs

Algorithm Types
- Offline Algorithms
- Online Algorithms
- Dynamic Algorithms

Workload Types
- Online Queries
- Analytics Workloads

Graph Characteristics
- Degree distribution; max degree
- Diameter
- Global/local density
- Connectedness
- Directed/undirected
- Weighted/unweighted
- Homogeneous/heterogeneous
- Simple/multi/hyper
How I Think of This Domain!

Graph Types
- RDF Graphs
- Property Graphs

Graph Dynamism
- Static Graphs
- Dynamic Graphs
- Streaming Graphs

Algorithm Types
- Offline
- Online
- Dynamic

Workload Types
- Online Queries
- Analytics Workloads

Graph Characteristics
- Degree distribution; max degree
- Diameter
- Global/local density
- Connectedness
- Directed/undirected
- Weighted/unweighted
- Homogeneous/heterogeneous
- Simple/multi/hyper
Example Design Points

Graph Type
- RDF Graphs
- Property Graphs
- Static Graphs

Graph Dynamism
- Dynamic Graphs
- Streaming Graphs

Algorithm Types
- Offline
- Online
- Dynamic

Workload Types
- Online Queries
- Analytics Workloads

Compute the query result over the graph as it exists.
Example Design Points

Graph Type
- RDF Graphs
- Property Graphs

Graph Dynamism
- Static Graphs
- Dynamic Graphs
- Streaming Graphs

Algorithm Types
- Offline
- Online
- Dynamic

Workload Types
- Online Queries
- Analytics Workloads

Compute the query result over the graph incrementally.
Example Design Points

Perform the analytic computation from scratch on each snapshot.
Dynamic (or batch-dynamic) algorithms do not make sense for static graphs.
Alternative Classification

- Inputs
  - Input ingestion
    - Generative model
    - Queued/non-queued
    - ...
  - Input data
    - Graph type
    - Graph characteristics
  - Graph dynamism
    - ...
  - Input workload
    - ...

- Processing
  - Algorithms ...

- Output
  - Output generation (or release) time
  - Output type
  - Output interface
Graph System Architectural Design Decisions

- Disk-based vs memory-based
  - Most graph analytics systems are memory-based
  - Others are mixed
- Scale-up vs scale-out
  - Controversial point discussed next
- Computing paradigm
  - A number of alternatives exist
  - Discussed separately for each type of system
Scale-up or Scale-out?

- **Scale-up**: Single machine execution
  - Graph datasets are small and can fit in a single machine – even in main memory
  - Single machine avoids parallel execution complexities (multithreading is a different issue)
Scale-up or Scale-out?

**Scale-up:** Single machine execution

**Scale-out:** Parallel (cluster) execution

- Graph data sets grow when they are expanded to their storage formats
- Workstations of appropriate size are still expensive
- Some graphs are very large: Billions of vertices, hundreds of billions of edges
- Dataset size may not be the only factor → parallelizing computation is important
- Applications may operate in a distributed environment
- Downside: graph partitioning is difficult

| Dataset      | |V| |E| | Regular size | Single Machine* |
|--------------|---|---|---|---------------|-----------------|
| Live Journal | 4,847,571 | 68,993,773 | 1.08GB | 6.3GB         |
| USA Road     | 23,947,347 | 58,333,344 | 951MB  | 9.09GB        |
| Twitter      | 41,652,230 | 1,468,365,182 | 26GB  | 128 GB        |
| UK0705       | 82,240,700 | 2,829,101,180 | 48GB  | 247GB         |
| World Road   | 682,496,072 | 717,016,716 | 15GB  | 194GB         |
| CommonCrawl2014 | 1,727,000,000 | 64,422,000,000 | 1.3TB | Out of memory |

* Using (PowerLyra)
Scale-up or Scale-out?

- **Scale-up**: Single machine execution
- **Scale-out**: Parallel (cluster) execution

Scale-out is the only way to go!...

There is no way to deal with the emerging real graph sizes on single (ordinary) machines.
Response to “Scale Up or Scale Out for Graph Processing”

Our colleague Jimmy Lin in the University of Waterloo’s Data Systems Group wrote an article for this department giving his perspective on whether organizations should scale up or scale out for graph analytics. Similarly to that article, for rhetorical convenience, we use “scale up” to refer to using software running on multicore large-memory machines and “scale out” to refer to using distributed software running on multiple machines.

It is difficult to disagree with the central message of Jimmy’s article: For many organizations that have large-scale graphs and want to run analytical computations, using a multicore single machine with a lot of RAM is a better option than a distributed cluster because single-machine software, compared to distributed software, is easier to develop in-house or use out of the box, is often more efficient, and is easier to maintain. This is indeed true, and for the social-network graphs and the computations discussed in that article—e.g., a search for a diamond structure or an online random-walk computation for recommendations—scale-up is likely the better approach. However, Jimmy’s article gave the impression that only a handful of applications require scale-out computing, and it failed to highlight several common scenarios in which scale-out is necessary.

In this response article, we discuss three cases for scale-out:

- **Trillion-edge-size graphs.** Several application domains, such as finance, retail, e-commerce, telecommunications, and scientific computing, have naturally appearing graphs at the trillion-edge-size scale.

---

### Table: CommonCrawl2014

<table>
<thead>
<tr>
<th>Dataset</th>
<th>PowerLyra</th>
<th>UK0705</th>
<th>Twitter</th>
<th>Live Journal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size (E)</td>
<td>6.3GB</td>
<td>951MB</td>
<td>1.08GB</td>
<td>39,333,344</td>
</tr>
<tr>
<td></td>
<td>247GB</td>
<td>951MB</td>
<td>39,333,344</td>
<td>26GB</td>
</tr>
<tr>
<td></td>
<td>4,847,571</td>
<td>128 GB</td>
<td>1,727,000,000</td>
<td>10GB</td>
</tr>
<tr>
<td></td>
<td>951MB</td>
<td>247GB</td>
<td>1,727,000,000</td>
<td>10GB</td>
</tr>
<tr>
<td></td>
<td>1.3TB</td>
<td>247GB</td>
<td>1,727,000,000</td>
<td>10GB</td>
</tr>
<tr>
<td></td>
<td>9.09GB</td>
<td>247GB</td>
<td>1,727,000,000</td>
<td>10GB</td>
</tr>
</tbody>
</table>

---

**Note:** Jimmy’s article is a great read, but it doesn’t cover several important points I would like to discuss in this response. The main scenarios I want to cover are trillion-edge-size graphs, which are common in various domains, and the fact that scale-out computing is often necessary for organizations that are not yet large enough to experience such explosive growth.
Graph Systems

There are a number of others!...
There are a number of others!...
RDF Engines
### RDF Example Instance

Prefixes:
- `mdb=http://data.linkedmdb.org/resource/`  
- `geo=http://sws.geonames.org/`  
- `bm=http://wifo5-03.informatik.uni-mannheim.de/bookmashup/`  
- `lexvo=http://lexvo.org/id/`  
- `wp=http://en.wikipedia.org/wiki/`  

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>mdb:film/2014</td>
<td>movie:director</td>
<td>mdb:director/8476</td>
</tr>
<tr>
<td>mdb:film/2014</td>
<td>movie:music_contributor</td>
<td>mdb: music_contributor/4110</td>
</tr>
<tr>
<td>geo:2635167</td>
<td>gn:name</td>
<td>“United Kingdom”</td>
</tr>
<tr>
<td>geo:2635167</td>
<td>gn:population</td>
<td>62348447</td>
</tr>
<tr>
<td>bm:books/0743424425</td>
<td>dc:creator</td>
<td>bm:persons/Stephen+King</td>
</tr>
<tr>
<td>bm:books/0743424425</td>
<td>rev:rating</td>
<td>4.7</td>
</tr>
</tbody>
</table>

© M. Tamer Özsu
SELECT ?name
WHERE {
  ?m rdfs:label ?name .
  ?m movie:director ?d .
  ?d movie:director_name "Stanley Kubrick" .
  ?b rev:rating ?r .
  FILTER(?r > 4.0)
}
Direct Relational Mapping
Bad Idea!...
Direct Relational Mapping

SELECT ?name
WHERE {
  ?m rdfs:label ?name .
  ?m movie:director ?d .
  ?d movie:director_name "Stanley Kubrick" .
  ?b rev:rating ?r .
  FILTER (?r > 4.0)
}

SELECT T1.object
FROM T as T1, T as T2, T as T3, T as T4, T as T5
WHERE T1.p="rdfs:label"
AND T2.p="movie:relatedBook"
AND T3.p="movie:director"
AND T4.p="rev:rating"
AND T5.p="movie:director_name"
AND T1.s=T2.s
AND T1.s=T3.s
AND T2.o=T4.s
AND T3.o=T5.s
AND T4.o > 4.0
AND T5.o="Stanley Kubrick"
Direct Relational Mapping

```
SELECT ?name
WHERE {
  ?d movie:director_name "Stanley Kubrick" .
  FILTER (?r > 4.0)
}
```

**Easy to implement but too many self-joins!**

---

© M. Tamer Özsu
VLDB 2019 (2019/08/27) 23
<table>
<thead>
<tr>
<th>Objectives</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Eliminate/Reduce the number of self-joins</td>
</tr>
<tr>
<td>2. Use merge-join</td>
</tr>
</tbody>
</table>

**Approaches**

1. **Property table**
   - Group together the properties that tend to occur in the same (or similar) subjects
   - Examples: Jena [Wilkinson, 2006], DB2-RDF [Bornea et al., 2013]

2. **Vertically partitioned tables**
   - For each property, build a two-column table, containing both subject and object, ordered by subjects
   - Binary tables [Abadi et al., 2007, 2009]

3. **Exhaustive indexing**
   - Create indexes for each permutation of the three columns: SPO, SOP, PSO, POS, OPS, OSP
   - RDF-3X [Neumann and Weikum, 2008, 2009], Hexastore [Weiss et al., 2008]
Optimizations to Tabular Representation

Objectives

1. Eliminate/Reduce the number of self-joins
2. Use merge-join

Approaches

1. Property table
   - Group together the properties that tend to occur in the same (or similar) subjects
   - Examples: Jena [Wilkinson, 2006], DB2-RDF [Bornea et al., 2013]
2. Vertically partitioned tables
   - For each property, build a two-column table, containing both subject and object, ordered by subjects
   - Binary tables [Abadi et al., 2007, 2009]
3. Exhaustive indexing
   - Create indexes for each permutation of the three columns: SPO, SOP, PSO, POS, OPS, OSP
   - RDF-3X [Neumann and Weikum, 2008, 2009], Hexastore [Weiss et al., 2008]
Graph-based Approach

- Answering SPARQL query $\equiv$ subgraph matching using homomorphism
- gStore [Zou et al., 2011, 2014], chameleon-db [Aluç et al., 2013]
Graph-based Approach

- Answering SPARQL query $\equiv$ subgraph matching using homomorphism
- gStore [Zou et al., 2011, 2014], chameleon-db [Aluç et al., 2013]

**Advantages**

- Maintains the graph structure
- Full set of queries can be handled

**Disadvantages**

- Graph pattern matching is expensive

FILTER(?r > 4.0)
Scaling-out RDF Engines

- Cloud-based solutions [Kaoudi and Manolescu, 2015]
  - RDF dataset $D$ is partitioned into $\{D_1, \ldots, D_n\}$ and placed on cloud platforms (such as HDFS, HBase)
  - SPARQL query is run through MapReduce jobs
  - Data parallel execution
  - Examples: HARD [Rohloff and Schantz, 2010], HadoopRDF [Husain et al., 2011], EAGRE [Zhang et al., 2013] and JenaHBase [Khadilkar et al., 2012]
Scaling-out RDF Engines

- Cloud-based solutions [Kaoudi and Manolescu, 2015]

- Partition-based approaches
  - Partition an RDF dataset $D$ into fragments $\{D_1, \ldots, D_n\}$ each of which is located at a site
  - SPARQL query $Q$ is decomposed into a set of subqueries $\{Q_1, \ldots, Q_k\}$
  - Distributed execution of $\{Q_1, \ldots, Q_k\}$ over $\{D_1, \ldots, D_n\}$
  - Examples: GraphPartition [Huang et al., 2011], WARP [Hose and Schenkel, 2013], Partout [Galarraga et al., 2014], Vertex-block [Lee and Liu, 2013]
Scaling-out RDF Engines

- Cloud-based solutions [Kaoudi and Manolescu, 2015]
- Partition-based approaches
- Partial Query Evaluation (PQE)
  - Partition an RDF dataset $D$ into several fragments \(\{D_1, \ldots, D_n\}\) each of which is located at a site
  - SPARQL query is not decomposed; the full query is sent to each site
  - PQE at each site producing partial results
  - Join the results (similar to distributed join processing) to find matching edges that might cross fragments
- Distributed gStore [Peng et al., 2016]
What are Some Open Issues?

- These systems are not performant or scalable to large data sets
  - What is the right scale-out architecture and techniques?
  - It is not clear what the best storage format is
  - Optimization of SPARQL queries
  - RDF Engines require more experimentation
What are Some Open Issues?

- These systems are not performant or scalable to large data sets
- How to implement SPARQL **fully**
  - Current focus on basic graph patterns (sets of triple patterns)
  - Additional constructs, e.g., property paths, OPTIONAL, UNION, FILTER, aggregation, ...
  - Reasoning over RDF needs to be considered ⇒ entailment regimes (see *Ontologies and Semantic Web*, [Tena Cucala et al., 2019])

Data quality over RDF datasets is a real issue
What are Some Open Issues?

- These systems are not performant or scalable to large data sets
- How to implement SPARQL fully
- Support for dynamic and streaming RDF graphs
  - Few existing systems (e.g., C-SPARQL [Barbieri et al., 2010] and CQUELS[Phuoc et al., 2011]) are early attempts; more work required
What are Some Open Issues?

- These systems are not performant or scalable to large data sets
- How to implement SPARQL **fully**
- Support for dynamic and streaming RDF graphs
- Data quality over RDF datasets is a real issue
  - Some initial work exists, e.g., CLAMS [Farid et al., 2016]
What are Some Open Issues?

- These systems are not performant or scalable to large data sets
- How to implement SPARQL **fully**
- Support for dynamic and streaming RDF graphs
- Data quality over RDF datasets is a real issue
- RDF in IoT
  - Both as a common model across devices, and
  - as embedded into devices
What are Some Open Issues?

- These systems are not performant or scalable to large data sets
- How to implement SPARQL **fully**
- Support for dynamic and streaming RDF graphs
- Data quality over RDF datasets is a real issue
- RDF in IoT

⚠️ Most important ...

1. DB community really needs to get engaged to develop performant & scalable engines
2. SPARQL is not easy ⇒ language front-ends are desperately needed
3. Proper implementation of full SPARQL with optimizations for performance & scalability
Graph DBMSs
**Graph DBMS Properties**

- **Property graph model**
  - Vertices and edges have one or more **labels** and zero or more **properties**
  - Graph can be directed or undirected

```text
film
2014
(initial_release_date, "1980-05-23")
(label, "The Shining")
(music_contributor, music_contributor/4110)
(language, (iso639_3/eng)
(label, "English")
(usedIn, iso3166/CA)
(usesScript, script/latn))

books_0743424425
(rating, 4.7)
(creator, StephenKing)
(relatedBook)

film_3418
(label, "The Passenger")

geo_2635167
(name, "United Kingdom")
(population, 62348447)
(based_near)

UnitedKingdom
(wikipediaArticle)

actor_29704
(actor_name, "Jack Nicholson")
(relatedBook)

film_1267
(label, "The Last Tycoon")

actor_30013
(actor_name, "Shelley Duvall")
(hasOffer)

director_8476
(director_name, "Stanley Kubrick")
(hasOffer)

film_2685
(label, "A Clockwork Orange")
(hasOffer)

film_424
(label, "Spartacus")
(hasOffer)
```
Graph DBMS Properties

● Property graph model
  ● Vertices and edges have one or more labels and zero or more properties
  ● Graph can be directed or undirected

● Online workloads
  ● Each query accesses a portion of the graph
  ● Can be assisted by indexes
  ● Query latency is important
  ● Examples
    ● Reachability
    ● Single source shortest-path
    ● Subgraph matching
    ● SPARQL queries
Graph DBMS Properties

- **Property graph model**
  - Vertices and edges have one or more *labels* and zero or more *properties*
  - Graph can be directed or undirected

- **Online workloads**

- **Graph query languages**
  - Regular Queries [Reutter et al., 2017]
    - Unions of Conjunctive Nested 2-Way Regular Path Queries (UC2NRPQ)
    - Formalism for structural graph queries
  - Cypher (Neo4j)
    - Declarative, comparable to UCRPQ
  - Gremlin (TinkerPop)
    - Imperative, XPath like navigation language
  - G-Core (LDBC) [Angles et al., 2018]
    - Declarative, graphs as first-class citizens
  - PGQL (Oracle PGX)
    - SQL-like language with pattern matching and reachability
  - GSQL (TigerGraph)
Property Graph Storage Approaches

- **Key-value stores**
  - Vertices are keys, entire edge and property information is stored in the value
  - Examples: Titan (Datastax Enterprise Graph), JanusGraph, Dgraph
Property Graph Storage Approaches

- Key-value stores
  - Vertices are keys, entire edge and property information is stored in the value
  - Examples: Titan (Datastax Enterprise Graph), JanusGraph, Dgraph

- Ternary tables
  - Each edge, vertex and property is a separate record
  - E.g., Neo4j keeps data in separate files, each of which holds data of one type (nodes/relations/properties)
Property Graph Storage Approaches

- **Key-value stores**
  - Vertices are keys, entire edge and property information is stored in the value
  - Examples: Titan (Datastax Enterprise Graph), JanusGraph, Dgraph

- **Ternary tables**
  - Each edge, vertex and property is a separate record
  - E.g., Neo4j keeps data in separate files, each of which holds data of one type (nodes/relations/properties)

- **Pivoted tables**
  - Similar to above, tables are pivoted
  - This is similar to property table approach in RDF engines
  - Each column is a property key and the column value is the property value
  - All properties of a vertex is stored in a single row
  - Examples: SAP Hana Graph, IBM SQLGraph
Graph Querying

- Querying graph **topology and graph properties**
  - Data queries are essentially relational queries
  - Querying these are usually treated separately

- **Core graph query functionalities**
  - Path navigation (reachability) queries
  - Subgraph pattern queries
Can you reach film_1267 from film_2014?
Is there a book whose rating is $> 4.0$ associated with a film that was directed by Stanley Kubrick?
Reachability Queries

Think of Facebook graph and finding friends of friends.
This is computing the transitive closure

- Fully materialized: $O(n \times m)$ index time ($n$ vertices, $m$ edges), $O(1)$ query time
- BFS/DFS: $O(1)$ index time, $O(n + m)$ query time
This is computing the transitive closure

- Fully materialized: $O(n \times m)$ index time ($n$ vertices, $m$ edges), $O(1)$ query time
- BFS/DFS: $O(1)$ index time, $O(n + m)$ query time
Regular Path Queries (RPQ)
- RPQ = path query that defines desired paths using a regular expression
  ⇒ labels of a path form a word in the language specified by RPQ
- Generalization of reachability queries ⇒ reachability query = A RPQ
  that accepts all words

α-Rα – Relational Algebra extended with Transitive Closure

Finite-automata Based RPQ Evaluation
- Traversal guided by an FA: G+ [Cruz et al., 1987; Mendelzon and Wood, 1995]

Hybrid α-Rα & FA-based traversals: Waveguide [Yakovets et al., 2016]

\[ Q(p, f) \leftarrow \text{knows} \cdot \text{worksFor} \cdot \text{knows}^+ \]
Subgraph Matching

FILTER(?r > 4.0)

?b

movie:relatedBook

?m

movie:director

?d

name

rdfs:label

“Stanley Kubrick"

movie:director_name

?r

rev:rating

4.7

bm:books/0743424425

scam:hasOffer

bm:offers/0743424425amazonOffer

“The Shining”

movie:initial_release_date

“1980-05-23”

mdb:film/2014

refs:label

mdb:film/2685

mdb:director/8476

movie:director

“Spartacus”

mdb:film/424

mdb:actor/30013

movie:actor

“Jack Nicholson”

mdb:actor/29704

movie:actor

“The Passenger”

mdb:film/3418

refs:label

“United Kingdom”

gn:population

62348447

gn:name

movie:relatedBook

movie:actor

foaf:based_near

movie:actor

“The Last Tycoon”

mdb:film/1267

refs:label

mdb:director/8476

movie:director

“Stanley Kubrick”

mdb:film/2685

refs:label

“A Clockwork Orange”

mdb:director/8476

movie:actor

movie:actor

movie:actor

movie:actor
Subgraph Pattern Query Execution Approaches

- **Conjunctive Graph Queries (CQ)**
  - Set of edge predicates to define substructures of interest
  - Akin to joins in relational query processing

- **Worst-case Optimal Join Processing**
  - Leapfrog Triejoin [Veldhuizen, 2014]
  - EmptyHeaded [Aberger et al., 2017]
    - Generalized hypertree decompositions
  - Graphflow Hybrid [Mhedhbi and Salihoglu, 2019]
    - Adaptive, cost-based planning with WCO and binary joins

\[
Q(p, c) \leftarrow \text{knows}(p, x) \land \text{knows}(p, y) \land \text{worksFor}(x, c) \land \text{worksFor}(y, c)
\]
Subgraph Pattern Query Execution Approaches

- Conjunctive Graph Queries (CQ)
  - Set of edge predicates to define substructures of interest
  - Akin to joins in relational query processing

- Worst-case Optimal Join Processing
  - Leapfrog Triejoin [Veldhuizen, 2014]
  - EmptyHeaded [Aberger et al., 2017]
    - Generalized hypertree decompositions
  - Graphflow Hybrid [Mhedhbi and Salihoglu, 2019]
    - Adaptive, cost-based planning with WCO and binary joins

\[ Q(p, c) \leftarrow \text{knows}(p, x) \land \text{knows}(p, y) \land \text{worksFor}(x, c) \land \text{worksFor}(y, c) \]
What are Some Open Issues?

- Existing systems generally have performance issues
  - Generally involve joins of intermediate results, which may be quite large
  - There are not extensive performance studies (LDBC is a good start [Erling et al., 2015])
What are Some Open Issues?

- Existing systems generally have performance issues
- There is poor locality in graph workloads
  - Caching does not help much

![A commercial system; LDBC-1](image-url)

**LDBC Queries**
- Hit rate

<table>
<thead>
<tr>
<th></th>
<th>Q6</th>
<th>Q7</th>
<th>Q8</th>
<th>Q9</th>
<th>Q10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Native</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LRU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What are Some Open Issues?

- Existing systems generally have performance issues
- There is poor locality in graph workloads
  - Caching does not help much
  - Proper clustering of vertices and edges on pages may reduce page I/O

Query languages need attention
- Query processing and optimization
- Fuzzy querying over uncertain and probabilistic graphs

Too much focus on simple homogeneous graphs
⇒ multigraphs, heterogeneous graphs are important

Graph-aware Disk Layout - Locality Experiments

(Commercial system; LDBC-1)

![Graph-aware Disk Layout - Locality Experiments](image_url)

- Native
- Clustered
What are Some Open Issues?

- Existing systems generally have performance issues
- There is poor locality in graph workloads
  - Caching does not help much
  - Proper clustering of vertices and edges on pages may reduce page I/O
  - Native graph storage system design requires more work
- What should graph databases cache? (subgraphs, paths, vertices, query plans, or what)

Query languages need attention
- Query processing and optimization
- Fuzzy querying over uncertain and probabilistic graphs
  - [Yuan et al., 2011, 2012]

⇒ too much focus on simple homogeneous graphs

© M. Tamer Özsu
VLDB 2019

(2019/08/27) 37
What are Some Open Issues?

- Existing systems generally have performance issues
- There is poor locality in graph workloads
- Query languages need attention
  - Need to capture both graph topology and properties
    - Most current work is simplistic
    - Promising: Register automata-based execution for RPQ evaluation
  - Query semantics (and syntax) are still not clarified or standardized
    - Are the proposed languages complete? Proof?
    - How to determine a query is safe?
    - G-Core effort is important
What are Some Open Issues?

- Existing systems generally have performance issues
- There is poor locality in graph workloads
- Query languages need attention
- Query processing and optimization
  - What are the primary operators? Can we have a closed algebra? (see [Salihoglu and Widom, 2014; Mattson et al., 2013])
  - Advanced query plan generation issues
What are Some Open Issues?

- Existing systems generally have performance issues
- There is poor locality in graph workloads
- Query languages need attention
- Query processing and optimization
- Fuzzy querying over uncertain and probabilistic graphs [Yuan et al., 2011, 2012]
What are Some Open Issues?

- Existing systems generally have performance issues
- There is poor locality in graph workloads
- Query languages need attention
- Query processing and optimization
- Fuzzy querying over uncertain and probabilistic graphs [Yuan et al., 2011, 2012]
- Too much focus on simple homogeneous graphs $\Rightarrow$ multigraphs, heterogeneous graphs are important
  - Some work exists – on multigraphs:
    - Constraints on individual edges [Erling et al., 2015]
    - Constraints on a full path [Zhang and Özsu, 2019]
What are Some Open Issues?

- Existing systems generally have performance issues
- There is poor locality in graph workloads
- Query languages need attention
- Query processing and optimization
- Fuzzy querying over uncertain and probabilistic graphs [Yuan et al., 2011, 2012]
- Too much focus on simple homogeneous graphs, heterogeneous graphs are important
  - Some work exists – on multigraphs:
    - Constraints on individual edges [Erling et al., 2015]
    - Constraints on a full path [Zhang and Özsu, 2019]

Research session 18 on Thursday @ 11:00
What are Some Open Issues?

- Existing systems generally have performance issues
- There is poor locality in graph workloads
- Query languages need attention
- Query processing and optimization

![Most important ...](image.png)

1. Disk-based systems ⇒ storage system design needs much work & experimentation
2. Query languages/semantics are current bottleneck ⇒ optimization work would benefit
3. Non-trivial scale-out architectures and processing requires further study
Graph Analytics System Properties

- Property graph model

```
books_0743424425
  (rating, 4.7)
  (creator, StephenKing)

offers_0743424425
  (hasOffer,)

UnitedKingdom
  (geo_2635167)
  (name, "United Kingdom")
  (population, 62348447)

film_3418
  (label, "The Passenger")
  (wikipedia_article)

film_1267
  (label, "The Last Tycoon")

actor_29704
  (actor)
  (actor_name, "Jack Nicholson")

actor_30013
  (actor)
  (actor_name, "Shelley Duvall")

director_8476
  (director)
  (director_name, "Stanley Kubrick")

film_2685
  (label, "A Clockwork Orange")
  (director)

film_424
  (label, "Spartacus")
  (director)

film_2014
  (initial_release_date, "1980-05-23")
  (label, "The Shining")
  (music_contributor, music_contributor/4110)
  (language, iso639_3/eng)
  (usesScript, script/latn)

StephenKing
  (creator)

UnitedKingdom
  (geo_2635167)
  (name, "United Kingdom")
  (population, 62348447)

actor_29704
  (actor)
  (actor_name, "Jack Nicholson")

actor_30013
  (actor)
  (actor_name, "Shelley Duvall")

director_8476
  (director)
  (director_name, "Stanley Kubrick")

UnitedKingdom
  (geo_2635167)
  (name, "United Kingdom")
  (population, 62348447)
```

- Offline workloads
  - Each query accesses the entire graph – indexes may not help
  - Queries are iterative until a fix point is reached

- Examples
  - PageRank
  - Clustering
  - Connected components
  - Diameter finding
  - Graph colouring
  - All pairs shortest path
  - Graph pattern mining
  - Machine learning algorithms (Belief propagation, Gaussian non-negative matrix factorization)

- Almost all of the existing systems are scale-out
Graph Analytics System Properties

- Property graph model
- Offline workloads
  - Each query accesses the entire graph – indexes may not help
  - Queries are iterative until a fix point is reached
  - Examples
    - PageRank
    - Clustering
    - Connected components
    - Diameter finding
    - Graph colouring
    - All pairs shortest path
    - Graph pattern mining
    - Machine learning algorithms (Belief propagation, Gaussian non-negative matrix factorization)
Graph Analytics System Properties

- Property graph model
- Offline workloads
  - Each query accesses the entire graph – indexes may not help
  - Queries are iterative until a fix point is reached
  - Examples
    - PageRank
    - Clustering
    - Connected components
    - Diameter finding
    - Graph colouring
    - All pairs shortest path
    - Graph pattern mining
    - Machine learning algorithms (Belief propagation, Gaussian non-negative matrix factorization)

- Almost all of the existing systems are scale-out
Can MapReduce be Used for Graph Analytics?

Yes, but not a good idea

- Immutable data & computation is not guaranteed to be on the same machine in subsequent iterations
- High I/O cost due to repeated read/write to/from store between iterations

There are systems that try to address these concerns

- HaLoop [Bu et al., 2010, 2012]
- GraphX over Spark [Gonzalez et al., 2014]
Can MapReduce be Used for Graph Analytics?

Yes, but not a good idea

- Immutable data & computation is not guaranteed to be on the same machine in subsequent iterations
- High I/O cost due to repeated read/write to/from store between iterations
Can MapReduce be Used for Graph Analytics?

Yes, but not a good idea

▶ Immutable data & computation is not guaranteed to be on the same machine in subsequent iterations
▶ High I/O cost due to repeated read/write to/from store between iterations

There are systems that try to address these concerns

▶ HaLoop [Bu et al., 2010, 2012]
▶ GraphX over Spark [Gonzalez et al., 2014]
Classification of Graph Analytics Systems

- Programming model
- Computation model

Programming Model

- Vertex-centric
- Partition-centric
- Edge-centric

Computation Model

- Block Synchronous Parallel (BSP)
- Asynchronous
- Gather-Apply Scatter (GAS)
Vertex-centric

- Computation on a vertex is the focus
- "Think like a vertex"
- Vertex computation depends on its own state + states of its neighbors
- Compute(vertex v)
- GetValue(), WriteValue()
Programming Models

- **Vertex-centric**
- **Partition-centric (Block-centric)**
  - Computation on an entire partition is specified
  - “Think like a block” or “Think like a graph”
  - Aim is to reduce the communication cost among vertices
Programming Models

- Vertex-centric
- Partition-centric (Block-centric)
- Edge-centric
  - Computation is specified on each edge rather than on each vertex or block
  - Compute(edge e)
Computational Models

- Bulk Synchronous Parallel (BSP) [Valiant, 1990]

Each machine performs computation on its graph partition. At the end of each superstep, results are pushed to other workers.
Computational Models

- Bulk Synchronous Parallel (BSP) [Valiant, 1990]
- Asynchronous Parallel
  - No communication barriers
  - Uses the *most recent* values
  - Implemented via distributed locking
Computational Models

- Bulk Synchronous Parallel (BSP) [Valiant, 1990]
- Asynchronous Parallel
- Gather-Apply-Scatter (GAS)
  - Similar to BSP, but pull-based
  - Gather: pull state
  - Apply: Compute function
  - Scatter: Update state
  - Updates of states separated from scheduling
Classification of Graph Analytics Systems

- **Vertex-centric**
  - GAS
  - Asynchronous

- **Partition-centric**
  - BSP

- **Edge-centric**
  - BSP

**Programming Models**:
- **Bulk Synchronous Parallel (BSP)**
- **Gather-Apply Scatter (GAS)**
- **Asynchronous**

**Systems**:
- GiraphCU
- GraphLab
- Blogel
- X-Stream

© M. Tamer Özsu

VLDB 2019
Classification of Graph Analytics Systems

- Pregel [Malewicz et al., 2010], Apache Giraph, GPS [Salihoglu and Widom, 2013], Mizan [Khayyat et al., 2013], Trinity [Shao et al., 2013]

- Vertex-centric: GAS, Asynchronous
- Partition-centric: BSP
- Edge-centric: BSP

- Programming Model: Bulk Synchronous Parallel (BSP), Asynchronous, Gather-Apply Scatter (GAS), Vertex-centric, Partition-centric, Edge-centric
Classification of Graph Analytics Systems

- **Vertex-centric**
  - **GAS**
  - **GraphLab** [Low et al., 2010]
  - **Asynchronous**
  - **GiraphCU** [Han and Daudjee, 2015]

- **Partition-centric**
  - **BSP**
  - **Blogel** [Yan et al., 2014]
  - **X-Stream** [Roy et al., 2013]

- **Edge-centric**
  - **BSP**

**Programming Model**
- **Vertex-centric**
- **Partition-centric**
- **Edge-centric**

**Gather-Apply Scatter (GAS)**
- **Asynchronous**
- **Bulk Synchronous Parallel (BSP)**

**Systems**
- **Pregel** [Malewicz et al., 2010], Apache Giraph, GPS [Salihoglu and Widom, 2013], Mizan [Khayyat et al., 2013], Trinity [Shao et al., 2013]
Classification of Graph Analytics Systems

Computation Model

Bulk Synchronous Parallel (BSP)
- Vertex-centric
- Partition-centric
- Edge-centric

Asynchronous
- Vertex-centric

Gather-Apply Scatter (GAS)
- Vertex-centric

Programming Model

Giraph (CU)
- [Han and Daudjee, 2015]

GraphLab
- [Low et al., 2010]

Blogel
- [Yan et al., 2014]

X-Stream
- [Roy et al., 2013]

© M. Tamer Özsu
VLDB 2019
(2019/08/27) 44
OLAP Over Graphs

- **OLAP in RDBMS**
  - Usage: Data Warehousing + Business Intelligence
  - Model: Multidimensional cube
  - Operations: Roll-up, drill-down, and slice and dice

- Analytics that we discussed over graphs is much different
- Can we do OLAP-style analytics over graphs?
  - There is some work
    - Graph summarization [Tian et al., 2008]
    - Snapshot-based Aggregation [Chen et al., 2008]
    - Graph Cube [Zhao et al., 2011]
    - Pagrol [Wang et al., 2014]
    - Gagg Model [Maali et al., 2015]
Some Open Problems

- Current systems would have difficulty scaling to some large graphs
  - Graphs with billions of vertices, hundreds of billions of edges are becoming more common
  - Brain network is a trillion edge graph
  - Even the large graphs we play with are small
Some Open Problems

- Current systems would have difficulty scaling to some large graphs
  - Graphs with billions of vertices, hundreds of billions edges are becoming more common
  - Brain network is a trillion edge graph
  - Even the large graphs we play with are small

- Integration with data science workflows
  - Focus has been mostly on single computation
  - Analytics as part of a complete workflow: financial analysis, litigation analytics
  - Single algorithms → systems
Some Open Problems

- Current systems would have difficulty scaling to some large graphs
  - Graphs with billions of vertices, hundreds of billions edges are becoming more common
  - Brain network is a trillion edge graph
  - Even the large graphs we play with are small

- Integration with data science workflows
  - Focus has been mostly on single computation
  - Analytics as part of a complete workflow: financial analysis, litigation analytics
  - Single algorithms → systems

- ML workloads over graphs are interesting and requires more attention
Some Open Problems

- Current systems would have difficulty scaling to some large graphs
  - Graphs with billions of vertices, hundreds of billions edges are becoming more common
  - Brain network is a trillion edge graph
  - Even the large graphs we play with are small

- Integration with data science workflows

⚠️ Most important...

1. Are the types of systems we have been focusing on still relevant & reasonable?
2. Serious scaling ⇒ computation over HPC infrastructures might become important
3. Consider analytics as part of a full workflow
Dynamic Graphs

A graph sees updates over time and these updates can be both on topology and properties. Existing work predominantly focuses on topology updates. A graph is bounded and fully available to the algorithms. Computation approaches include batch computation of each snapshot, incremental computation, and general-purpose algorithms like differential dataflow [McSherry et al., 2013]. Specialized algorithms for specific workloads, such as subgraph matching [Ammar et al., 2018; Fan et al., 2011], shortest path [Nannicini and Liberti, 2008], and connected components [McColl et al., 2013], are also discussed.
Graphs see updates over time. Update can be both on topology and properties. Existing work predominantly focusing on topology updates. Graphs are bounded and fully available to the algorithms.

Computation approaches:
- Batch computation of each snapshot
- Incremental computation

- General purpose: Differential dataflow [McSherry et al., 2013]
- Specialized algorithms for specific workloads: E.g., subgraph matching [Ammar et al., 2018; Fan et al., 2011], shortest path [Nannicini and Liberti, 2008], connected components [McColl et al., 2013]
Dynamic Graphs

Graph sees updates over time

Update can be both on topology and properties

Existing work predominantly focusing on topology updates

Graph is bounded and fully available to the algorithms

Computation approaches

Batch computation of each snapshot

Incremental computation

General purpose: Differential dataflow

\cite{McSherry et al., 2013}

Specialized algorithms for specific workloads: E.g., subgraph matching

\cite{Ammar et al., 2018; Fan et al., 2011}, shortest path

\cite{Nannicini and Liberti, 2008}, connected components

\cite{McColl et al., 2013}
Dynamic Graphs

Graph sees updates over time

Update can be both on topology and properties

Existing work predominantly focusing on topology updates

Graph is bounded and fully available to the algorithms

Computation approaches

- Batch computation of each snapshot
- Incremental computation

General purpose: Differential dataflow

- [McSherry et al., 2013]

Specialized algorithms for specific workloads:
- E.g., subgraph matching
- [Ammar et al., 2018; Fan et al., 2011], shortest path
- [Nannicini and Liberti, 2008], connected components
- [McColl et al., 2013]
Dynamic Graphs

- Graph sees updates over time
  - Update can be both on topology and properties
  - Existing work predominantly focusing on topology updates
- Graph is bounded and fully available to the algorithms
- Computation approaches
  - Batch computation of each snapshot
  - Incremental computation
    - General purpose: Differential dataflow [McSherry et al., 2013]
    - Specialized algorithms for specific workloads: E.g., subgraph matching [Ammar et al., 2018; Fan et al., 2011], shortest path [Nannicini and Liberti, 2008], connected components [McColl et al., 2013]
Differential Dataflow

- Applies to any data flow computation
- Does not depend on the semantics of the computation

Changes arrive, each operator is asked if there are any changes. If there are, push the changes to the next operator. If not, stop. This can save work. Iterative workloads, e.g., graph analytics, changes come both from input and from previous iteration. Timestamped set of changes uses partial order to optimize. "Generalized incremental dataflow maintenance".

\[ \text{Input A} \quad \text{Input B} \]

\[ \text{op}_1 \quad \text{op}_2 \quad \text{op}_3 \]

\[ \ldots \]

\[ \text{filter} \]

\[ \text{Output} \]
Differential Dataflow

- Applies to any data flow computation
- Does not depend on the semantics of the computation
Differential Dataflow

- Applies to any data flow computation
- Does not depend on the semantics of the computation
- When changes arrive, each operator is asked if there are any changes
  - If there are, push the changes to next operator
  - If not, stop ⇒ early stop saves work

![Diagram of Differential Dataflow]

© M. Tamer Özsu
VLDB 2019 (2019/08/27) 49
Differential Dataflow

- Applies to any data flow computation
- Does not depend on the semantics of the computation
- When changes arrive, each operator is asked if there are any changes
  - If there are, push the changes to next operator
  - If not, stop ⇒ early stop saves work
- Iterative workloads, e.g., graph analytics
  - Changes come both from input and from previous iteration
  - Timestamped set of changes
  - Uses partial order to optimize
  - “Generalized incremental dataflow maintenance”
Streaming Graphs

\[ t_0 \quad t_1 \quad t_2 \quad t_3 \quad t_4 \quad t_5 \quad t_6 \quad t_7 \quad t_8 \quad t_9 \quad t_{10} \quad t_{11} \quad t_{12} \quad t_{13} \]

Time
Streaming Graphs

Time

A

B

A

B

t_1
Streaming Graphs

Diagram showing nodes A and B over time:
- Node A at time $t_1$
Streaming Graphs

- Graph at time $t_0$: A and B
- Graph at time $t_1$: A and B
- Graph at time $t_4$: A and C
- Graph at time $t_5$: B and C

Time progression from $t_0$ to $t_5$: $t_0$, $t_1$, $t_4$, $t_5$
Streaming Graphs

$t_0$  $t_1$  $t_2$  $t_3$  $t_4$  $t_5$  $t_6$  $t_7$  $t_8$  $t_9$  $t_{10}$  $t_{11}$  $t_{12}$  $t_{13}$  Time

$t_1$  $t_4$  $t_5$
Streaming Graphs

$t_0$ $t_1$ $t_2$ $t_3$ $t_4$ $t_5$ $t_6$ $t_7$ $t_8$ $t_9$ $t_{10}$ $t_{11}$ $t_{12}$ $t_{13}$

$t_1$ $t_4$ $t_5$ $t_7$
Streaming Graphs

A
→ B
→ C
→ D
→ E
→ F

A
→ B
→ C
→ D

B
→ C
→ D
→ E
→ F

© M. Tamer Özsu
VLDB 2019
(2019/08/27)
Streaming Graphs

Time $t_0$ $t_1$ $t_2$ $t_3$ $t_4$ $t_5$ $t_6$ $t_7$ $t_8$ $t_9$ $t_{10}$ $t_{11}$ $t_{12}$ $t_{13}$

$A$ $B$ $C$ $D$ $E$ $F$

$t_1$ $t_4$ $t_5$ $t_7$ $t_9$ $t_{12}$

© M. Tamer Özsu
VLDB 2019
Streaming Graphs

- Combines two difficult problems: streaming + graphs
- Unbounded ⇒ don’t see entire graph
- Streaming rates can be very high
- Computational models
  - Continuous: for simple transactional operations
Streaming Graphs

- Combines two difficult problems: streaming + graphs
- Unbounded ⇒ don’t see entire graph
- Streaming rates can be very high
- Computational models
  - Continuous: for simple transactional operations
  - Windowed: for more complex queries
Continuous Computation

Query: Vertices reachable from vertex A

<table>
<thead>
<tr>
<th>Time</th>
<th>Incoming edge</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>$\langle A,B \rangle$</td>
<td>${B}$</td>
</tr>
<tr>
<td>$t_2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_3$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_4$</td>
<td>$\langle B,C \rangle$</td>
<td>${B,C}$</td>
</tr>
<tr>
<td>$t_5$</td>
<td>$\langle A,D \rangle$, $\langle D,C \rangle$</td>
<td>${B,C,D}$</td>
</tr>
<tr>
<td>$t_6$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_7$</td>
<td>$\langle C,F \rangle$, $\langle D,F \rangle$</td>
<td>${B,C,D,F}$</td>
</tr>
<tr>
<td>$t_8$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_9$</td>
<td>$\langle D,E \rangle$, $\langle A,E \rangle$, $\langle B,E \rangle$, $\langle E,F \rangle$</td>
<td>${B,C,D,F,E}$</td>
</tr>
<tr>
<td>$t_{10}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Windowed Computation

Query: Vertices reachable from vertex A
Window size=5

<table>
<thead>
<tr>
<th>Time</th>
<th>Incoming edge</th>
<th>Expired edges</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>$\langle A, B \rangle$</td>
<td></td>
<td>${B}$</td>
</tr>
<tr>
<td>$t_2$</td>
<td>$\langle B, C \rangle$</td>
<td></td>
<td>${B, C}$</td>
</tr>
<tr>
<td>$t_3$</td>
<td>$\langle A, D \rangle, \langle D, C \rangle$</td>
<td></td>
<td>${B, C, D}$</td>
</tr>
<tr>
<td>$t_4$</td>
<td>$\langle A, B \rangle$</td>
<td></td>
<td>${B, C, D}$</td>
</tr>
<tr>
<td>$t_5$</td>
<td>$\langle C, F \rangle, \langle D, F \rangle$</td>
<td>$\langle A, B \rangle$</td>
<td>${C, D, F}$</td>
</tr>
<tr>
<td>$t_6$</td>
<td>$\langle D, E \rangle, \langle A, E \rangle, \langle B, E \rangle, \langle E, F \rangle$</td>
<td>$\langle B, C \rangle$</td>
<td>${C, D, F, E}$</td>
</tr>
<tr>
<td>$t_7$</td>
<td>$\langle A, D \rangle, \langle D, C \rangle$</td>
<td>$\langle A, D \rangle, \langle D, C \rangle$</td>
<td>${C, D, F, E}$</td>
</tr>
</tbody>
</table>
Graph Stream Algorithms

- Unboundedness brings up space issues
  - Continuous computation (pure streams) model requires linear space $\Rightarrow$ unrealistic
    - Many graph problems are not solvable (see [McGregor, 2014] for a survey)
  - Semi-streaming model $\Rightarrow$ sublinear space [Feigenbaum et al., 2005]
    - Sufficient to store vertices but not edges (typically $|V| \ll |E|$)
    - Approximation for many graph algorithms, spanners [Elkin, 2011], connectivity [Feigenbaum et al., 2005], matching [Kapralov, 2013], etc.
Querying Graph Streams

- Remember graph query functionalities
  - Subgraph matching queries & reachability (path) queries
  - Doing these in the streaming context
  - This is querying beyond simple transactional operations on an incoming edge
    - Edge represents a user purchasing an item → do some operation
    - Edge represents events in news → send an alert

- Subgraph pattern matching under stream of updates
  - Windowed join processing
  - Graphflow [Kankanamge et al., 2017], TurboFlux [Kim et al., 2018]
    - These are not designed to deal with unboundedness of the data graph

- Path queries under stream of updates
  - Windowed RPQ evaluation on unbounded streams
Analytics on Graph Streams

- Many use cases
  - Recommender systems
  - Fraud detection [Qiu et al., 2018]
  - ...

- Existing relevant work
  - Snapshot-based systems
    - Aspen [Dhulipala et al., 2019], STINGER [Ediger et al., 2012]
    - Consistent graph views across updates
  - Snapshot + Incremental Computations
    - Kineograph [Cheng et al., 2012], GraPu [Sheng et al., 2018],
      GraphIn [Sengupta et al., 2016], GraphBolt [Mariappan and Vora,
      2019]
    - Identify and re-process subgraphs that are effected by updates
  - Designed to handle high velocity updates
  - Cannot handle unbounded streams
  - Similar to dynamic graph processing solutions
We can do more with dynamic graphs, but efficient systems that incorporate novel techniques are needed.

Unboundedness in streams raises real challenges.

Most graph problems are unbounded under edge insert/delete.

The entire field is pretty much open!...
So, what is the big story?...
Reorient research...

1. A lot of the research has been algorithmic; time to shift focus to systems-aspects
2. Storage system architectures & structures
3. Indexing graph data?
4. Query primitives, processing methodology & optimization techniques
There are few independent large-scale performance studies (e.g., [Rusu and Huang, 2019; Han et al., 2014; Ammar and Özsu, 2018])

Reasonable benchmarks are emerging: LDBC for graph DBMS [Erling et al., 2015], WatDiv for RDF [Aluç et al., 2014], Graph500 for very large graphs

These are application benchmarks; microbenchmarks for system testing are needed
Focus on dynamic & streaming graphs...

1. We paid enough attention to static graphs; many real graphs are not static & many real applications require real-time answers
2. Dynamic $\neq$ streaming
3. Alert: this area is tough and you are not likely to write as many papers
Common DBMS for RDF & property graphs?

1. They both deal with online workloads and focus on querying
2. SPARQL only deals with subgraph queries ⇒ how to efficiently do path queries?
3. SPARQL semantics is graph homomorphism; subgraph queries over property graphs use graph isomorphism
4. Some discussion has started: W3C Workshop on Web Standardization for Graph Data: Creating Bridges: RDF, Property Graph and SQL
Looking for graph HTAP systems...

1. There are use cases and demand from users/industry
2. We need to decide what type of analytics we are considering: OLAP or offline workloads
3. There is work: TigerGraph, Quegel [Yan et al., 2016]
Some Issues That I Did Not Talk About

Graphs in AI/ML

1. Graphs in ML models
2. ML for graph analytics
Some Issues That I Did Not Talk About

Hardware support for graph processing

1. Quite a bit of work in using GPUs for acceleration
2. Mostly focus on managing GPU restrictions
3. Some work on using FPGAs
4. Worthwhile to consider a unified architecture: CPU+GPU+FPGA
5. Use of NVM for both in-memory and on-disk graph systems
Some Issues That I Did Not Talk About

Security & privacy issues

1. What is appropriate security granularity? Can you get multilevel security as in relational systems?
2. Anonymization of graphs (especially dynamic graphs) is difficult
Some Issues That I Did Not Talk About

Graphs in related/other fields

1. Network analysis: E.g., “Networks, Crowds, and Markets”, “Information and Influence Propagation in Social Networks”
2. Biological networks
3. Neuroscience: E.g., “Networks of the Brain”
4. …
Thank you!

To the DSG group

... and 40+ grad students and post-docs

To my collaborators ...

To the supporters


http://doi.acm.org/10.1145/2567948.2577302.

https://www.usenix.org/conference/osdi14/technical-sessions/presentation/gonzalez.


References XVII


