Distributed Set Reachability

Sairam Gurajada* and Martin Theobald†

*Max-Plack Institute for Informatics, †University of Ulm

Germany

SIGMOD 2016, San Francisco, USA

Distributed Set Reachability

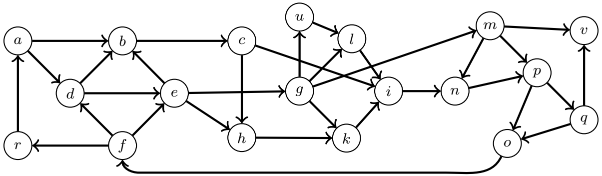
Distributed Set Reachability (DSR) is a generalization of graph reachability problem

- extended to sets
- in a distributed setting

Distributed Set Reachability

Distributed Set Reachability (DSR) is a generalization of graph reachability problem

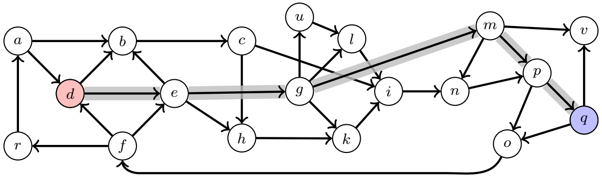
- extended to sets
- in a distributed setting



Distributed Set Reachability

Distributed Set Reachability (DSR) is a generalization of graph reachability problem

- extended to sets
- in a distributed setting



Graph G

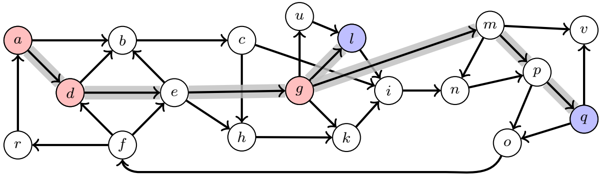
$d \rightsquigarrow q$: True

- **Reachability.** $s \rightsquigarrow t$, find if there exists a path from s to t in G [SABW13, YCZ10]

Distributed Set Reachability

Distributed Set Reachability (DSR) is a generalization of graph reachability problem

- extended to sets
- in a distributed setting



Graph G

Reachable pairs

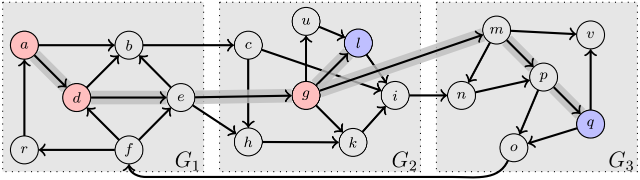
$\langle a, l\rangle, \langle a, q\rangle$
$\langle d, l\rangle, \langle d, q\rangle$
$\langle g, l\rangle, \langle g, q\rangle$

- **Reachability.** $s \rightsquigarrow t$, find if there exists a path from s to t in G [SABW13, YCZ10]
- **Set Reachability.** $S \rightsquigarrow T$, finds all pairs $\langle s, t\rangle$, such that $s \in S$, $t \in T$ and $s \rightsquigarrow t$ in G [TKC+14, GA13]

Distributed Set Reachability

Distributed Set Reachability (DSR) is a generalization of graph reachability problem

- extended to sets
- in a distributed setting



Graph G is partitioned into G_1, G_2, G_3

Reachable pairs

$\langle a, l\rangle, \langle a, q\rangle$

$\langle d, l\rangle, \langle d, q\rangle$

$\langle g, l\rangle, \langle g, q\rangle$

- **Distributed Reachability.** $s \rightsquigarrow t$, find if there exists a path from s to t in G [FWW12]
- **Distributed Set Reachability.** $S \rightsquigarrow T$, finds all pairs $\langle s, t\rangle$, such that $s \in S$, $t \in T$ and $s \rightsquigarrow t$ in G [?]

Applications

Application 1: SPARQL 1.1 property paths processing on knowledge graphs

Applications

Application 1: SPARQL 1.1 property paths processing on knowledge graphs

- Example: Find all people born in Europe who won a Nobel Prize

```
SELECT ?person WHERE {
?person <bornIn> ?city .   ?person <won> "Nobel Prize" .
?city <locatedIn>* ?country .   ?country <partOf> "Europe".}
```

Applications

Application 1: SPARQL 1.1 property paths processing on knowledge graphs

- Example: Find all people born in Europe who won a Nobel Prize

```
SELECT ?person WHERE {
?person <bornIn> ?city .   ?person <won> "Nobel Prize" .
?city <locatedIn>* ?country .   ?country <partOf> "Europe".}
```

$$\textcolor{red}{Cities} = \left\{\begin{array}{c}\text{Antwerp}\\ \text{Saarbrücken}\\ \text{Ulm}\end{array}\right\} \overset{\text{locatedIn}*}{\rightsquigarrow} \textcolor{blue}{Countries} = \left\{\begin{array}{c}\text{Belgium}\\ \text{France}\\ \text{Germany}\end{array}\right\}$$

Applications

Application 1: SPARQL 1.1 property paths processing on knowledge graphs

- Example: Find all people born in Europe who won a Nobel Prize

```
SELECT ?person WHERE {
?person <bornIn> ?city .   ?person <won> "Nobel Prize" .
?city <locatedIn>* ?country .   ?country <partOf> "Europe".}
```

$$\textcolor{red}{Cities} = \left\{ \begin{array}{c} \text{Antwerp} \\ \text{Saarbrücken} \\ \text{Ulm} \end{array} \right\} \overset{\text{locatedIn}*}{\rightsquigarrow} \textcolor{blue}{Countries} = \left\{ \begin{array}{c} \text{Belgium} \\ \text{France} \\ \text{Germany} \end{array} \right\}$$

Set Reachability Query:
$\textcolor{red}{Cities} \rightsquigarrow \textcolor{blue}{Countries}$

Applications

Application 2: Community connectedness in social networks

Applications

Application 2: Community connectedness in social networks

- Example: How billionaries and philanthropic organizations are connected

$$\color{red}{Billionaries} = \left\{\begin{array}{c}\text{Bill Gates}\\ \text{Warren Buffet}\\ \text{Donald J. Trump}\end{array}\right\} \qquad \color{blue}{Organizations} = \left\{\begin{array}{c}\text{The Giving Pledge}\\ \text{Ford Foundation}\\ \text{Melinda Gates Foundation}\end{array}\right\}$$

Applications

Application 2: Community connectedness in social networks

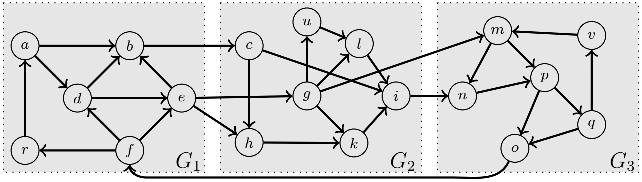
- Example: How billionaries and philanthropic organizations are connected

$$Billionaries = \left\{ \begin{array}{c} \text{Bill Gates} \\ \text{Warren Buffet} \\ \text{Donald J. Trump} \end{array} \right\} \qquad Organizations = \left\{ \begin{array}{c} \text{The Giving Pledge} \\ \text{Ford Foundation} \\ \text{Melinda Gates Foundation} \end{array} \right\}$$

Set Reachability Query:
$Billionaries \rightsquigarrow Organizations$

How to solve a DSR query?

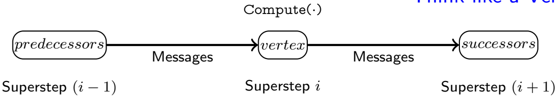
Query: $\{a, d, g\} \rightsquigarrow \{l, q\}$



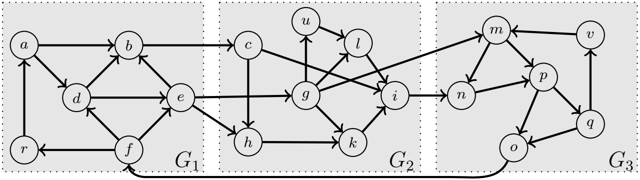
How to solve a DSR query?

Vertex-Centric approaches: Like Pregel, Giraph,...

"Think like a Vertex"



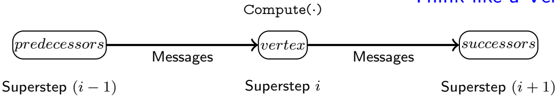
Query: $\{a, d, g\} \rightsquigarrow \{l, q\}$



How to solve a DSR query?

Vertex-Centric approaches: Like Pregel, Giraph,...

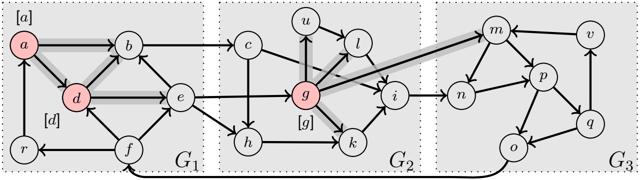
"Think like a Vertex"



Query: $\{a, d, g\} \rightsquigarrow \{l, q\}$

Supersteps

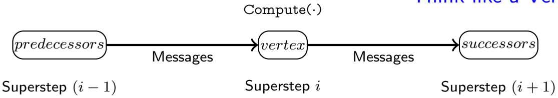
1



How to solve a DSR query?

Vertex-Centric approaches: Like Pregel, Giraph,...

"Think like a Vertex"

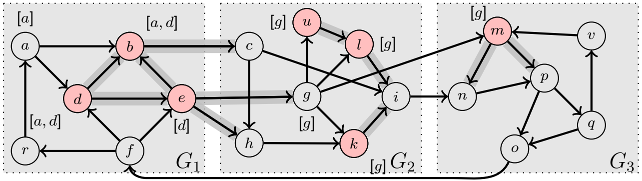


Query: $\{a, d, g\} \rightsquigarrow \{l, q\}$

Supersteps

1

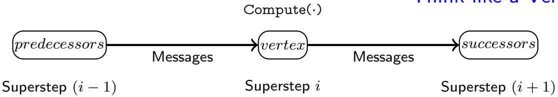
2



How to solve a DSR query?

Vertex-Centric approaches: Like Pregel, Giraph,...

"Think like a Vertex"



Query: $\{a, d, g\} \rightsquigarrow \{l, q\}$

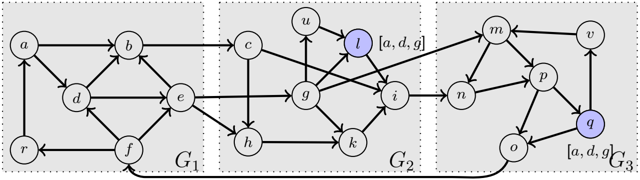
Supersteps

1

2

⋮

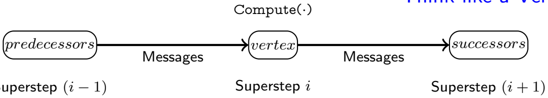
d



How to solve a DSR query?

Vertex-Centric approaches: Like Pregel, Giraph,...

"Think like a Vertex"



Query: $\{a, d, g\} \rightsquigarrow \{l, q\}$

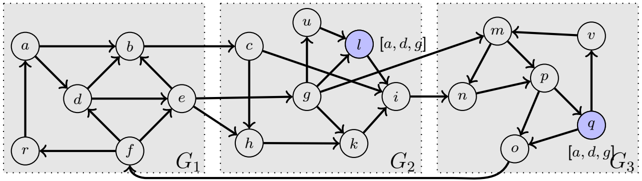
Supersteps

1

2

⋮

d



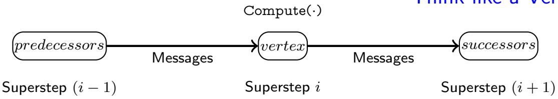
Perfomance: On small graphs ($\leq$ 10Mi edges) and query with $|S| = 10$ and $|T| = 10$

Dataset	Time(in sec)	Supersteps	Comm. Size(MB)
NotreDame	94.8	70	35.2
Stanford	341.9	267	79.1

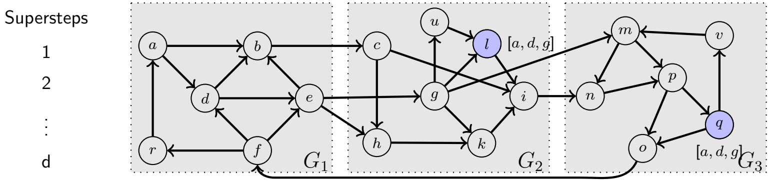
How to solve a DSR query?

Vertex-Centric approaches: Like Pregel, Giraph,...

"Think like a Vertex"



Query: $\{a, d, g\} \rightsquigarrow \{l, q\}$



Challenges:

- no reuse of computations & no support for local indexes
- leading to many iterations ($\leq$ diameter)
- and thus high communication costs and high query processing times

An efficient solution for DSR?

Objectives:

1. minimize the size and number of messages exchanged
2. to speed up & reuse computations as much as possible in local nodes
3. scalable to large graphs

An efficient solution for DSR?

Objectives:

1. **minimize the size and number of messages exchanged**
   - reachability is invariant between supersteps
   - precompute & replicate the partial reachability
     i.e., reachability among boundary nodes $\Rightarrow$ **Boundary graph**
2. to speed up & reuse computations as much as possible in local nodes
3. scalable to large graphs

An efficient solution for DSR?

Objectives:

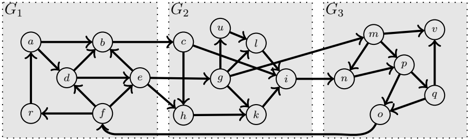
1. minimize the size and number of messages exchanged
   - reachability is invariant between supersteps
   - precompute & replicate the partial reachability
     i.e., reachability among boundary nodes ⇒ **Boundary graph**
2. **to speed up & reuse computations as much as possible in local nodes**
   - local graph + boundary graph ⇒ **Compound graph**
   - on the compound graph,
     build indexes via centralized (set) reachability approaches
     [YCZ10, SABW13, GA13, TKC+14]
3. scalable to large graphs

An efficient solution for DSR?

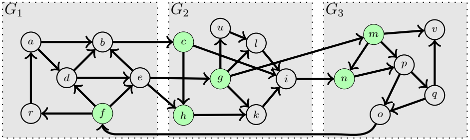
Objectives:

1. minimize the size and number of messages exchanged
   - reachability is invariant between supersteps
   - precompute & replicate the partial reachability
     i.e., reachability among boundary nodes $\Rightarrow$ **Boundary graph**
2. to speed up & reuse computations as much as possible in local nodes
   - local graph + boundary graph $\Rightarrow$ **Compound graph**
   - on the compound graph,
     build indexes via centralized (set) reachability approaches
     [YCZ10, SABW13, GA13, TKC+14]
3. **scalable to large graphs**
   - boundary graph compression via equivalence sets grouping
   - condense compound graphs by computing SCCs

Precomputation

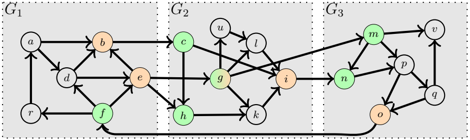


Precomputation



$I_1 = \{f\}$ $I_2 = \{c, g, h\}$ $I_3 = \{m, n\}$

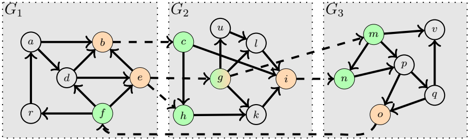
Precomputation



$I_1 = \{f\}$ $\quad$ $I_2 = \{c, g, h\}$ $\quad$ $I_3 = \{m, n\}$

$O_1 = \{b, e\}$ $\quad$ $O_2 = \{g, i\}$ $\quad$ $O_3 = \{o\}$

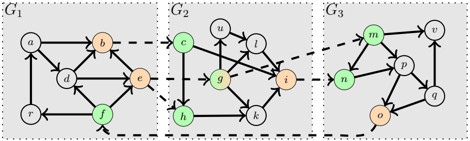
Precomputation



$I_1 = \{f\}$ $I_2 = \{c, g, h\}$ $I_3 = \{m, n\}$

$O_1 = \{b, e\}$ $O_2 = \{g, i\}$ $O_3 = \{o\}$

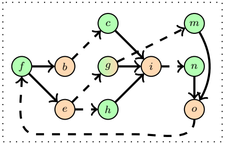
Precomputation



$I_1 = \{f\}$ $\quad$ $I_2 = \{c, g, h\}$ $\quad$ $I_3 = \{m, n\}$

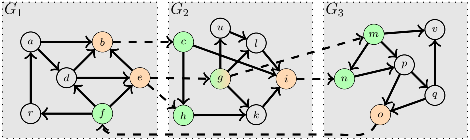
$O_1 = \{b, e\}$ $\quad$ $O_2 = \{g, i\}$ $\quad$ $O_3 = \{o\}$

Step 1: Compute reachability from $I_i \rightsquigarrow O_i$ (set reachability) **(Boundary Graph)**



Boundary Graph

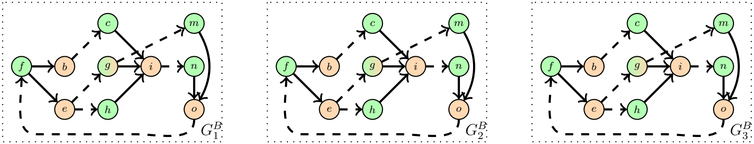
Precomputation



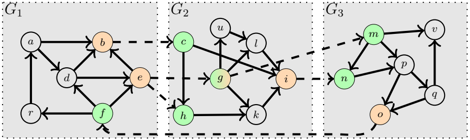
$I_1 = \{f\}$ $\quad$ $I_2 = \{c, g, h\}$ $\quad$ $I_3 = \{m, n\}$

$O_1 = \{b, e\}$ $\quad$ $O_2 = \{g, i\}$ $\quad$ $O_3 = \{o\}$

Step 1: Compute reachability from $I_i \rightsquigarrow O_i$ (set reachability) **(Boundary Graph)**



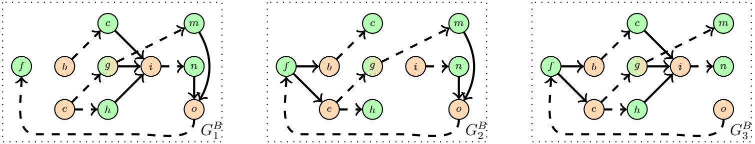
Precomputation



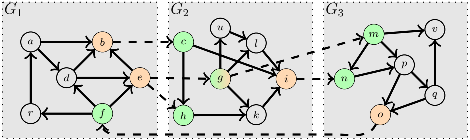
$I_1 = \{f\}$ $\quad$ $I_2 = \{c, g, h\}$ $\quad$ $I_3 = \{m, n\}$

$O_1 = \{b, e\}$ $\quad$ $O_2 = \{g, i\}$ $\quad$ $O_3 = \{o\}$

Step 1: Compute reachability from $I_i \rightsquigarrow O_i$ (set reachability) **(Boundary Graph)**



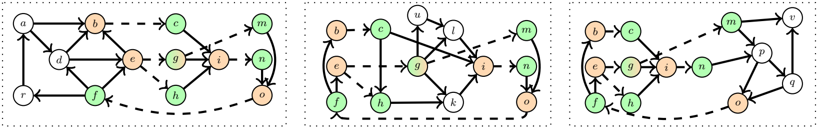
Precomputation



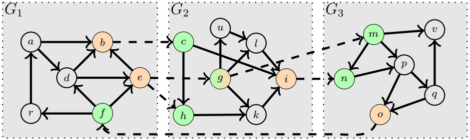
$I_1 = \{f\}$ $\quad I_2 = \{c, g, h\}$ $\quad I_3 = \{m, n\}$

$O_1 = \{b, e\}$ $\quad O_2 = \{g, i\}$ $\quad O_3 = \{o\}$

Step 1: Compute reachability from $I_i \rightsquigarrow O_i$ (set reachability) **(Boundary Graph)**
Step 2: Merge local graph and boundary graph **(Compound Graph)**



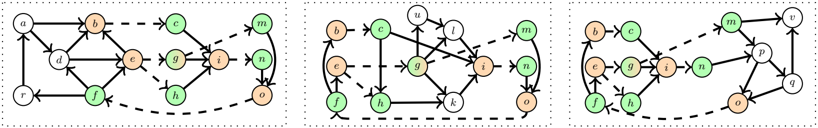
Precomputation



$I_1 = \{f\}$ $\quad I_2 = \{c, g, h\}$ $\quad I_3 = \{m, n\}$

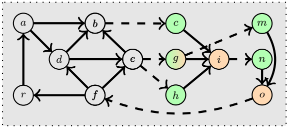
$O_1 = \{b, e\}$ $\quad O_2 = \{g, i\}$ $\quad O_3 = \{o\}$

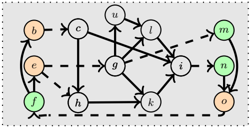
Step 1: Compute reachability from $I_i \rightsquigarrow O_i$ (set reachability) **(Boundary Graph)**
Step 2: Merge local graph and boundary graph **(Compound Graph)**

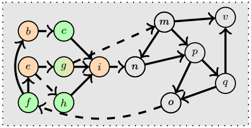


Step 3: Optionally, build local indexes to reuse computations and speed up query processing

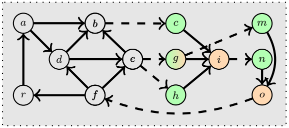
Query processing



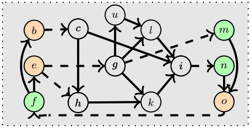




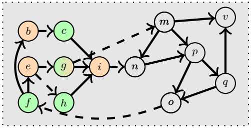
Query processing



Forward List: $F_1 = \{c, g, h, m, n\}$
Backward List: $B_1 = \{g, i, o\}$



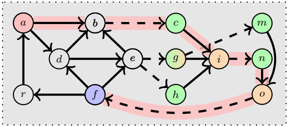
$F_2 = \{f, m, n\}$
$B_2 = \{b, e, o\}$



$F_3 = \{f, c, g, h\}$
$B_3 = \{b, e, i, g\}$

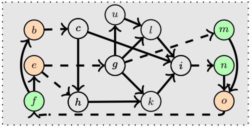
Query processing

Query: $a \rightsquigarrow f$



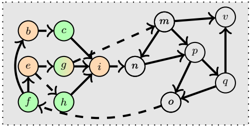
Forward List: $F_1 = \{c, g, h, m, n\}$

Backward List: $B_1 = \{g, i, o\}$



$F_2 = \{f, m, n\}$

$B_2 = \{b, e, o\}$



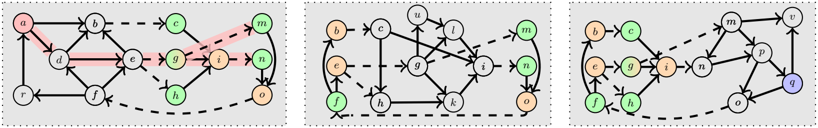
$F_3 = \{f, c, g, h\}$

$B_3 = \{b, e, i, g\}$

If s and t are local $\Rightarrow$ no communication, entire query can be processed locally

Query processing

Query: $a \rightsquigarrow q$



Forward List: $F_1 = \{c, g, h, m, n\}$

Backward List: $B_1 = \{g, i, o\}$

$F_2 = \{f, m, n\}$

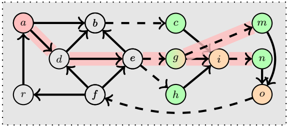
$B_2 = \{b, e, o\}$

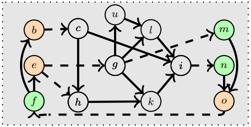
$F_3 = \{f, c, g, h\}$

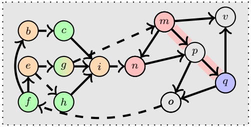
$B_3 = \{b, e, i, g\}$

Query processing

Query: $a \rightsquigarrow q$







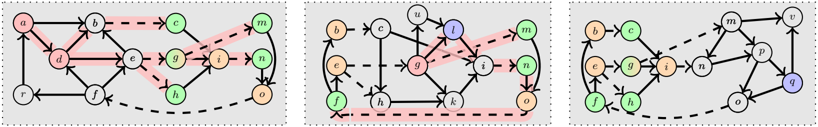
Forward List: $F_1 = \{c, g, h, m, n\}$ $F_2 = \{f, m, n\}$ $F_3 = \{f, c, g, h\}$

Backward List: $B_1 = \{g, i, o\}$ $B_2 = \{b, e, o\}$ $B_3 = \{b, e, i, g\}$

If s and t are non-local $\Rightarrow$ one step communication, involves atmost two partitions

Query processing

Query: $\{a, d, g\} \rightsquigarrow \{l, q\}$



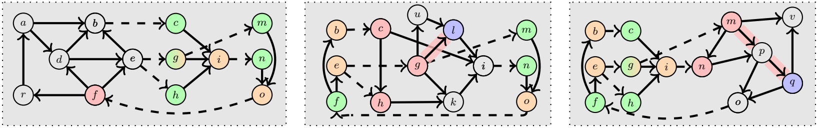
Forward List: $F_1 = \{c, g, h, m, n\}$ $\quad F_2 = \{f, m, n\}$ $\quad F_3 = \{f, c, g, h\}$

Backward List: $B_1 = \{g, i, o\}$ $\quad B_2 = \{b, e, o\}$ $\quad B_3 = \{b, e, i, g\}$

Step1: $\emptyset$ $\quad \{\langle g, l\rangle\}$ $\quad \emptyset$

Query processing

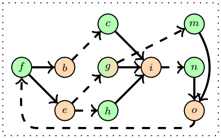
Query: $\{a, d, g\} \rightsquigarrow \{l, q\}$



Forward List:	$F_1 = \{c, g, h, m, n\}$	$F_2 = \{f, m, n\}$	$F_3 = \{f, c, g, h\}$
Backward List:	$B_1 = \{g, i, o\}$	$B_2 = \{b, e, o\}$	$B_3 = \{b, e, i, g\}$
Step1:	$\emptyset$	$\{\langle g, l\rangle\}$	$\emptyset$
Step2:	$\emptyset$	$\{\langle a, l\rangle, \langle d, l\rangle\}$	$\{\langle a, q\rangle, \langle d, q\rangle\ \langle g, q\rangle\}$

Scaling to Large Graphs

One of the main challenges for scalability is the size of boundary graph



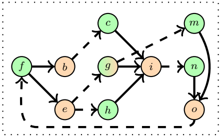
Boundary Graph

Size: $\mathcal{O}(\sum_{i=1}^{k}(|I_i| \cdot |O_i|) + |E_C|)$

Google: 43.6Mi $(8.5 \times |E|)$
LiveJ-20M: 861.4Mi $(43.07 \times |E|)$

Scaling to Large Graphs

One of the main challenges for scalability is the size of boundary graph

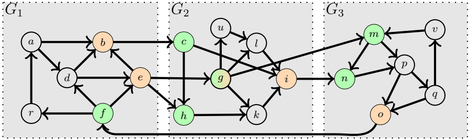


Boundary Graph

Size: $\mathcal{O}(\sum_{i=1}^{k}(|I_i| \cdot |O_i|) + |E_C|)$

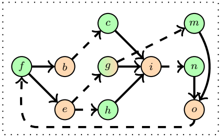
Google: 43.6Mi $(8.5 \times |E|)$
LiveJ-20M: 861.4Mi $(43.07 \times |E|)$

For a given partitioning $\mathcal{G} = \{G_1, G_2, G_3\}$, we reduce the boundary graph as follows.



Scaling to Large Graphs

One of the main challenges for scalability is the size of boundary graph

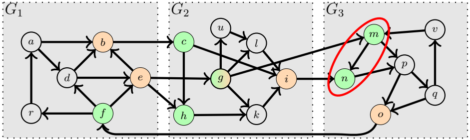


Boundary Graph

Size: $\mathcal{O}(\sum_{i=1}^{k}(|I_i| \cdot |O_i|) + |E_C|)$

Google: 43.6Mi $(8.5 \times |E|)$
LiveJ-20M: 861.4Mi $(43.07 \times |E|)$

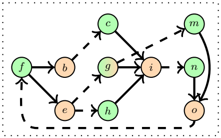
For a given partitioning $\mathcal{G} = \{G_1, G_2, G_3\}$, we reduce the boundary graph as follows.



in-boundaries m, n reach same set of vertices in $G_3 \Rightarrow$ forward equivalent

Scaling to Large Graphs

One of the main challenges for scalability is the size of boundary graph

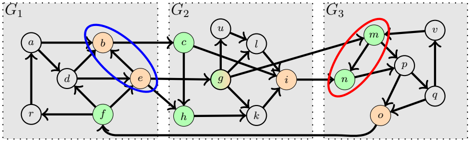


Boundary Graph

Size: $\mathcal{O}(\sum_{i=1}^{k}(|I_i| \cdot |O_i|) + |E_C|)$

Google: 43.6Mi $(8.5 \times |E|)$
LiveJ-20M: 861.4Mi $(43.07 \times |E|)$

For a given partitioning $\mathcal{G} = \{G_1, G_2, G_3\}$, we reduce the boundary graph as follows.

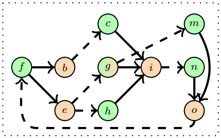


in-boundaries m, n reach same set of vertices in $G_3 \Rightarrow$ forward equivalent

out-boundaries b, e are reachable from same set of vertices in $G_1 \Rightarrow$ backward equivalent

Scaling to Large Graphs

One of the main challenges for scalability is the size of boundary graph

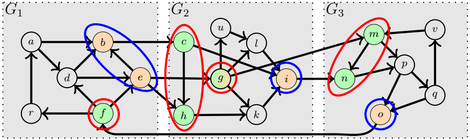


Boundary Graph

Size: $\mathcal{O}(\sum_{i=1}^{k}(|I_i| \cdot |O_i|) + |E_C|)$

Google: 43.6Mi ($8.5 \times |E|$)
LiveJ-20M: 861.4Mi ($43.07 \times |E|$)

For a given partitioning $\mathcal{G} = \{G_1, G_2, G_3\}$, we reduce the boundary graph as follows.

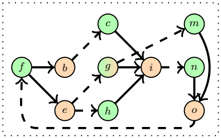


in-boundaries m, n reach same set of vertices in $G_3 \Rightarrow$ forward equivalent

out-boundaries b, e are reachable from same set of vertices in $G_1 \Rightarrow$ backward equivalent

Scaling to Large Graphs

One of the main challenges for scalability is the size of boundary graph

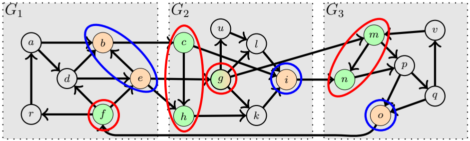


Boundary Graph

Size: $\mathcal{O}(\sum_{i=1}^{k}(|I_i| \cdot |O_i|) + |E_C|)$

Google: 43.6Mi $(8.5 \times |E|)$
LiveJ-20M: 861.4Mi $(43.07 \times |E|)$

For a given partitioning $\mathcal{G} = \{G_1, G_2, G_3\}$, we reduce the boundary graph as follows.

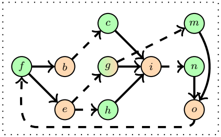


$I_1 = \{f\}$ $\quad$ $I_2 = \{c, g, h\}$ $\quad$ $I_3 = \{m, n\}$

$O_1 = \{b, e\}$ $\quad$ $O_2 = \{i, g\}$ $\quad$ $O_3 = \{o\}$

Scaling to Large Graphs

One of the main challenges for scalability is the size of boundary graph



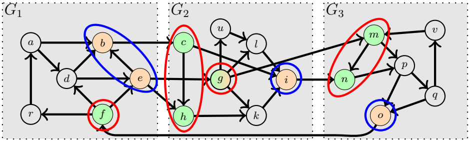
Boundary Graph

Size: $\mathcal{O}(\sum_{i=1}^{k}(|I_i| \cdot |O_i|) + |E_C|)$

Google: 43.6Mi $(8.5 \times |E|)$

LiveJ-20M: 861.4Mi $(43.07 \times |E|)$

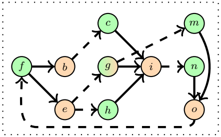
For a given partitioning $\mathcal{G} = \{G_1, G_2, G_3\}$, we reduce the boundary graph as follows.



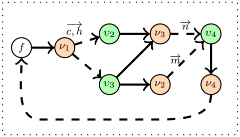
$I_1 = \{f\}$	$I_2 = \{c, g, h\}$	$I_3 = \{m, n\}$	
$O_1 = \{b, e\}$	$O_2 = \{i, g\}$	$O_3 = \{o\}$	
$\upsilon_1 = \{f\}$	$\upsilon_2 = \{c, h\}, \upsilon_3 = \{g\}$	$\upsilon_4 = \{m, n\}$	Fwd eq sets (in-virtual nodes)
$\nu_1 = \{b, e\}$	$\nu_2 = \{i\}, \nu_3 = \{g\}$	$\nu_4 = \{o\}$	Bwd eq sets (out-virtual nodes)

Scaling to Large Graphs

One of the main challenges for scalability is the size of boundary graph



Boundary Graph



Compressed Boundary Graph

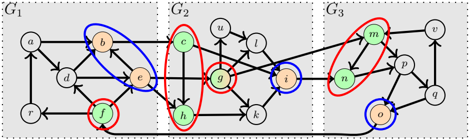
Size: $\mathcal{O}(\sum_{i=1}^{k}(|N_i| \cdot |\Upsilon_i|) + |E'_C|)$

N_i: set of in-virtual nodes at i

Υ_i: set of out-virtual nodes at i

$N_i \leq I_i$, $\Upsilon_i \leq O_i$, and $|E'_C| \leq |E_C|$

For a given partitioning $\mathcal{G} = \{G_1, G_2, G_3\}$, we reduce the boundary graph as follows.



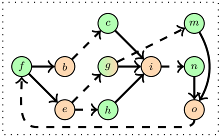
$I_1 = \{f\}$	$I_2 = \{c, g, h\}$	$I_3 = \{m, n\}$
$O_1 = \{b, e\}$	$O_2 = \{i, g\}$	$O_3 = \{o\}$

$\upsilon_1 = \{f\}$ $\quad \upsilon_2 = \{c, h\}, \upsilon_3 = \{g\}$ $\quad \upsilon_4 = \{m, n\}$ Fwd eq sets (in-virtual nodes)

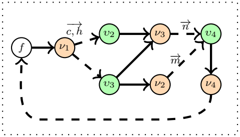
$\nu_1 = \{b, e\}$ $\quad \nu_2 = \{i\}, \nu_3 = \{g\}$ $\quad \nu_4 = \{o\}$ Bwd eq sets (out-virtual nodes)

Scaling to Large Graphs

One of the main challenges for scalability is the size of boundary graph



Boundary Graph

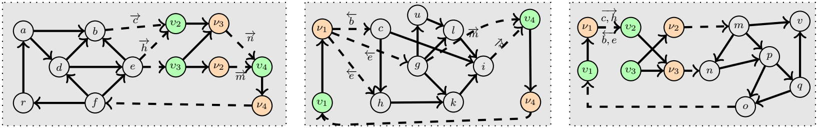


Compressed Boundary Graph

Size: $\mathcal{O}(\sum_{i=1}^{k}(|N_i| \cdot |\Upsilon_i|) + |E'_C|)$

N_i: set of in-virtual nodes at i
Υ_i: set of out-virtual nodes at i
$N_i \leq I_i$, $\Upsilon_i \leq O_i$, and $|E'_C| \leq |E_C|$

For a given partitioning $\mathcal{G} = \{G_1, G_2, G_3\}$, we reduce the boundary graph as follows.



lower communication costs, scalable to very large graphs

Evaluation

Datasets:

Small Graphs	|V|	|E|	Large Graphs	|V|	|E|
Amazon	403,394	3,387,388	LiveJ-68M	4,847,571	68,993,773
BerkStan	685,230	7,600,595	Twitter-1.4B	41,652,230	1,468,364,884
Google	875,713	5,105,039	Freebase-500M	97,290,357	499,982,284
NotreDame	325,729	1,497,134	Freebase-1B	156,595,723	999,965,047
Stanford	281,903	2,312,497	LUBM-500M	115,561,430	500,002,176
LiveJ-20M	2,545,981	20,000,000	LUBM-1B	222,213,904	961,394,352

Graph datasets

Setup: Master:1, Slaves:9, Memory:64 GB

Evaluation

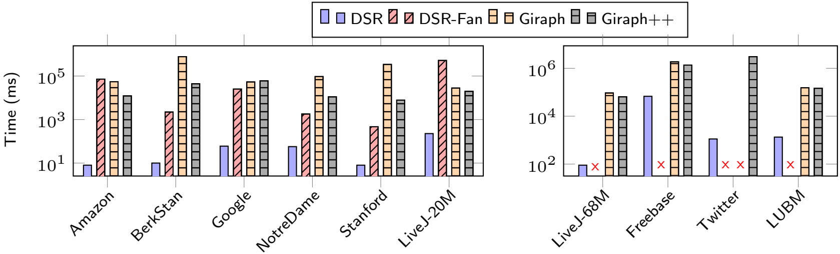
Datasets:

Small Graphs	|V|	|E|	Large Graphs	|V|	|E|
Amazon	403,394	3,387,388	LiveJ-68M	4,847,571	68,993,773
BerkStan	685,230	7,600,595	Twitter-1.4B	41,652,230	1,468,364,884
Google	875,713	5,105,039	Freebase-500M	97,290,357	499,982,284
NotreDame	325,729	1,497,134	Freebase-1B	156,595,723	999,965,047
Stanford	281,903	2,312,497	LUBM-500M	115,561,430	500,002,176
LiveJ-20M	2,545,981	20,000,000	LUBM-1B	222,213,904	961,394,352

Graph datasets

Setup: Master:1, Slaves:9, Memory:64 GB

Performance:



DSR is orders of magnitude faster than Giraph, Giraph++, and DSR-Fan

Evaluation

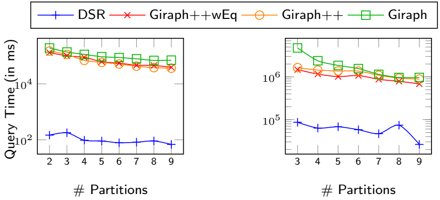
Datasets:

Small Graphs	|V|	|E|	Large Graphs	|V|	|E|
Amazon	403,394	3,387,388	LiveJ-68M	4,847,571	68,993,773
BerkStan	685,230	7,600,595	Twitter-1.4B	41,652,230	1,468,364,884
Google	875,713	5,105,039	Freebase-500M	97,290,357	499,982,284
NotreDame	325,729	1,497,134	Freebase-1B	156,595,723	999,965,047
Stanford	281,903	2,312,497	LUBM-500M	115,561,430	500,002,176
LiveJ-20M	2,545,981	20,000,000	LUBM-1B	222,213,904	961,394,352

Graph datasets

Setup: Master:1, Slaves:9, Memory:64 GB

Scalability:



LiveJ-68 Freebase-1B

Evaluation

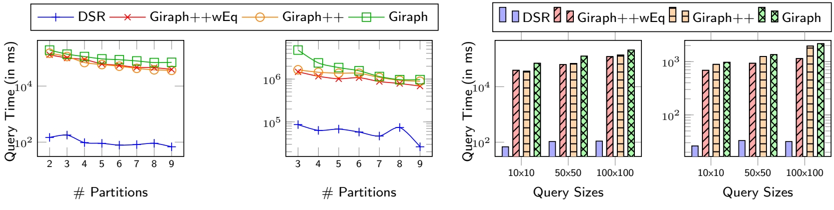
Datasets:

Small Graphs	|V|	|E|	Large Graphs	|V|	|E|
Amazon	403,394	3,387,388	LiveJ-68M	4,847,571	68,993,773
BerkStan	685,230	7,600,595	Twitter-1.4B	41,652,230	1,468,364,884
Google	875,713	5,105,039	Freebase-500M	97,290,357	499,982,284
NotreDame	325,729	1,497,134	Freebase-1B	156,595,723	999,965,047
Stanford	281,903	2,312,497	LUBM-500M	115,561,430	500,002,176
LiveJ-20M	2,545,981	20,000,000	LUBM-1B	222,213,904	961,394,352

Graph datasets

Setup: Master:1, Slaves:9, Memory:64 GB

Scalability:



LiveJ-68 Freebase-1B LiveJ-68 Freebase-1B

Conclusions

- We introduced and investigated the Distributed Set Reachability problem

Conclusions

- We introduced and investigated the Distributed Set Reachability problem
- Precomputation and the ability to use local indexes significantly improved the performance without hurting scalability

Conclusions

- We introduced and investigated the Distributed Set Reachability problem
- Precomputation and the ability to use local indexes significantly improved the performance without hurting scalability
- Compression via equivalent sets grouping helps in scaling to large graphs

Conclusions

- We introduced and investigated the Distributed Set Reachability problem
- Precomputation and the ability to use local indexes significantly improved the performance without hurting scalability
- Compression via equivalent sets grouping helps in scaling to large graphs
- Extensible to dynamic graphs (from our preliminary tests)

Conclusions

- We introduced and investigated the Distributed Set Reachability problem
- Precomputation and the ability to use local indexes significantly improved the performance without hurting scalability
- Compression via equivalent sets grouping helps in scaling to large graphs
- Extensible to dynamic graphs (from our preliminary tests)

Thank you for your attention

References I

- Wenfei Fan, Xin Wang, and Yinghui Wu, **Performance guarantees for distributed reachability queries**, PVLDB **5** (2012), no. 11, 1304–1315.
- Sidan Gao and Kemafor Anyanwu, **PrefixSolve: efficiently solving multi-source multi-destination path queries on RDF graphs by sharing suffix computations**, WWW, 2013, pp. 423–434.
- Stephan Seufert, Avishek Anand, Srikanta J. Bedathur, and Gerhard Weikum, **FERRARI: flexible and efficient reachability range assignment for graph indexing**, ICDE, 2013, pp. 1009–1020.
- Manuel Then, Moritz Kaufmann, Fernando Chirigati, Tuan-Anh Hoang-Vu, Kien Pham, Alfons Kemper, Thomas Neumann, and Huy T. Vo, **The more the merrier: Efficient multi-source graph traversal**, PVLDB **8** (2014), no. 4, 449–460.
- Hilmi Yildirim, Vineet Chaoji, and Mohammed J. Zaki, **GRAIL: Scalable Reachability Index for Large Graphs**, PVLDB **3** (2010), no. 1-2, 276–284.

Questions?

For more details, please visit my poster: 84