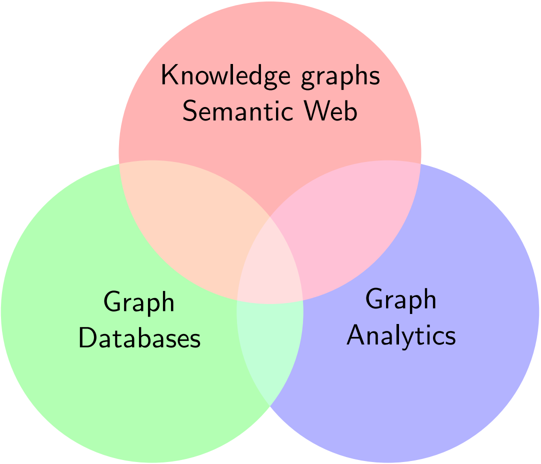
Graph Processing: A Panoramic View and Some Open Problems

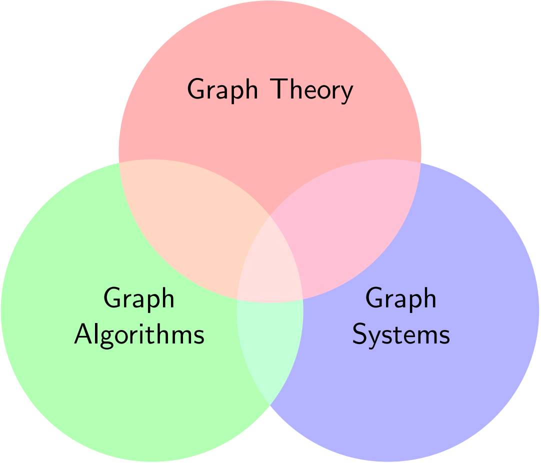
M. Tamer Özsu

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

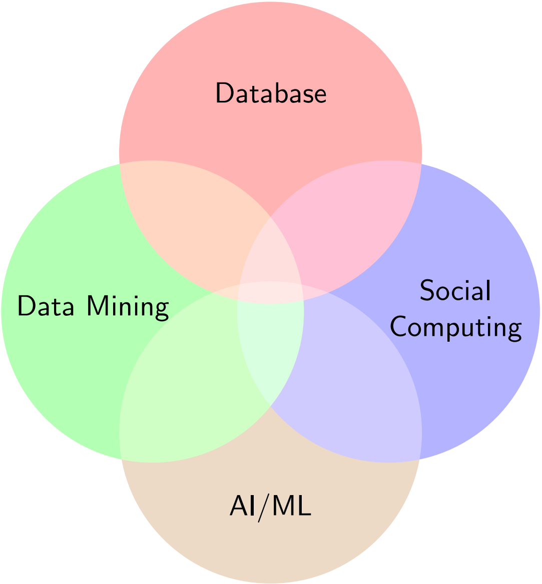
Graph Research is Dispersed



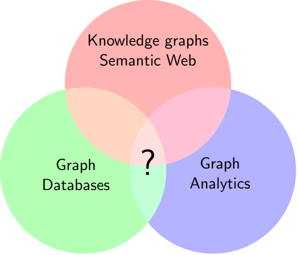
Graph Research is Dispersed

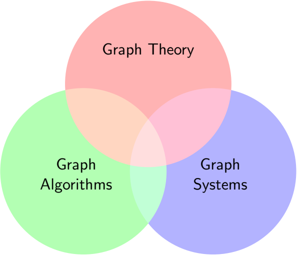


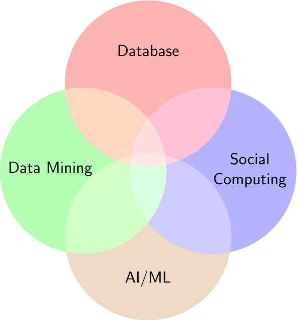
Graph Research is Dispersed



Graph Research is Dispersed



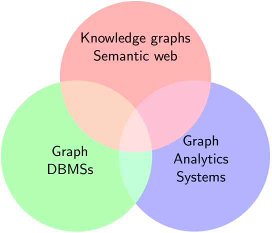




My objectives

Three things...

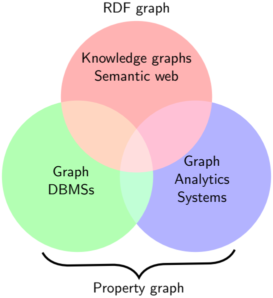
1. Discuss a way to coherently position work in the various communities;
2. A tour across different communities to provide a panoramic view of the research;
3. Highlight some problems that interest me!



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!



Freud's Recommendation for a Good Talk...

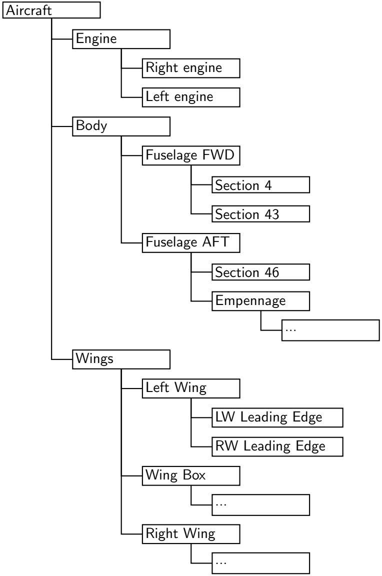
By Way of Moshe Vardi...

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,

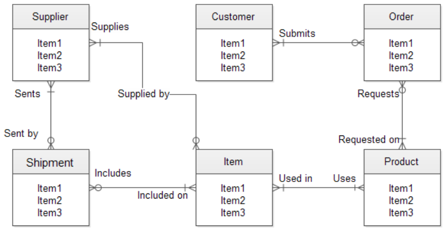
In the beginning...

... 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...

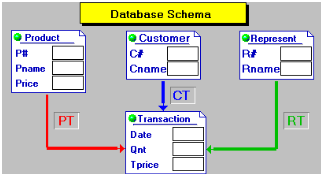


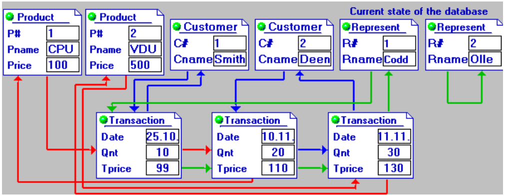
... and IDS

- 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

Network DBMSs



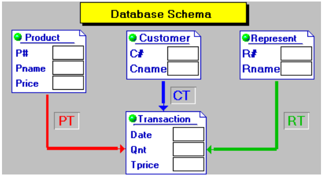


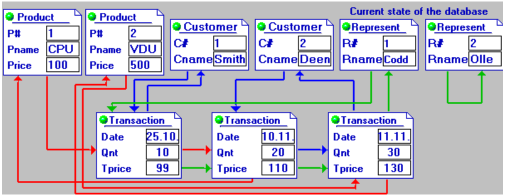
CODASYL Language

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

Source: Network (CODASYL) Data Model, https://coronet.iicm.tugraz.at/is/scripts/lesson03.pdf

Network DBMSs





CODASYL Language

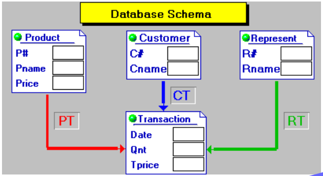
- FIND with key
- Navigate within the set, within elements of the same record type, etc

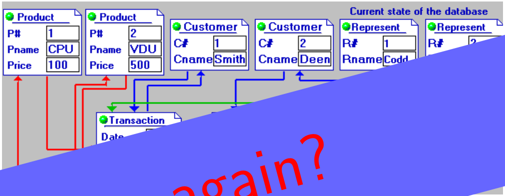
Network models were also used in

- Object DBMSs
- XML

Source: Network (CODASYL) Data Model, https://coronet.iicm.tugraz.at/is/scripts/lesson03.pdf

Network DBMSs





Déjà vu all over again?

CODA...

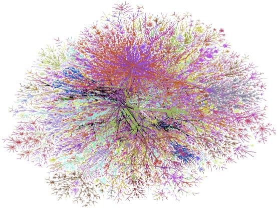
...the set, within elements of the same ...type, etc

Network models were also used in

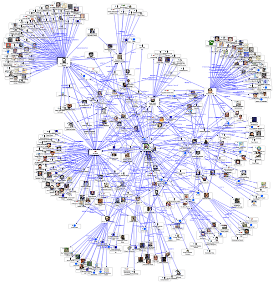
- Object DBMSs
- XML

Source: Network (CODASYL) Data Model, https://coronet.iicm.tugraz.at/is/scripts/lesson03.pdf

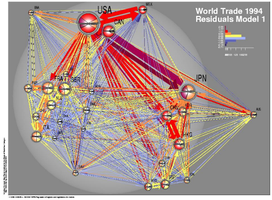
Modern graphs are different and diverse



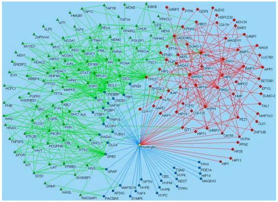
Internet



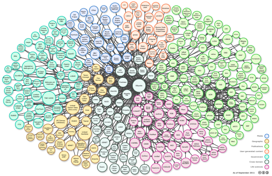
Social networks



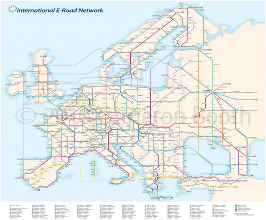
Trade volumes and connections



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)

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

1. Graphs are indeed everywhere!

Major Findings

[Sahu et al., 2017, 2019]

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

Entity	#Part.
Humans (e.g., customers, friends)	44
Non-human entities (e.g., web, products, files)	61
RDF	23
Scientific (e.g., proteins, molecules, bonds)	15

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?

Entity	#Part.
Humans (e.g., customers, friends)	44
Non-human entities (e.g., web, products, files)	61
RDF	23
Scientific (e.g., proteins, molecules, bonds)	15

Entity	#Part.
Web	4
Bus. & Financial	8
Product	13
Geo	7
Infrastructure	9
Digital	5
Knowledge	11

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?

Entity	#Part.
Humans (e.g., customers, friends)	44
Non-human entities (e.g., web, products, files)	61
RDF	23
Scientific (e.g., proteins, molecules, bonds)	15

Entity	#Part.	#Papers
Web	4	30
Bus. & Financial	8	8
Product	13	2
Geo	7	11
Infrastructure	9	2
Digital	5	0
Knowledge	11	3

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!

$\|E\|$	#Part.
<10K	23
10K–100K	22
100K–1M	13
1M–10M	9
10M–100M	21
100M–1B	21
>1B	20

#Bytes	#Part.
<10MB	23
10MB–100MB	22
100MB–1GB	13
1GB–10GB	9
10GB–100GB	21
100GB–1TB	21
>1TB	20

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

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

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
4. Scalability is the most pressing challenge!
   - Followed by visualization & query languages
5. RDBMS still play an important role!

More Info – Please read the papers

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:
Siddhartha Sahu, Amine Mhedhbi, Semih Salihoglu, Jimmy Lin, and M. Tamer Özsu. The Ubiquity of Large Graphs and Surprising Challenges of Graph Processing. *PVLDB*, 11(4): xxxx-yyyy, 2017.
DOI: https://doi.org/10.1145/3164135.3164139

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 [3,8,14,35,48,53], RDF engines [38,64,67], linear algebra software [6,46], visualization software [13,16], query languages [28, 52,55], 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 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:

- *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 enterprise 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.
- *Ubiquity of Very Large Graphs:* 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 refutes the sometimes heard assumption that large graphs are a problem for only a few large organizations such as Google, Facebook, and Twitter.
- *Challenge of 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.
- *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.
- *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 iden-

Proceedings of the VLDB Endowment, Vol. 11, No. 4
Copyright 2017 VLDB Endowment 2150-8097/17/12... $ 10.00.
DOI: https://doi.org/10.1145/3164135.3164139

The VLDB Journal
https://doi.org/10.1007/s00778-019-00548-x

SPECIAL ISSUE PAPER

Check for updates

The ubiquity of large graphs and surprising challenges of graph processing: extended survey

Siddhartha Sahu[1] · Amine Mhedhbi[1] · Semih Salihoglu[1] · Jimmy Lin[1] · M. Tamer Özsu[1]

Received: 21 January 2019 / Revised: 9 May 2019 / Accepted: 13 June 2019

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 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 [13,20,26,49,65,73,90], RDF engines [52,96], 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 in 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?

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s00778-019-00548-x) contains supplementary material, which is available to authorized users.

Siddhartha Sahu
s3sahu@uwaterloo.ca

Amine Mhedhbi
amine.mhedhbi@uwaterloo.ca

Semih Salihoglu
semih.salihoglu@uwaterloo.ca

Jimmy Lin
jimmylin@uwaterloo.ca

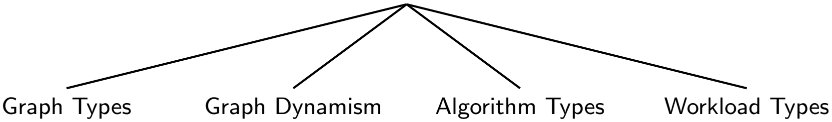
M. Tamer Özsu
tamer.ozsu@uwaterloo.ca

[1] University of Waterloo, Waterloo, Canada

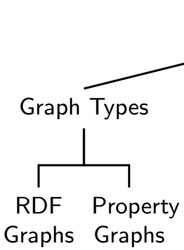
Published online: 29 June 2019

Springer

How I Think of This Domain!...



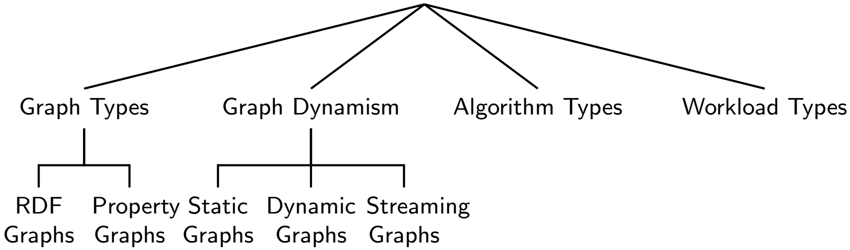
How I Think of This Domain!...



How I Think of This Domain!...

- Graph Types
  - RDF Graphs
  - Property Graphs
  - Graph Characteristics (Not complete)
    - Degree distribution; max degree
    - Diameter
    - Global/local density
    - Connectedness
    - Directed/undirected
    - Weighted/unweighted
    - Homogeneous/heterogeneous
    - Simple/multi/hyper
- Graph Dynamism
- Algorithm Types
- Workload Types

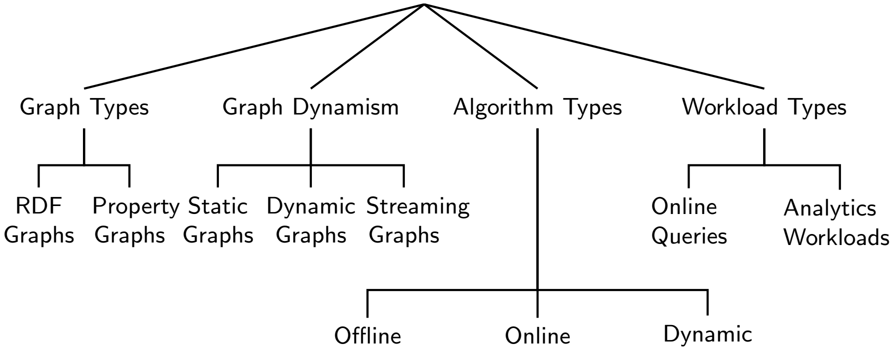
How I Think of This Domain!...



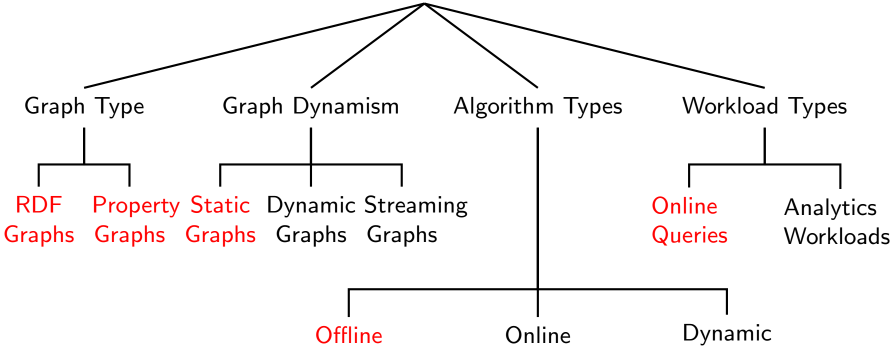
How I Think of This Domain!...

- Graph Types
  - RDF Graphs
  - Property Graphs
- Graph Dynamism
  - Static Graphs
  - Dynamic Graphs
  - Streaming Graphs
- Algorithm Types
- Workload Types
  - Online Queries
  - Analytics Workloads

How I Think of This Domain!...

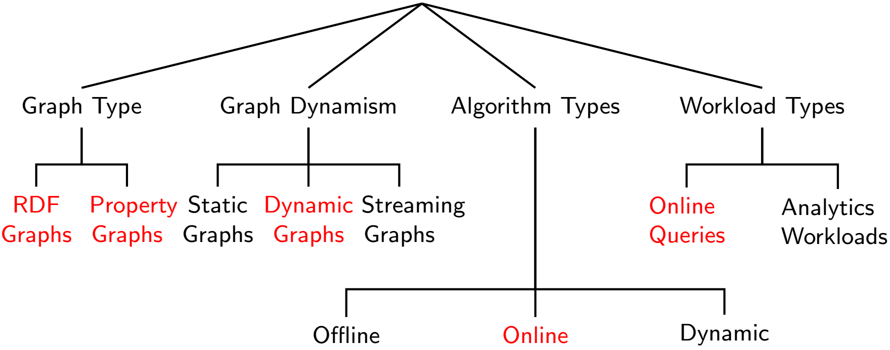


Example Design Points



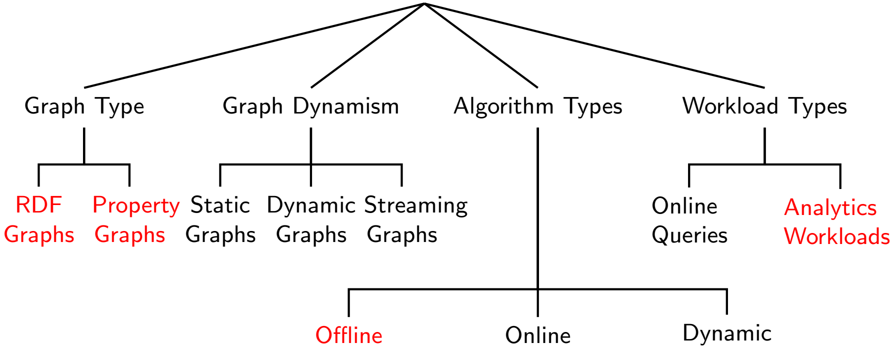
Compute the query result over the graph as it exists.

Example Design Points



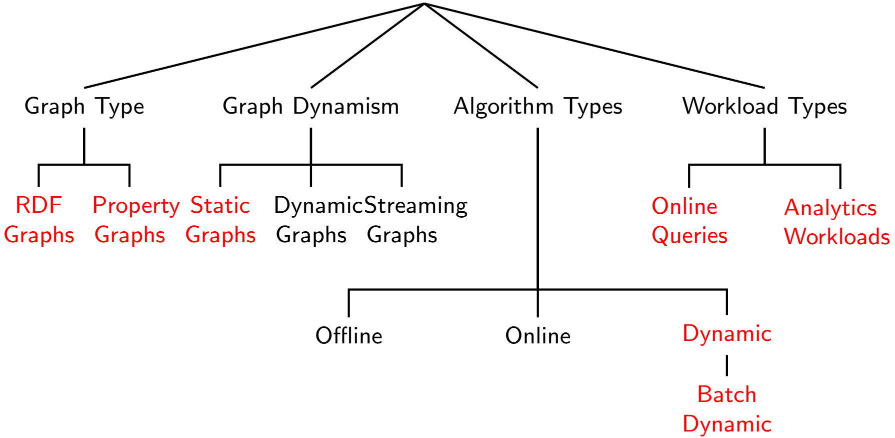
Compute the query result over the graph incrementally.

Example Design Points



Perform the analytic computation from scratch on each snapshot.

Example Design Points – Not all alternatives make sense



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 $\Rightarrow$ 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

Scale-up or Scale-out?

Scale Up or Scale Out for Graph Processing?

Jimmy Lin
University of Waterloo

This column explores a simple question: scale up or scale out for graph processing? Should we simply throw "beefier" individual multi-core, large-memory machines at graph processing tasks and focus on developing more efficient multi-threaded algorithms, or are investments in distributed graph processing frameworks and accompanying algorithms worthwhile? For rhetorical convenience, I adopt customary definitions, referring to the former as scale up and the latter as scale out. Under what circumstances should we prefer one approach over the other?

Whether we should scale up or out for graph processing is a consequential question from two perspectives: For big data practitioners, it would be desirable to develop a set of best practices that provide guidance to organizations building and deploying graph-processing capabilities. For big data researchers, these best practices translate into priorities for future work and provide a roadmap prioritizing real-world pain points.

tl;dr – I advocate scale-up solutions for graph processing as *the first thing to try*, since they are much simpler to design, implement, deploy, and maintain. If you really need distributed scale-out solutions, it means that your organization has become immensely successful. Not just "successful" but on a growth trajectory that is outpacing Moore's Law (it'll become clear what this means below). This is unlikely to be the case for most organizations, and even if you were fortunate enough to experience such explosive growth, success would likely bring commensurate resources to throw at the problem. So as long as you leave yourself enough headroom, it should be possible to build a scale-out graph processing solution just in time. The alternative is sunk costs in distributed graph processing infrastructure, which comes with huge parallelization overheads, anticipating a problem that never arrives.

It makes sense to begin by more carefully describing the scope of the problem: the type of graph processing I am referring to involves analytical queries to extract insights or to power data products. An example of the former might be clustering a large social network to infer latent communities of interest. An example of the latter might be building a graph-based recommendation system. Typically, these queries require traversing large portions of the graph where throughput, not latency, is the more important performance consideration.

Gartner defines "operational" databases as "relational and non-relational DBMS products suitable for a broad range of enterprise-level transactional applications, and DBMS products supporting interactions and observations as alternative types of transactions." I am specifically *not*

IEEE Internet Computing
May/June 2018
72
Published by the IEEE Computer Society
1089-7801/18/$33.00 ©2018 IEEE

Response to "Scale Up or Scale Out for Graph Processing"

Semih Salihoglu
University of Waterloo

M. Tamer Özsu
University of Waterloo

Editor:
Jimmy Lin
jimmylin@umd.edu

In this article, the authors provide their views on whether organizations should scale up or scale out their graph computations. This question was explored in a previous installment of this column by Jimmy Lin, where he made a case for scale-up through several examples. In response, the authors discuss three cases for scale-out.

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.[1] 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.

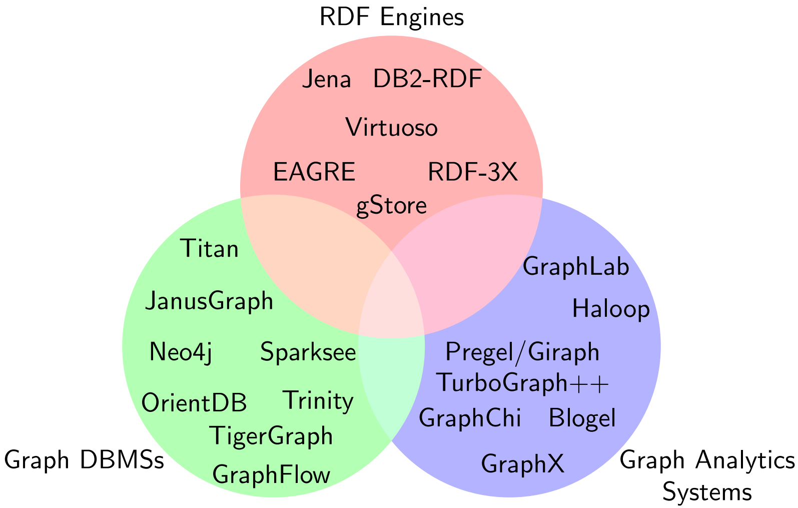
It is difficult to disagree with the central message of Jimmy's article: For many organizations that have large-scale graphs and want to run analytical computations, using a multicore single machine with a lot of RAM is a better option than a distributed cluster because single-machine software, compared 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.

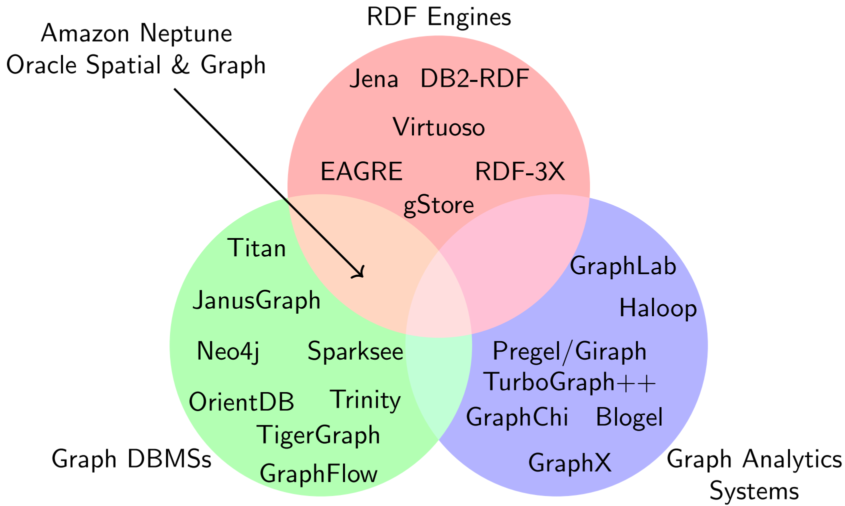
IEEE Internet Computing
September/October 2018
18
Published by the IEEE Computer Society
1089-7801/18/$33.00 ©2018 IEEE

Graph Systems



There are a number of others!...

Graph Systems



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/

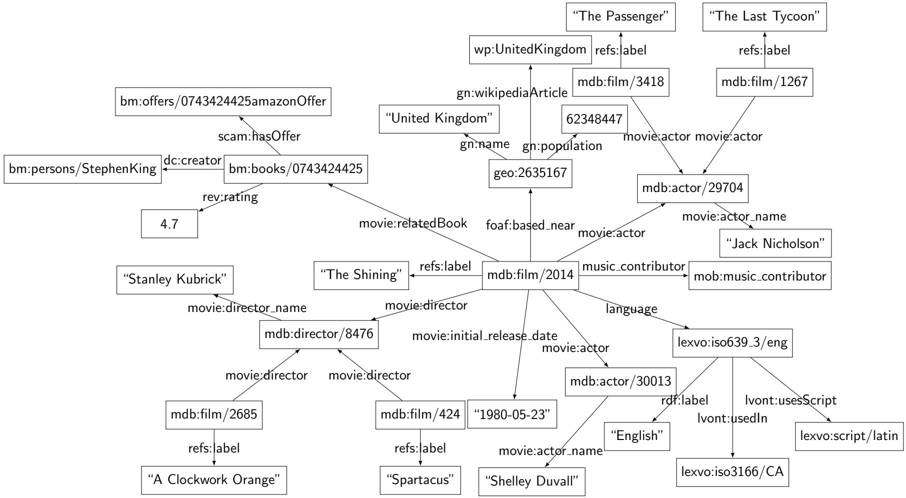
URI

Subject	Predicate	Object
mdb: film/2014	rdfs:label	"The Shining"
mdb:film/2014	movie:initial_release_date	"1980-05-23"'
mdb:film/2014	movie:director	mdb:director/8476
mdb:film/2014	movie:actor	mdb:actor/29704
mdb:film/2014	movie:actor	mdb: actor/30013
mdb:film/2014	movie:music_contributor	mdb: music_contributor/4110
mdb:film/2014	foaf:based_near	geo:2635167
mdb:film/2014	movie:relatedBook	bm:0743424425
mdb:film/2014	movie:language	lexvo:iso639-3/eng
mdb:director/8476	movie:director_name	"Stanley Kubrick"
mdb:film/2685	movie:director	mdb:director/8476
mdb:film/2685	rdfs:label	"A Clockwork Orange"
mdb:film/424	movie:director	mdb:director/8476
mdb:film/424	rdfs:label	"Spartacus"
mdb:actor/29704	movie:actor_name	"Jack Nicholson"
mdb:film/1267	movie:actor	mdb:actor/29704
mdb:film/1267	rdfs:label	"The Last Tycoon"
mdb:film/3418	movie:actor	mdb:actor/29704
mdb:film/3418	rdfs:label	"The Passenger"
geo:2635167	gn:name	"United Kingdom"
geo:2635167	gn:population	62348447
geo:2635167	gn:wikipediaArticle	wp:United_Kingdom
bm:books/0743424425	dc:creator	bm:persons/Stephen+King
bm:books/0743424425	rev:rating	4.7
bm:books/0743424425	scom:hasOffer	bm:offers/0743424425amazonOffer
lexvo:iso639-3/eng	rdfs:label	"English"
lexvo:iso639-3/eng	lvont:usedIn	lexvo:iso3166/CA
lexvo:iso639-3/eng	lvont:usesScript	lexvo:script/Latn

Literal

URI

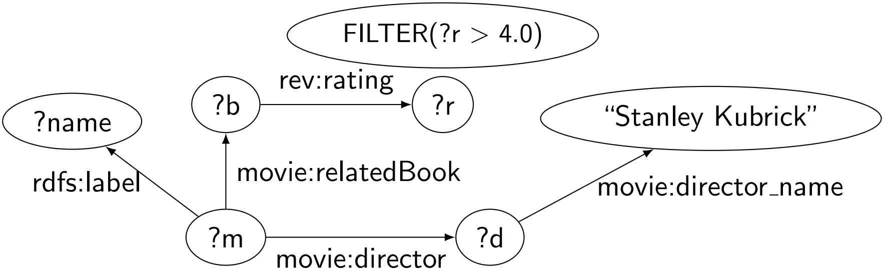
RDF Graph



"The Passenger"
"The Last Tycoon"
wp:UnitedKingdom
refs:label
refs:label
mdb:film/3418
mdb:film/1267
gn:wikipediaArticle
bm:offers/0743424425amazonOffer
"United Kingdom"
62348447
movie:actor
movie:actor
scam:hasOffer
gn:name
gn:population
dc:creator
bm:persons/StephenKing
bm:books/0743424425
geo:2635167
mdb:actor/29704
rev:rating
4.7
movie:relatedBook
foaf:based_near
movie:actor
movie:actor_name
"Jack Nicholson"
"Stanley Kubrick"
"The Shining"
refs:label
mdb:film/2014
music_contributor
mob:music_contributor
movie:director_name
movie:director
language
mdb:director/8476
movie:initial_release_date
movie:actor
lexvo:iso639_3/eng
movie:director
movie:director
mdb:actor/30013
rdf:label
lvont:usesScript
mdb:film/2685
mdb:film/424
"1980-05-23"
lvont:usedIn
refs:label
refs:label
movie:actor_name
"English"
lexvo:script/latin
lexvo:iso3166/CA
"A Clockwork Orange"
"Spartacus"
"Shelley Duvall"

SPARQL Queries

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



Direct Relational Mapping

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".
  ?m movie:relatedBook ?b. ?b rev:rating ?r.
  FILTER(?r > 4.0)
}
```

Subject	Property	Object
mdb:film/2014	rdfs:label	"The Shining"
mdb:film/2014	movie:initial_release_date	"1980-05-23"
mdb:film/2014	movie:director	mdb:director/8476
mdb:film/2014	movie:actor	mdb:actor/29704
mdb:film/2014	movie:actor	mdb: actor/30013
mdb:film/2014	movie:music_contributor	mdb: music_contributor/4110
mdb:film/2014	foaf:based_near	geo:2635167
mdb:film/2014	movie:relatedBook	bm:0743424425
mdb:film/2014	movie:language	lexvo:iso639-3/eng
mdb:director/8476	movie:director_name	"Stanley Kubrick"
mdb:film/2685	movie:director	mdb:director/8476
mdb:film/2685	rdfs:label	"A Clockwork Orange"
mdb:film/424	movie:director	mdb:director/8476
mdb:film/424	rdfs:label	"Spartacus"
mdb:actor/29704	movie:actor_name	"Jack Nicholson"
mdb:film/1267	movie:actor	mdb:actor/29704
mdb:film/1267	rdfs:label	"The Last Tycoon"
mdb:film/3418	movie:actor	mdb:actor/29704
mdb:film/3418	rdfs:label	"The Passenger"
geo:2635167	gn:name	"United Kingdom"
geo:2635167	gn:population	62348447
geo:2635167	gn:wikipediaArticle	wp:United_Kingdom
bm:books/0743424425	dc:creator	bm:persons/Stephen+King
bm:books/0743424425	rev:rating	4.7
bm:books/0743424425	scom:hasOffer	bm:offers/0743424425amazonOffer
lexvo:iso639-3/eng	rdfs:label	"English"
lexvo:iso639-3/eng	lvont:usedIn	lexvo:iso3166/CA
lexvo:iso639-3/eng	lvont:usesScript	lexvo:script/Latn

```
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 {
  ?m rdfs:label ?name. ?m movie:directo
  ?d movie:director_name "Stanley Kub
  ?m movie:relatedBook ?b. ?b rev:
  FILTER(?r > 4.0)
}
```

Subject	Property	Object
mdb:film/2014	rdfs:label	"The Shining"
mdb:film/2014	movie:initial_release_date	"1980-05-23"
mdb:film/2014	movie:director	mdb:director/8476
mdb:film/2014	movie:actor	mdb:actor/29704
mdb:film/2014	movie:actor	mdb: actor/30013
mdb:film/2014	movie:music_contributor	mdb: music_contributor/4110
mdb:film/2014	foaf:based_near	geo:2635167
mdb:film/2014	movie:relatedBook	bm:0743424425
mdb:film/2014	movie:language	lexvo:iso639-3/eng
mdb:director/8476	movie:director_name	"Stanley Kubrick"
mdb:film/2685	movie:director	mdb:director/8476
mdb:film/2685	rdfs:label	"A Clockwork Orange"
mdb:film/424	movie:director	mdb:director/8476
mdb:film/424	rdfs:label	"Spartacus"
mdb:actor/29704	movie:actor_name	"Jack Nicholson"
mdb:film/1267	movie:actor	mdb:actor/29704
mdb:film/1267	rdfs:label	"The Last Tycoon"
mdb:film/3418	movie:actor	mdb:actor/29704
mdb:film/3418	rdfs:label	"The Passenger"
geo:2635167	gn:name	"United Kingdom"
geo:2635167	gn:population	62348447
geo:2635167	gn:wikipediaArticle	wp:United_Kingdom
bm:books/0743424425	dc:creator	bm:persons/Stephen+King
bm:books/0743424425	rev:rating	4.7
bm:books/0743424425	scom:hasOffer	bm:offers/0743424425amazonOffer
lexvo:iso639-3/eng	rdfs:label	"English"
lexvo:iso639-3/eng	lvont:usedIn	lexvo:iso3166/CA
lexvo:iso639-3/eng	lvont:usesScript	lexvo:script/Latn

Easy to implement but too many self-joins!

```
F                                     T3,
  T          ,              5
WHERE T1.p="rdfs:l
AND T2.p="movie:rela     ook"
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"
```

Optimizations to Tabular Representation

Objectives

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

Optimizations to Tabular Representation

Objectives

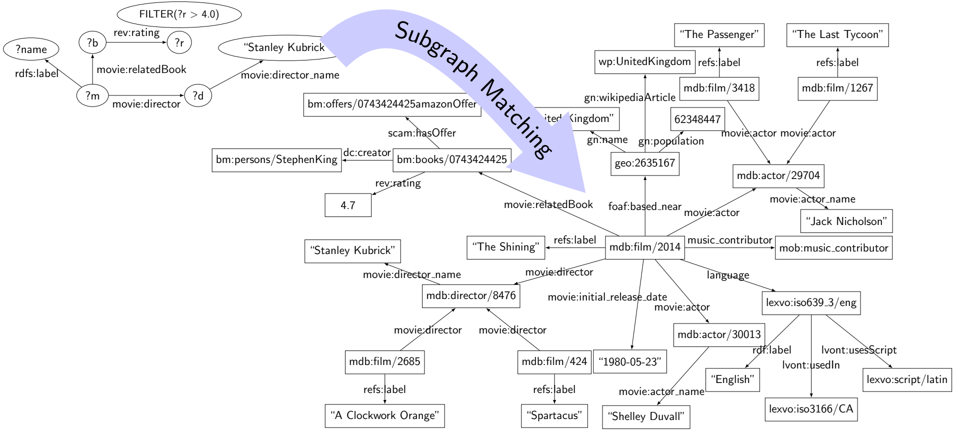
1. Eliminate/Reduce the number of self-joins
2. Use merge-join

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 [Zou and Özsu, 2017]

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



Graph-based Approach

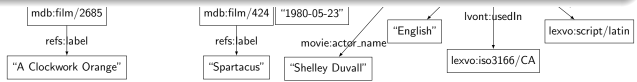
- Answering SPARQL query ≡ subgraph matching using homomorphism
- gStore [Zou et al., 2011, 2014], chameleon-db [Aluç et al., 2013]

Advantages

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

Disadvantages

- Graph pattern matching is expensive



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])

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

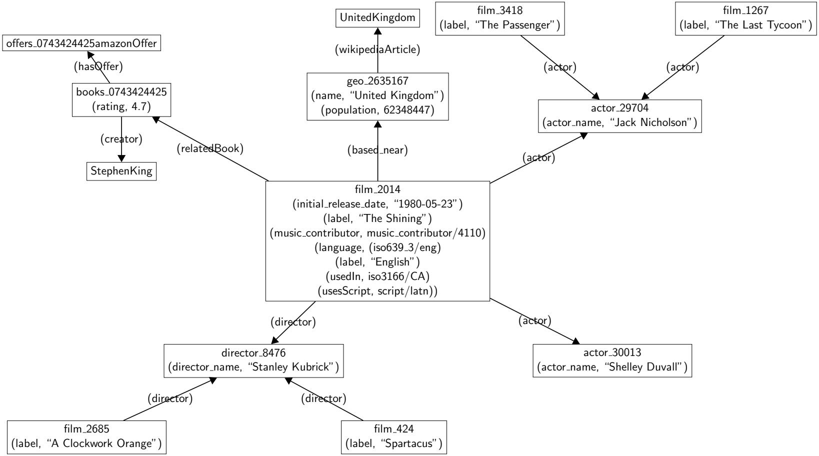
Most important ...

1. DB community really needs to get engaged to develop performant & scalable engines
2. SPARQL is not easy $\Rightarrow$ 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



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)

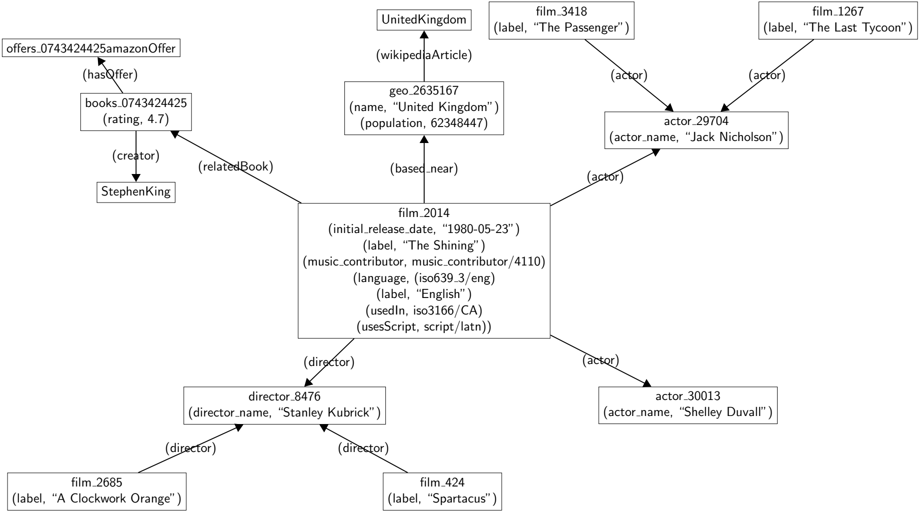
Property Graph Storage Approaches

- 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 [Bonifati et al., 2018]

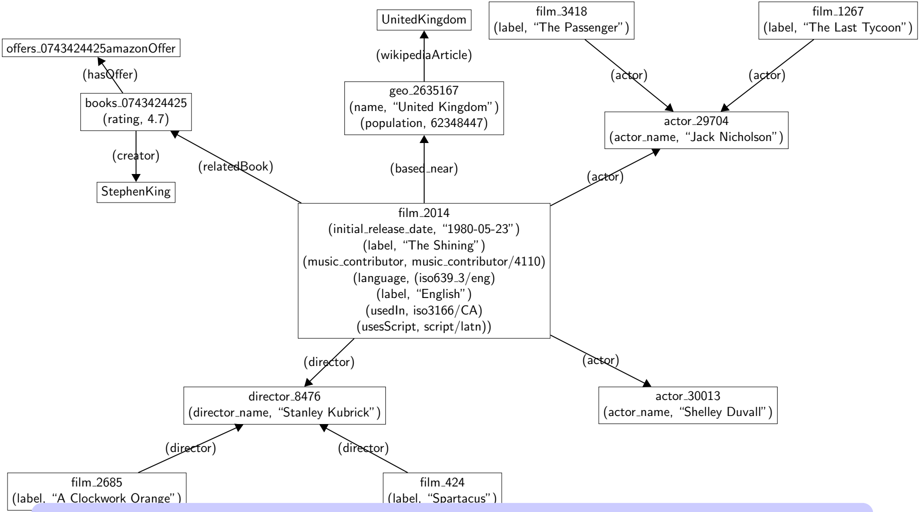
- Querying graph topology and graph properties
  - Data queries are essentially relational queries
  - Querying these are usually treated separately
- Core graph query functionalities
  - Path navigation (reachability) queries
  - Subgraph pattern queries

Reachability Queries



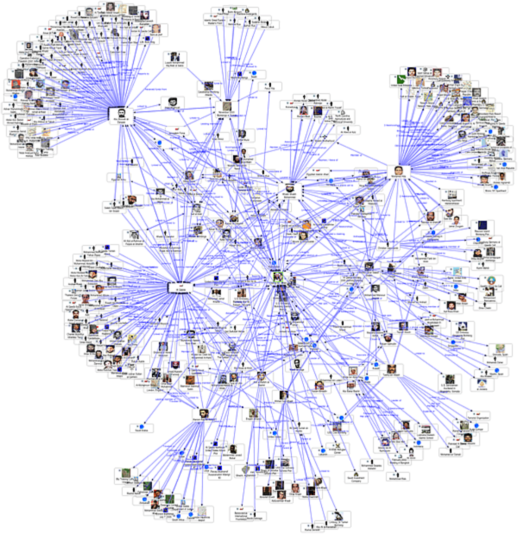
Can you reach film_1267 from film_2014?

Reachability Queries



Is there a book whose rating is > 4.0 associated with a film that was directed by Stanley Kubrick?

Reachability Queries



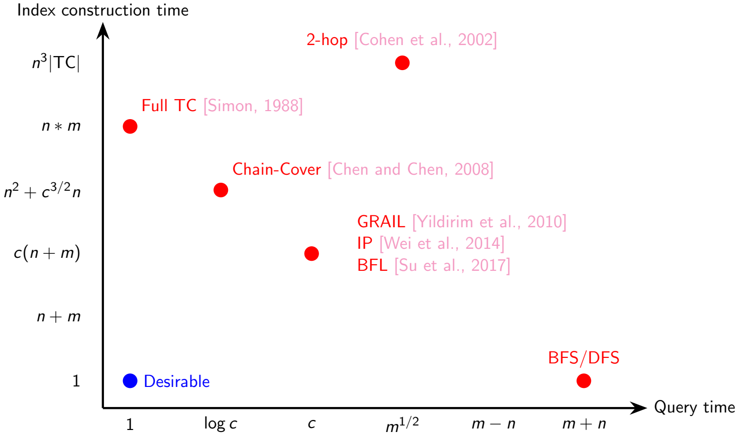
Think of Facebook graph and finding friends of friends.

Path (Reachability) Query Execution Approaches

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

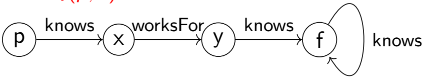
Path (Reachability) Query Execution Approaches

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



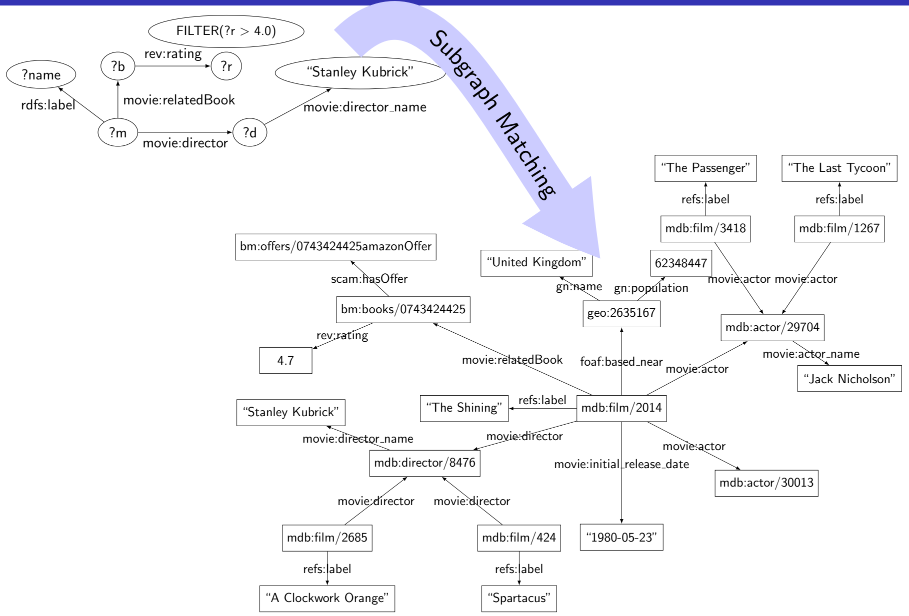
Path (Reachability) Query Execution Approaches

$Q(p, f) \leftarrow knows \cdot worksFor \cdot knows^{+}$



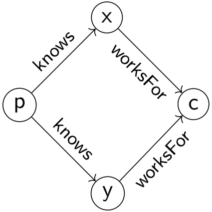
- Regular Path Queries (RPQ)
  - RPQ = path query that defines desired paths using a regular expression $\Rightarrow$ labels of a path form a word in the language specified by RPQ
  - Generalization of reachability queries $\Rightarrow$ reachability query = A RPQ that accepts all words
- α-RA – Relational 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-RA & FA-based traversals: Waveguide [Yakovets et al., 2016]

Subgraph Matching



Subgraph Pattern Query Execution Approaches

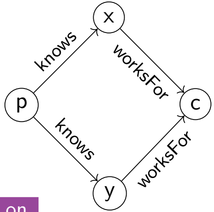
$Q(p, c) \leftarrow knows(p, x) \wedge knows(p, y) \wedge$
$worksFor(x, c) \wedge worksFor(y, c)$



- 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

Subgraph Pattern Query Execution Approaches

$$Q(p, c) \leftarrow knows(p, x) \wedge knows(p, y) \wedge worksFor(x, c) \wedge worksFor(y, c)$$



- 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, 2
  - EmptyHeaded [Aberger et al., 2
    - Generalized hypertree decompositions
  - Graphflow Hybrid [Mhedhbi and Salihoglu, 2019]
    - Adaptive, cost-based planning with WCO and binary joins

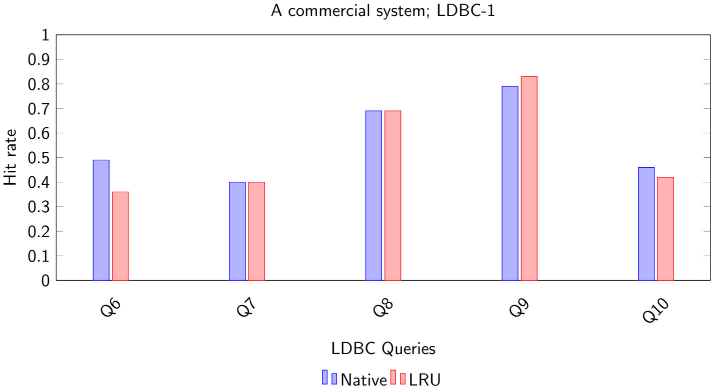
Research session 28 on Thursday 16:00

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?

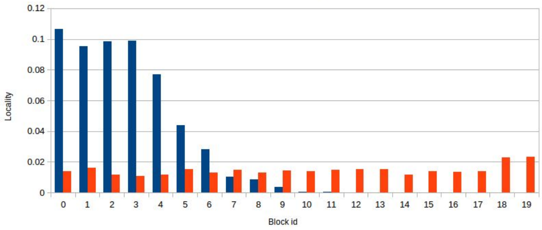
- Existing systems generally have performance issues
- There is poor locality in graph workloads
  - Caching does not help much



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

(Commercial system; LDBC-1)



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)

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 ⇒ 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 Research session 18 on Thursday @ 11:00 , 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

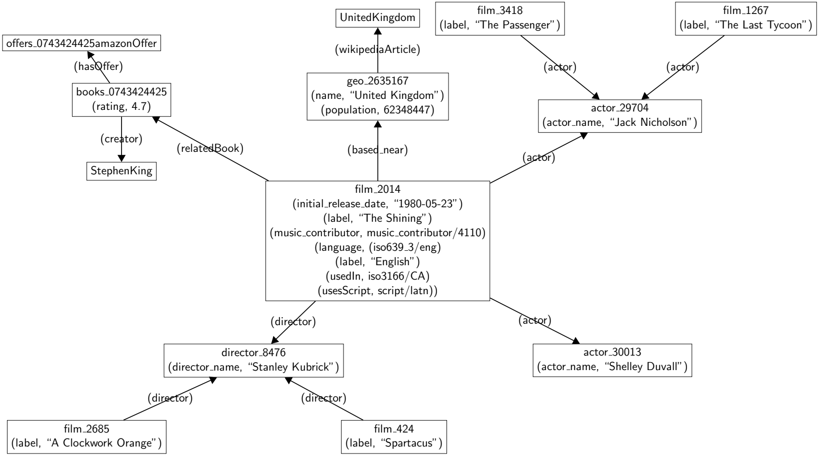
Most important ...

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 Systems

Graph Analytics System Properties

- Property graph model



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?

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

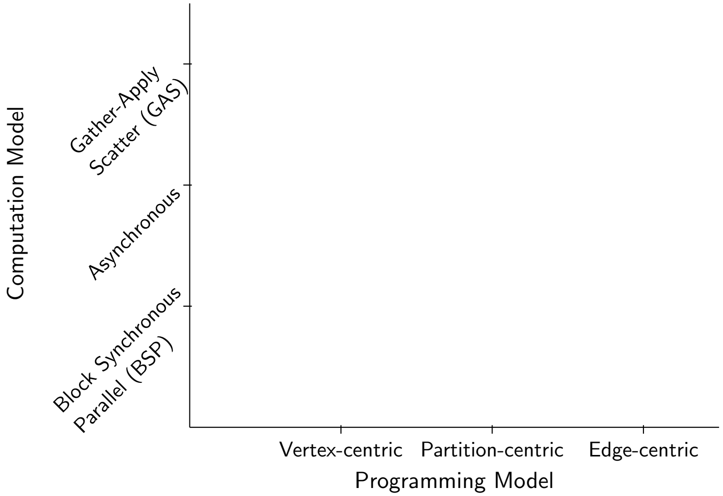
- Immutable data & computation is not guaranteed to be on the same machine in subsequent iterations
- High I/O cost due to repeated read/write to/from store between iterations

There are systems that try to address these concerns

- HaLoop [Bu et al., 2010, 2012]
- GraphX over Spark [Gonzalez et al., 2014]

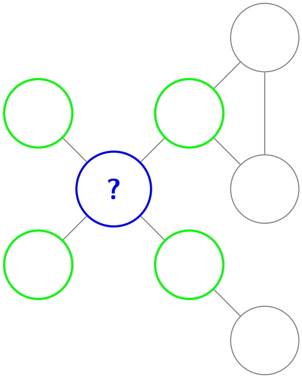
Classification of Graph Analytics Systems [Han, 2015]

- Programming model
- Computation model



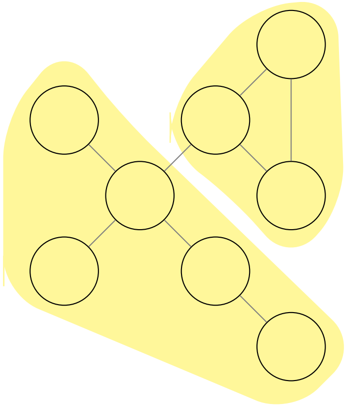
Programming Models

- 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()`



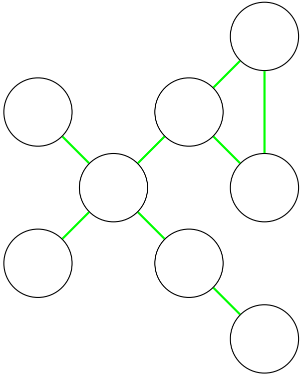
Programming Models

- 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



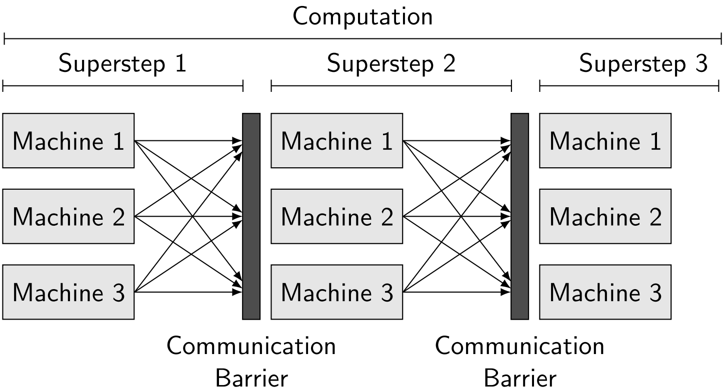
Programming Models

- 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]

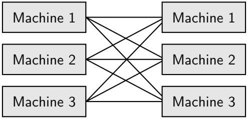


Each machine performs computation on its graph partition

At the end of each superstep results are pushed to other workers

Computational Models

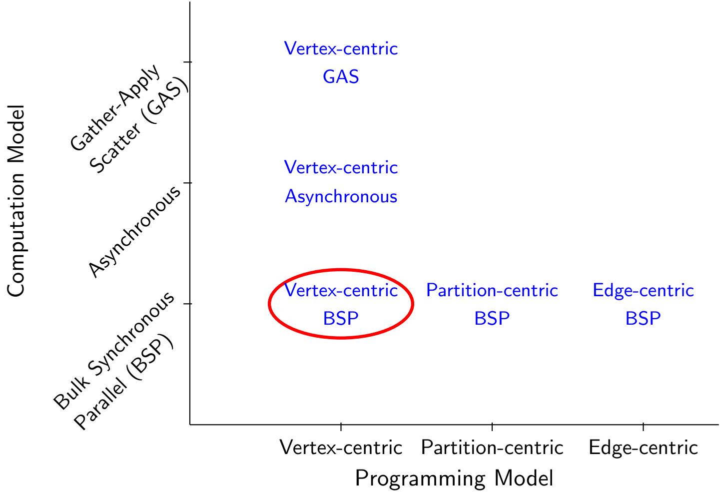
- Bulk Synchronous Parallel (BSP) [Valiant, 1990]
- Asynchronous Parallel
  - No communication barriers
  - Uses the *most recent* values
  - Implemented via distributed locking



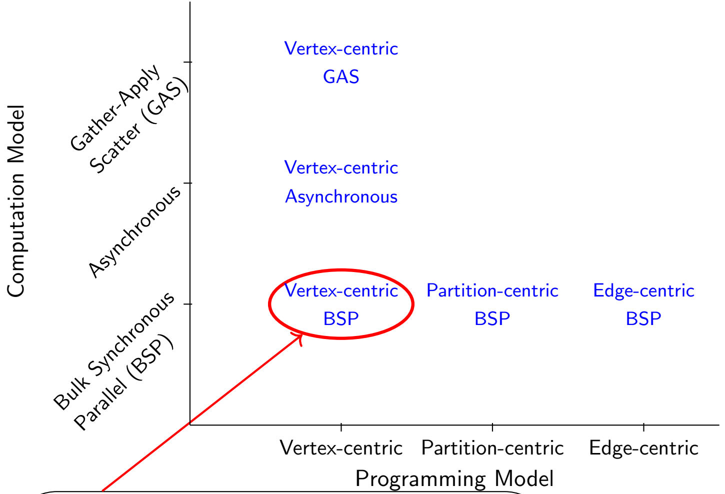
Computational Models

- 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

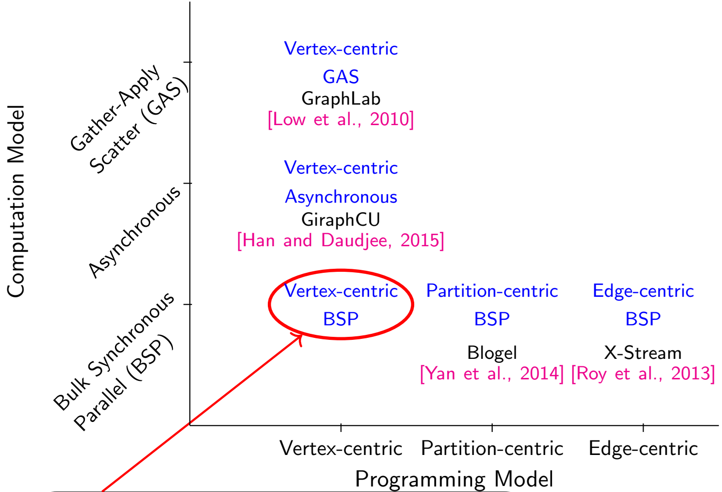


Classification of Graph Analytics Systems



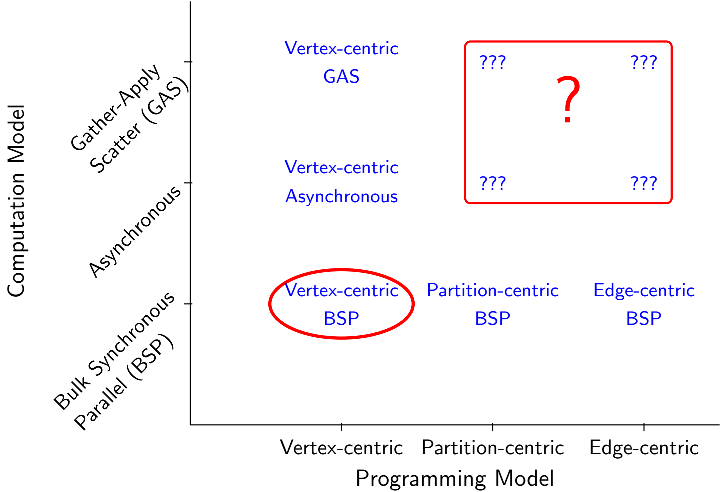
Pregel [Malewicz et al., 2010], Apache Giraph, GPS [Salihoglu and Widom, 2013], Mizan [Khayyat et al., 2013], Trinity [Shao et al., 2013]

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]

Classification of Graph Analytics Systems



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 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 $\rightarrow$ 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 $\rightarrow$ 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

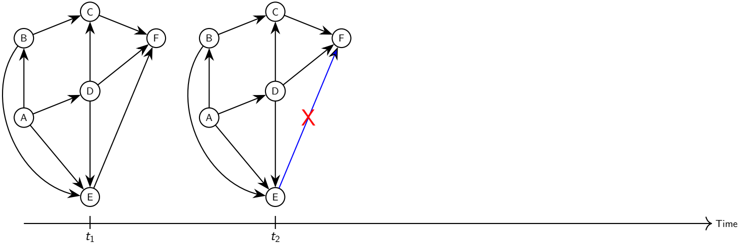
Most important...

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

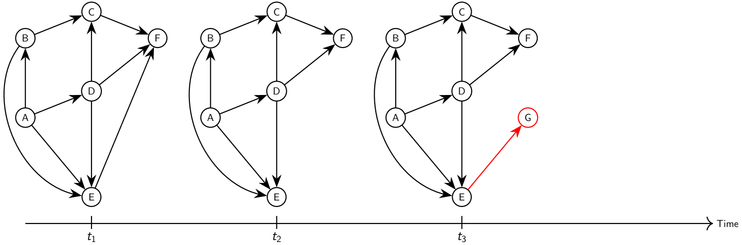
Dynamic & Streaming Graphs

Dynamic Graphs

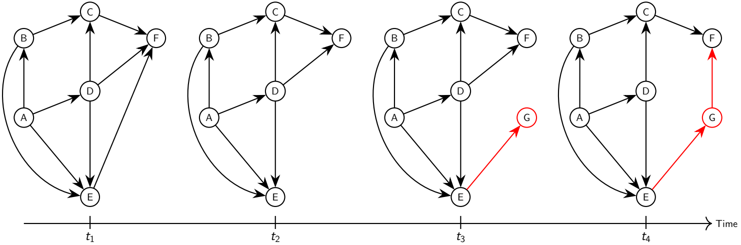
Dynamic Graphs



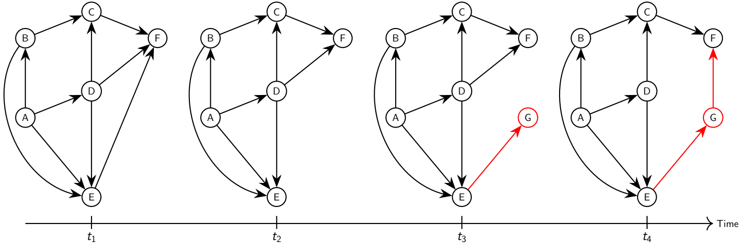
Dynamic Graphs



Dynamic Graphs



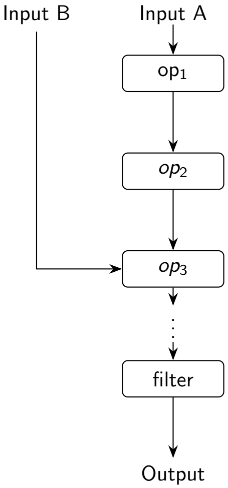
Dynamic Graphs



- 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]

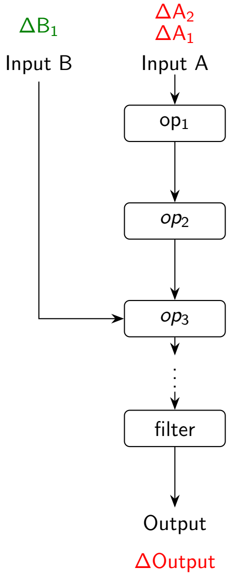
Differential Dataflow

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



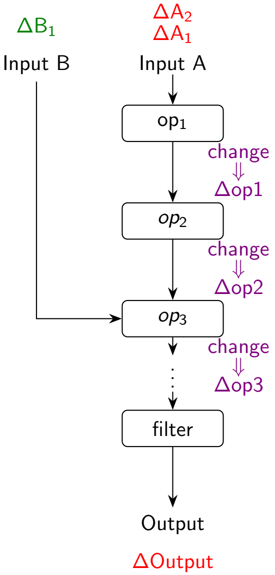
Differential Dataflow

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



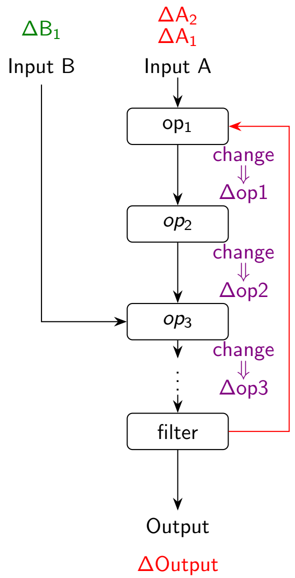
Differential Dataflow

- 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 $\Rightarrow$ early stop saves work

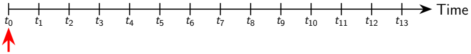


Differential Dataflow

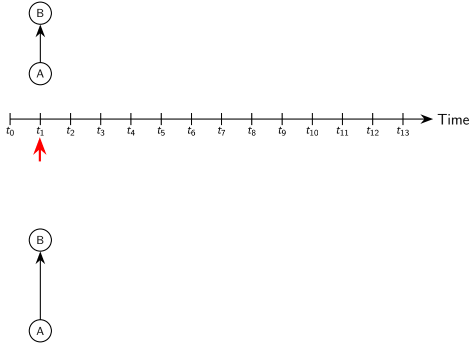
- Applies to any data flow computation
- Does not depend on the semantics of the computation
- When changes arrive, each operator is asked if there are any changes
  - If there are, push the changes to next operator
  - If not, stop $\Rightarrow$ early stop saves work
- Iterative workloads, e.g., graph 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

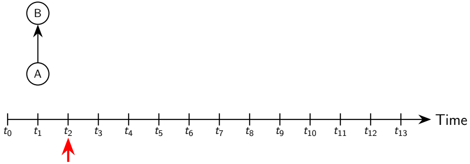


Streaming Graphs



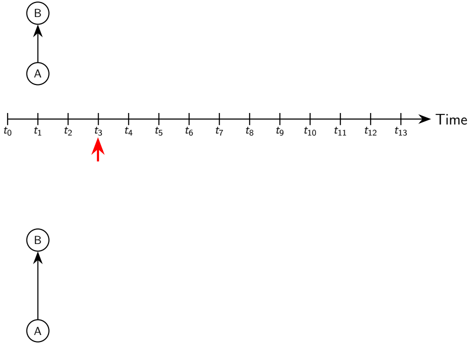
t_1

Streaming Graphs



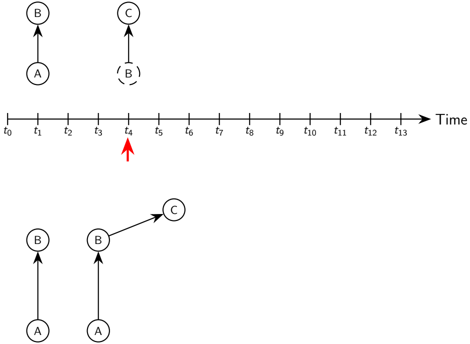
t_1

Streaming Graphs



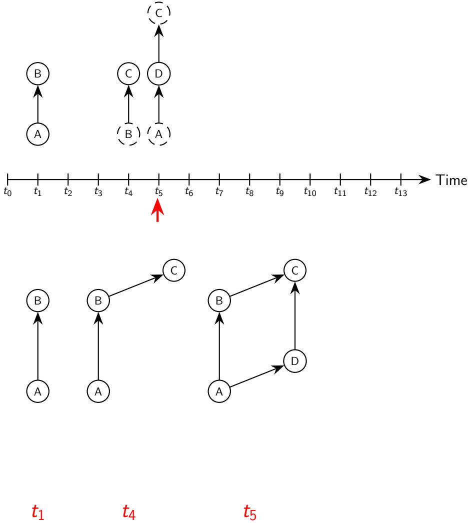
t_1

Streaming Graphs

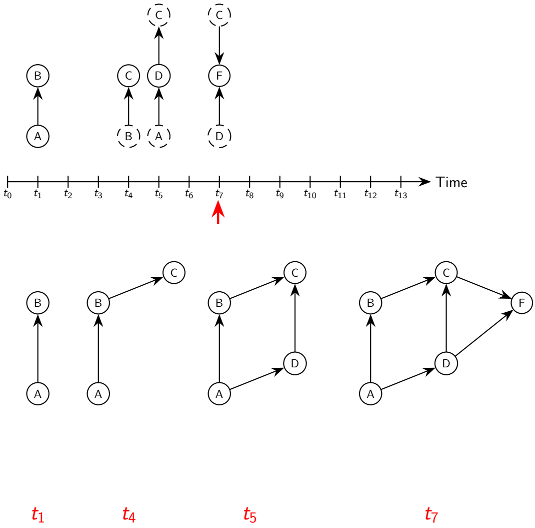


t_1 t_4

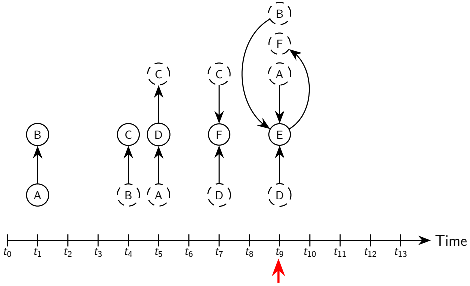
Streaming Graphs

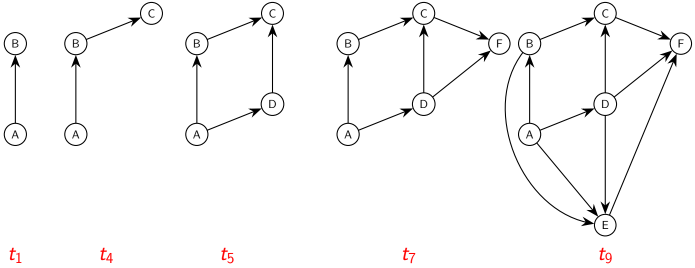


Streaming Graphs

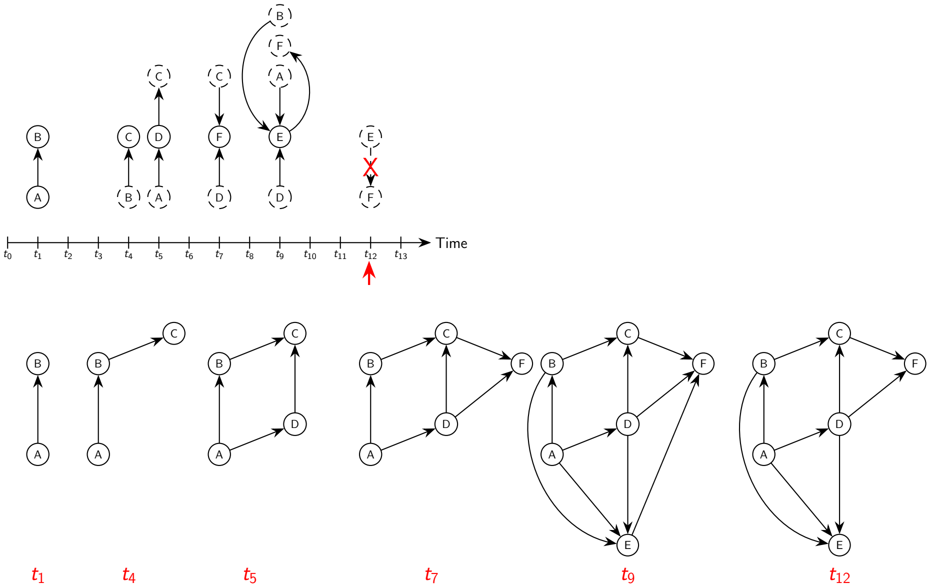


Streaming Graphs

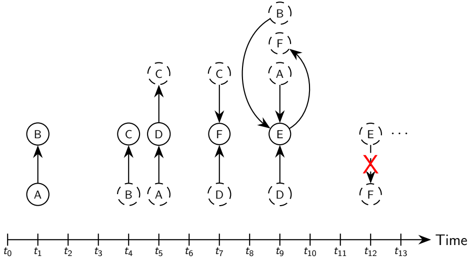




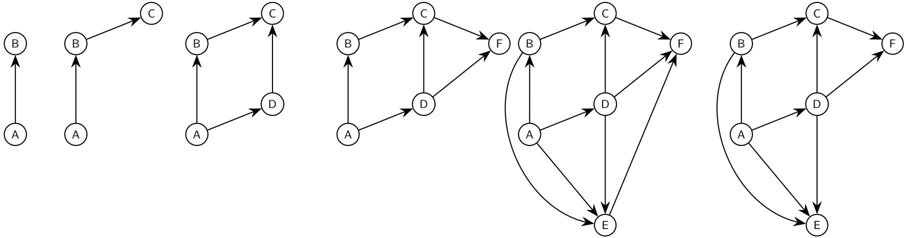
Streaming Graphs



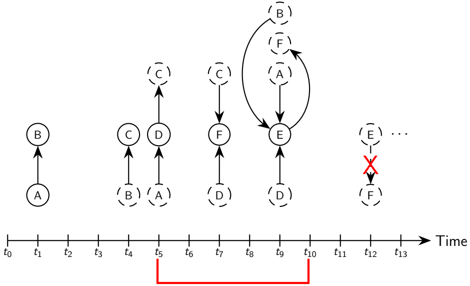
Streaming Graphs



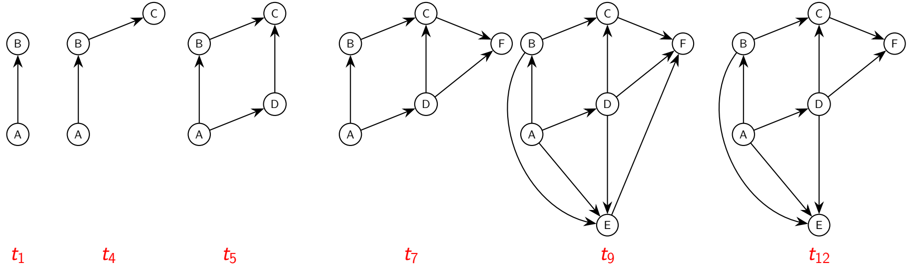
- Combines two difficult problems: streaming+graphs
- Unbounded ⇒ don't see entire graph
- Streaming rates can be very high
- Computational models
    - Continuous: for simple transactional operations



Streaming Graphs

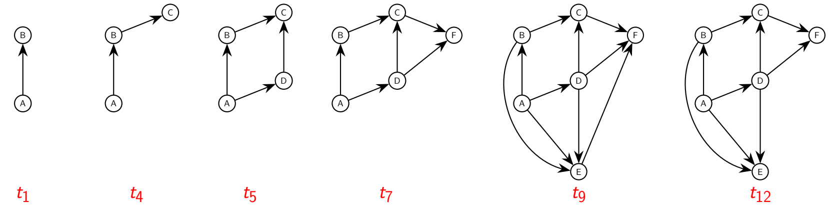


- Combines two difficult problems: streaming+graphs
- Unbounded ⇒ don't see entire graph
- Streaming rates can be very high
- Computational models
  - Continuous: for simple transactional operations
  - Windowed: for more complex queries



Continuous Computation

Query: Vertices reachable from vertex A

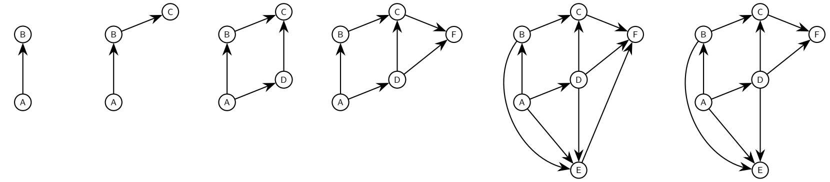


Time	Incoming edge	Results
t_1	⟨A,B⟩	{B}
t_2		
t_3		
t_4	⟨B,C⟩	{B,C}
t_5	⟨A,D⟩, ⟨D,C⟩	{B,C,D}
t_6		
t_7	⟨C,F⟩, ⟨D,F⟩	{B,C,D,F}
t_8		
t_9	⟨D,E⟩, ⟨A,E⟩, ⟨B,E⟩, ⟨E,F⟩	{B,C,D,F,E}
t_{10}		

Windowed Computation

Query: Vertices reachable from vertex A

Window size=5



t_1 t_4 t_5 t_7 t_9 t_{12}

Time	Incoming edge	Expired edges	Results
t_1	⟨A,B⟩		{B}
t_2			
t_3			
t_4	⟨B,C⟩		{B,C}
t_5	⟨A,D⟩, ⟨D,C⟩		{B,C,D}
t_6		⟨A,B⟩	{~~B~~,C,D}
t_7	⟨C,F⟩, ⟨D,F⟩		{C,D,F}
t_8			
t_9	⟨D,E⟩, ⟨A,E⟩, ⟨B,E⟩, ⟨E,F⟩	⟨B,C⟩	{C,D,F,E}
t_{10}		⟨A,D⟩, ⟨D,C⟩	{~~C,D~~,F,E}

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 $\rightarrow$ do some operation
    - Edge represents events in news $\rightarrow$ 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

Concluding Remarks

The entire field is pretty much open!...

1. We can do more with dynamic graphs, but efficient systems that incorporate novel techniques are needed
2. Unboundedness in streams raises real challenges
3. Most graph problems are unbounded under edge insert/delete

So, what is the big story?...

Take home message!...

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

Take home message!...

Performance & scaling are real problems...

1. There are few **independent** large-scale performance studies (e.g., [Rusu and Huang, 2019; Han et al., 2014; Ammar and Özsu, 2018])
2. Reasonable benchmarks are emerging: LDBC for graph DBMS [Erling et al., 2015], WatDiv for RDF [Aluç et al., 2014], Graph500 for very large graphs
3. These are application benchmarks; microbenchmarks for system testing are needed

Take home message!...

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

Take home message!...

Common DBMS for RDF & property graphs?

1. They both deal with online workloads and focus on querying
2. SPARQL only deals with subgraph queries $\Rightarrow$ 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

Take home message!...

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

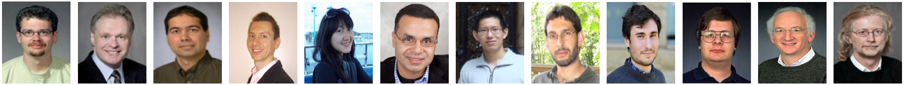
Some Issues That I Did Not Talk About

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. ...

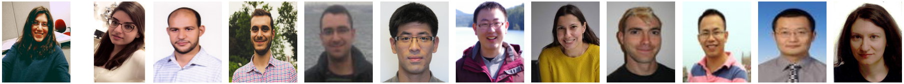
Thank you!

To the DSG group



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

To my collaborators ...



To the supporters

NSERC CRSNG INNOVATION.CA Ontario HUAWEI Google BOSCH IBM Canada Lab CAS

References I

Abadi, D. J., Marcus, A., Madden, S., and Hollenbach, K. (2009). SW-Store: a vertically partitioned DBMS for semantic web data management. *VLDB J.*, 18(2):385–406. Available from: https://doi.org/10.1007/s00778-008-0125-y.

Abadi, D. J., Marcus, A., Madden, S. R., and Hollenbach, K. (2007). Scalable semantic web data management using vertical partitioning. In *Proc. 33rd Int. Conf. on Very Large Data Bases*, pages 411–422.

Aberger, C. R., Lamb, A., Tu, S., Nötzli, A., Olukotun, K., and Ré, C. (2017). EmptyHeaded: A relational engine for graph processing. *ACM Trans. Database Syst.*, 42(4):20:1–20:44. Available from: http://doi.acm.org/10.1145/3129246.

Aluç, G., Hartig, O., Özsu, M. T., and Daudjee, K. (2014). Diversified stress testing of RDF data management systems. In *Proc. 13th Int. Semantic Web Conf.*, pages 197–212. Available from: https://doi.org/10.1007/978-3-319-11964-9_13.

Aluç, G., Özsu, M. T., Daudjee, K., and Hartig, O. (2013). chameleon-db: a workload-aware robust RDF data management system. Technical Report CS-2013-10, University of Waterloo. Available at https://cs.uwaterloo.ca/sites/ca.computer-science/files/uploads/files/CS-2013-10.pdf.

References II

Ammar, K., McSherry, F., Salihoglu, S., and Joglekar, M. (2018). Distributed evaluation of subgraph queries using worst-case optimal and low-memory dataflows. *Proc. VLDB Endowment*, 11(6):691–704. Available from: https://doi.org/10.14778/3184470.3184473.

Ammar, K. and Özsu, M. T. (2018). Experimental analysis of distributed graph systems. *Proc. VLDB Endowment*, 11(10):1151–1164. Available from: https://doi.org/10.14778/3231751.3231764.

Angles, R., Arenas, M., Barceló, P., Boncz, P. A., Fletcher, G. H. L., Gutierrez, C., Lindaaker, T., Paradies, M., Plantikow, S., Sequeda, J. F., van Rest, O., and Voigt, H. (2018). G-CORE: A core for future graph query languages. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 1421–1432. Available from: http://doi.acm.org/10.1145/3183713.3190654.

Barbieri, D. F., Braga, D., Ceri, S., Valle, E. D., and Grossniklaus, M. (2010). C-SPARQL: a continuous query language for RDF data streams. *Int. J. Semantic Computing*, 4(1):3–25. Available from: https://doi.org/10.1142/S1793351X10000936.

References III

Bonifati, A., Fletcher, G., Voigt, H., and Yakovets, N. (2018). *Querying Graphs*. Synthesis Lectures on Data Management. Morgan & Claypool. Available from: https://doi.org/10.2200/S00873ED1V01Y201808DTM051.

Bornea, M. A., Dolby, J., Kementsietsidis, A., Srinivas, K., Dantressangle, P., Udrea, O., and Bhattacharjee, B. (2013). Building an efficient RDF store over a relational database. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 121–132. Available from: http://doi.acm.org/10.1145/2463676.2463718.

Bu, Y., Howe, B., Balazinska, M., and Ernst, M. D. (2010). HaLoop: efficient iterative data processing on large clusters. *Proc. VLDB Endowment*, 3(1):285–296. Available from: http://dl.acm.org/citation.cfm?id=1920841.1920881.

Bu, Y., Howe, B., Balazinska, M., and Ernst, M. D. (2012). The HaLoop approach to large-scale iterative data analysis. *VLDB J.*, 21(2):169–190. Available from: https://doi.org/10.1007/s00778-012-0269-7.

Chen, C., Yan, X., Zhu, F., Han, J., and Yu, P. S. (2008). Graph OLAP: Towards online analytical processing on graphs. In *Proc. 8th IEEE Int. Conf. on Data Mining*, pages 103–112. Available from: https://doi.org/10.1109/ICDM.2008.30.

References IV

Chen, Y. and Chen, Y. (2008). An efficient algorithm for answering graph reachability queries. In *Proc. 24th Int. Conf. on Data Engineering*, pages 893 –902.

Cheng, R., Hong, J., Kyrola, A., Miao, Y., Weng, X., Wu, M., Yang, F., Zhou, L., Zhao, F., and Chen, E. (2012). Kineograph: Taking the pulse of a fast-changing and connected world. In *Proc. 7th ACM SIGOPS/EuroSys European Conf. on Comp. Syst.*, pages 85–98. Available from: http://doi.acm.org/10.1145/2168836.2168846.

Cohen, E., Halperin, E., Kaplan, H., and Zwick, U. (2002). Reachability and distance queries via 2-hop labels. In *Proc. 13th Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 937–946. Available from: http://doi.acm.org/10.1145/545381.545503.

Cruz, I. F., Mendelzon, A. O., and Wood, P. T. (1987). A graphical query language supporting recursion. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 323–330.

Dhulipala, L., Blelloch, G. E., and Shun, J. (2019). Low-latency graph streaming using compressed purely-functional trees. In *Proc. ACM SIGPLAN 2019 Conf. on Programming Language Design and Implementation*, pages 918–934. Available from: http://doi.acm.org/10.1145/3314221.3314598.

Ediger, D., McColl, R., Riedy, J., and Bader, D. A. (2012). STINGER: High performance data structure for streaming graphs. In *Proc. 2012 IEEE Conf. on High Performance Extreme Comp.*, pages 1–5.

Elkin, M. (2011). Streaming and fully dynamic centralized algorithms for constructing and maintaining sparse spanners. *ACM Trans. Algorithms*, 7(2):20:1–20:17. Available from: http://doi.acm.org/10.1145/1921659.1921666.

Erling, O., Averbuch, A., Larriba-Pey, J., Chafi, H., Gubichev, A., Prat, A., Pham, M.-D., and Boncz, P. (2015). The ldbc social network benchmark: Interactive workload. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 619–630. Available from: http://doi.acm.org/10.1145/2723372.2742786.

Fan, W., Li, J., Luo, J., Tan, Z., Wang, X., and Wu, Y. (2011). Incremental graph pattern matching. In *Proc. ACM International Conference on Management of Data*, pages 925–936.

Farid, M. H., Roatis, A., Ilyas, I. F., Hoffmann, H., and Chu, X. (2016). CLAMS: bringing quality to data lakes. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 2089–2092. Available from: https://doi.org/10.1145/2882903.2899391.

References VI

Feigenbaum, J., Kannan, S., McGregor, A., Suri, S., and Zhang, J. (2005). On graph problems in a semi-streaming model. *Theor. Comp. Sci.*, 348(2):207–216. Available from: http://www.sciencedirect.com/science/article/pii/S0304397505005323.

Galarraga, L., Hose, K., and Schenkel, R. (2014). Partout: a distributed engine for efficient RDF processing. In *Proc. 26th Int. World Wide Web Conf. (Companion Volume)*, pages 267–268. Available from: http://doi.acm.org/10.1145/2567948.2577302.

Gonzalez, J. E., Xin, R. S., Dave, A., Crankshaw, D., Franklin, M. J., and Stoica, I. (2014). GraphX: graph processing in a distributed dataflow framework graph processing in a distributed dataflow framework. In *Proc. 11th USENIX Symp. on Operating System Design and Implementation*, pages 599–613. Available from: https://www.usenix.org/conference/osdi14/technical-sessions/presentation/gonzalez.

Han, M. (2015). On improving distributed Pregel-like graph processing systems. Master's thesis, University of Waterloo, David R. Cheriton School of Computer Science. Available from: http://hdl.handle.net/10012/9484.

References VII

Han, M. and Daudjee, K. (2015). Giraph unchained: Barrierless asynchronous parallel execution in Pregel-like graph processing systems. *Proc. VLDB Endowment*, 8(9):950–961. Available from: http://www.vldb.org/pvldb/vol8/p950-han.pdf.

Han, M., Daudjee, K., Ammar, K., Özsu, M. T., Wang, X., and Jin, T. (2014). An experimental comparison of Pregel-like graph processing systems. *Proc. VLDB Endowment*, 7(12):1047–1058. Available from: http://www.vldb.org/pvldb/vol7/p1047-han.pdf.

Hose, K. and Schenkel, R. (2013). WARP: Workload-aware replication and partitioning for RDF. In *Proc. Workshops of 29th Int. Conf. on Data Engineering*, pages 1–6. Available from: http://dx.doi.org/10.1109/ICDEW.2013.6547414.

Huang, J., Abadi, D. J., and Ren, K. (2011). Scalable SPARQL querying of large RDF graphs. *Proc. VLDB Endowment*, 4(11):1123–1134.

Husain, M. F., McGlothlin, J., Masud, M. M., Khan, L. R., and Thuraisingham, B. (2011). Heuristics-based query processing for large RDF graphs using cloud computing. *IEEE Trans. Knowl. and Data Eng.*, 23(9):1312–1327.

References VIII

Kankanamge, C., Sahu, S., Mhedbhi, A., Chen, J., and Salihoglu, S. (2017). Graphflow: An active graph database. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 1695–1698. Available from: http://doi.acm.org/10.1145/3035918.3056445.

Kaoudi, Z. and Manolescu, I. (2015). RDF in the clouds: A survey. *VLDB J.*, 24:67–91.

Kapralov, M. (2013). Better bounds for matchings in the streaming model. In *Proc. 24th Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 1679–1697. Available from: http://dl.acm.org/citation.cfm?id=2627817.2627938.

Khadilkar, V., Kantarcioglu, M., Thuraisingham, B. M., and Castagna, P. (2012). Jena-HBase: A distributed, scalable and efficient RDF triple store. In *Proc. International Semantic Web Conference Posters & Demos Track*.

Khayyat, Z., Awara, K., Alonazi, A., Jamjoom, H., Williams, D., and Kalnis, P. (2013). Mizan: A system for dynamic load balancing in large-scale graph processing. In *Proc. 8th ACM SIGOPS/EuroSys European Conf. on Comp. Syst.*, pages 169–182. Available from: http://doi.acm.org/10.1145/2465351.2465369.

References IX

Kim, K., Seo, I., Han, W., Lee, J., Hong, S., Chafi, H., Shin, H., and Jeong, G. (2018). Turboflux: A fast continuous subgraph matching system for streaming graph data. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 411–426.

Lee, K. and Liu, L. (2013). Scaling queries over big RDF graphs with semantic hash partitioning. *Proc. VLDB Endowment*, 6(14):1894–1905. Available from: http://www.vldb.org/pvldb/vol6/p1894-lee.pdf.

Low, Y., Gonzalez, J., Kyrola, A., Bickson, D., and Guestrin, C. (2010). GraphLab: new framework for parallel machine learning. In *Proc. 26th Conf. on Uncertainty in Artificial Intelligence*, pages 340–349.

Maali, F., Campinas, S., and Decker, S. (2015). Gagg: A graph aggregation operator. In *Proc. 14th Int. Semantic Web Conf.*, pages 491–504.

Malewicz, G., Austern, M. H., Bik, A. J. C., Dehnert, J. C., Horn, I., Leiser, N., and Czajkowski, G. (2010). Pregel: a system for large-scale graph processing. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 135–146.

Mariappan, M. and Vora, K. (2019). GraphBolt: Dependency-driven synchronous processing of streaming graphs. In *Proc. 14th ACM SIGOPS/EuroSys European Conf. on Comp. Syst.*, pages 25:1–25:16. Available from: http://doi.acm.org/10.1145/3302424.3303974.

References X

Mattson, T., Bader, D., Berry, J., Buluc, A., Dongarra, J., Faloutsos, C., Feo, J., Gilbert, J., Gonzalez, J., Hendrickson, B., Kepner, J., Leiserson, C., Lumsdaine, A., Padua, D., Poole, S., Reinhardt, S., Stonebraker, M., Wallach, S., and Yoo, A. (2013). Standards for graph algorithm primitives. In *Proc. 2013 IEEE High Performance Extreme Comp. Conf.*, pages 1–2.

McColl, R., Green, O., and Bader, D. A. (2013). A new parallel algorithm for connected components in dynamic graphs. In *20th Annual Int. Conf. High Performance Computing*, pages 246–255.

McGregor, A. (2014). Graph stream algorithms: A survey. *ACM SIGMOD Rec.*, 43(1):9–20. Available from: http://doi.acm.org/10.1145/2627692.2627694.

McSherry, F., Murray, D. G., Isaacs, R., and Isard, M. (2013). Differential dataflow. In *Proc. 6th Biennial Conf. on Innovative Data Systems Research*.

Mendelzon, A. O. and Wood, P. T. (1995). Finding regular simple paths in graph databases. *SIAM J. on Comput.*, 24(6):1235—1258.

Mhedhbi, A. and Salihoglu, S. (2019). Optimizing subgraph queries by combining binary and worst-case optimal joins. *Proc. VLDB Endowment*, 12.

References XI

Nannicini, G. and Liberti, L. (2008). Shortest paths on dynamic graphs. *Int. Trans. Operations Research*, 15(5):551–563. Available from: https://doi.org/10.1111/j.1475-3995.2008.00649.x.

Neumann, T. and Weikum, G. (2008). RDF-3X: a RISC-style engine for RDF. *Proc. VLDB Endowment*, 1(1):647–659.

Neumann, T. and Weikum, G. (2009). The RDF-3X engine for scalable management of RDF data. *VLDB J.*, 19(1):91–113.

Peng, P., Zou, L., Özsu, M. T., Chen, L., and Zhao, D. (2016). Processing SPARQL queries over distributed RDF graphs. *VLDB J.*, 25(2):243–268.

Phuoc, D. L., Dao-Tran, M., Parreira, J. X., and Hauswirth, M. (2011). A native and adaptive approach for unified processing of linked streams and linked data. In *Proc. 10th Int. Semantic Web Conf.*, pages 370–388. Available from: https://doi.org/10.1007/978-3-642-25073-6_24.

Qiu, X., Cen, W., Qian, Z., Peng, Y., Zhang, Y., Lin, X., and Zhou, J. (2018). Real-time constrained cycle detection in large dynamic graphs. *Proc. VLDB Endowment*, 11(12):1876–1888.

Reutter, J. L., Romero, M., and Vardi, M. Y. (2017). Regular queries on graph databases. *Theory of Computing Systems*, 61(1):31–83. Available from: https://doi.org/10.1007/s00224-016-9676-2.

Rohloff, K. and Schantz, R. E. (2010). High-performance, massively scalable distributed systems using the mapreduce software framework: the shard triple-store. In *Proc. Int. Workshop on Programming Support Innovations for Emerging Distributed Applications*. Article No. 4.

Roy, A., Mihailovic, I., and Zwaenepoel, W. (2013). X-stream: edge-centric graph processing using streaming partitions. In *Proc. 24th ACM Symp. on Operating System Principles*, pages 472–488. Available from: http://doi.acm.org/10.1145/2517349.2522740.

Rusu, F. and Huang, Z. (2019). In-depth benchmarking of graph database systems with the linked data benchmark council (LDBC) social network benchmark (SNB). *CoRR*, abs/1907.07405. Available from: http://arxiv.org/abs/1907.07405.

Sahu, S., Mhedhbi, A., Salihoglu, S., Lin, J., and Özsu, M. T. (2017). The ubiquity of large graphs and surprising challenges of graph processing. *Proc. VLDB Endowment*, 11(4):420–431.

References XIII

Sahu, S., Mhedhbi, A., Salihoglu, S., Lin, J., and Özsu, M. T. (2019). The ubiquity of large graphs and surprising challenges of graph processing. *VLDB J.*

Salihoglu, S. and Widom, J. (2013). GPS: a graph processing system. In *Proc. 25th Int. Conf. on Scientific and Statistical Database Management*, pages 22:1–22:12. Available from: http://doi.acm.org/10.1145/2484838.2484843.

Salihoglu, S. and Widom, J. (2014). HelP: high-level primitives for large-scale graph processing. In *Proc. 2nd Int. Workshop on Graph Data Management Experiences and Systems*, pages 3:1–3:6. Available from: http://doi.acm.org/10.1145/2621934.2621938.

Sengupta, D., Sundaram, N., Zhu, X., Willke, T. L., Young, J., Wolf, M., and Schwan, K. (2016). GraphIn: An online high performance incremental graph processing framework. In *Proc. 22nd Int. Euro-Par Conf.*, pages 319–333. Available from: http://dx.doi.org/10.1007/978-3-319-43659-3_24.

Shao, B., Wang, H., and Li, Y. (2013). Trinity: a distributed graph engine on a memory cloud. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 505–516. Available from: http://doi.acm.org/10.1145/2463676.2467799.

References XIV

Sheng, F., Cao, Q., Cai, H., Yao, J., and Xie, C. (2018). Grapu: Accelerate streaming graph analysis through preprocessing buffered updates. In *Proc. 9th ACM Symp. on Cloud Computing*, pages 301–312. Available from: http://doi.acm.org/10.1145/3267809.3267811.

Simon, K. (1988). An improved algorithm for transitive closure on acyclic digraphs. *Theor. Comp. Sci.*, 58(1-3):325–346. Available from: http://dx.doi.org/10.1016/0304-3975(88)90032-1.

Su, J., Zhu, Q., Wei, H., and Yu, J. X. (2017). Reachability querying: Can it be even faster? *IEEE Trans. Knowl. and Data Eng.*, 29(3):683–697.

Tena Cucala, D., Cuenca Grau, B., and Horrocks, I. (2019). 15 years of consequence-based reasoning. In Lutz, C., Sattler, U., Tinelli, C., Turhan, A.-Y., and Wolter, F., editors, *Description Logic, Theory Combination, and All That: Essays Dedicated to Franz Baader on the Occasion of His 60th Birthday*, pages 573–587. Springer International Publishing. Available from: https://doi.org/10.1007/978-3-030-22102-7_27.

Tian, Y., Hankins, R. A., and Patel, J. M. (2008). Efficient aggregation for graph summarization. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 567–580.

References XV

Valiant, L. G. (1990). A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111.

Veldhuizen, T. L. (2014). Leapfrog triejoin: A simple, worst-case optimal join algorithm. In *Proc. 17th Int. Conf. on Database Theory*, pages 96–106.

Wang, Z., Fan, Q., Wang, H., Tan, K. L., Agrawal, D., and El Abbadi, A. (2014). Pagrol: PArallel GRaph OLap over lege-scale attributed graphs. In *Proc. 30th Int. Conf. on Data Engineering*, pages 496–507.

Wei, H., Yu, J. X., Lu, C., and Jin, R. (2014). Reachability querying: An independent permutation labeling approach. *Proc. VLDB Endowment*, 7(12):1191–1202. Available from: http://www.vldb.org/pvldb/vol7/p1191-wei.pdf.

Weiss, C., Karras, P., and Bernstein, A. (2008). Hexastore: sextuple indexing for semantic web data management. *Proc. VLDB Endowment*, 1(1):1008–1019.

Wilkinson, K. (2006). Jena property table implementation. Technical Report HPL-2006-140, HP Laboratories Palo Alto.

Yakovets, N., Godfrey, P., and Gryz, J. (2016). Query planning for evaluating SPARQL property paths. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 1875–1889.

Yan, D., Cheng, J., Lu, Y., and Ng, W. (2014). Blogel: A block-centric framework for distributed computation on real-world graphs. *Proc. VLDB Endowment*, 7(14):1981–1992. Available from: http://www.vldb.org/pvldb/vol7/p1981-yan.pdf.

Yan, D., Cheng, J., Özsu, M. T., Yang, F., Lu, Y., Liu, J. C., Zhang, Q., and Ng, W. (2016). A general-purpose query-centric framework for querying big graphs. *Proc. VLDB Endowment*, 9(7):564 – 575. Available from: http://www.vldb.org/pvldb/vol9/p564-yan.pdf.

Yildirim, H., Chaoji, V., and Zaki, M. J. (2010). GRAIL: scalable reachability index for large graphs. *Proc. VLDB Endowment*, 3(1):276–284. Available from: http://dl.acm.org/citation.cfm?id=1920841.1920879.

Yuan, Y., Wang, G., Chen, L., and Wang, H. (2012). Efficient subgraph similarity search on large probabilistic graph databases. *Proc. VLDB Endowment*, 5(9):800–811.

Yuan, Y., Wang, G., Wang, H., and Chen, L. (2011). Efficient subgraph search over large uncertain graphs. *Proc. VLDB Endowment*, 4(11):876–886.

References XVII

Zhang, X., Chen, L., Tong, Y., and Wang, M. (2013). EAGRE: towards scalable I/O efficient SPARQL query evaluation on the cloud. In *Proc. 29th Int. Conf. on Data Engineering*, pages 565–576.

Zhang, X. and Özsu, M. T. (2019). Correlation constraint shortest path over large multi-relation graphs. *Proc. VLDB Endowment*, 12(5):488 – 501.

Zhao, P., Li, X., Xin, D., and Han, J. (2011). Graph cube: on warehousing and OLAP multidimensional networks. In *Proc. ACM International Conference on Management of Data*, pages 853–864.

Zou, L., Mo, J., Chen, L., Özsu, M. T., and Zhao, D. (2011). gStore: answering SPARQL queries via subgraph matching. *Proc. VLDB Endowment*, 4(8):482–493.

Zou, L. and Özsu, M. T. (2017). Graph-based RDF data management. *Data Science and Engineering*, 2(1):56–70. Available from: https://dx.doi.org/10.1007/s41019-016-0029-6.

Zou, L., Özsu, M. T., Chen, L., Shen, X., Huang, R., and Zhao, D. (2014). gStore: A graph-based SPARQL query engine. *VLDB J.*, 23(4):565–590.