

Recursive Computation of Regions and Connectivity in Networks



Mengmeng Liu, Nicholas E. Taylor, Wenchao Zhou, Zachary G. Ives, Boon Thau Loo
Computer and Information Science Department, University of Pennsylvania

Motivation

- Declarative queries are revisited in both declarative networking and sensor networks scenarios.
- How to maintain a *view* over **dynamic** network state where the view is frequently **distributed**, **recursive** and may contain **aggregations**?
- Problem: Incremental Recursive View Maintenance** in the presence of distributed *streams* of update tuples (insertions and deletions).

Example Queries

Q1 (Network reachability query)

(r1) $reachable(x,y) :- link(x,y).$

(r2) $reachable(x,y) :- link(x,z), reachable(z,y).$

It finds all pairs of nodes that can reach each other.

Q2 (Sensing contiguous regions)

(r1) $activeRegion(rid,x) :- sensor(x,posx), isTriggered(x), mainSensorInRegion(rid, x).$

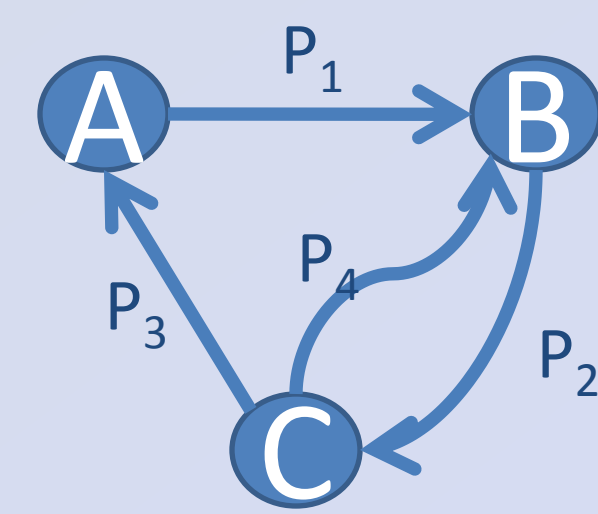
(r2) $activeRegion(rid,y) :- sensor(x, posx), sensor(y, posy), isTriggered(x), activeRegion(rid, x), distance(posx, posy) < k.$

It finds contiguous (within k meters) triggered nodes and adds them to regions.

Existing Solutions

- DRed [Gupta+ 93]: recursive view maintenance using delta rules
 - Insertions: incremental maintenance
 - Deletions: over-delete and re-derive
- Drawbacks:
 - centralized mechanism, not scalable to hundreds of nodes
 - bad performance, especially when deletions are frequent
 - worse than naïve method in some cases
- What we want:
 - efficiency
 - scalability
 - low communication overhead
 - generality

Our Approach: Absorption Provenance



We annotate tuple with Boolean expressions such that: the tuple is in the view iff the Boolean expression evaluates to **true**.

Insertions: record all the derivations of each tuple in an absorbed Boolean expression.

Deletions: zero out the provenance tokens in each Boolean expression.

Link			Reachable		
tuple	location	provenance	tuple	location	provenance
(A,B)	A	p_1	(A,A)	A	$p_1 p_2 p_3$
(B,C)	B	p_2	(A,B)	A	p_1
(C,A)	C	p_3	(A,C)	A	$p_1 p_2$
(C,B)	C	p_4	(B,A)	B	$p_2 p_3$
			(B,B)	B	$p_2 p_4 + p_1 p_2 p_3$
			(B,C)	B	p_2
			(C,A)	C	p_3
			(C,B)	C	$p_4 + p_1 p_3$
			(C,C)	C	$p_2 p_4 + p_1 p_2 p_3$
-(C,B)	C	p_4	-(B,B)	B	$-p_2 p_4$
			-(C,B)	C	$-p_4$
			-(C,C)	C	$-p_2 p_4$

Rules:

$\sigma_\theta(R)$: If tuple t in R satisfies θ , annotate t with $P(t)$.

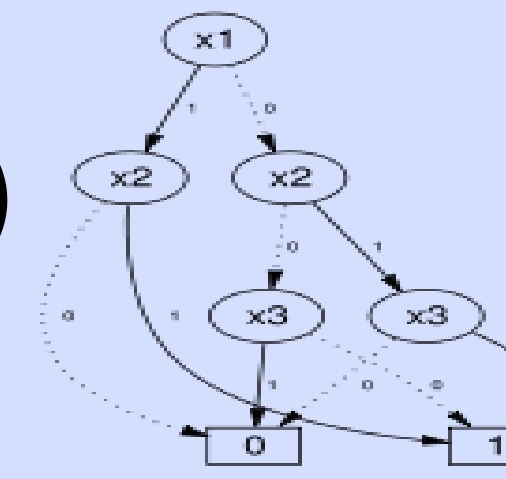
$R_1 \bowtie R_2$: For each tuple t_1 in R_1 and tuple t_2 in R_2 , annotate the output join tuple with $P(t_1) \wedge P(t_2)$

$R_1 \cup R_2$: For each output tuple t , annotate it with $P(t_1) \vee P(t_2)$.

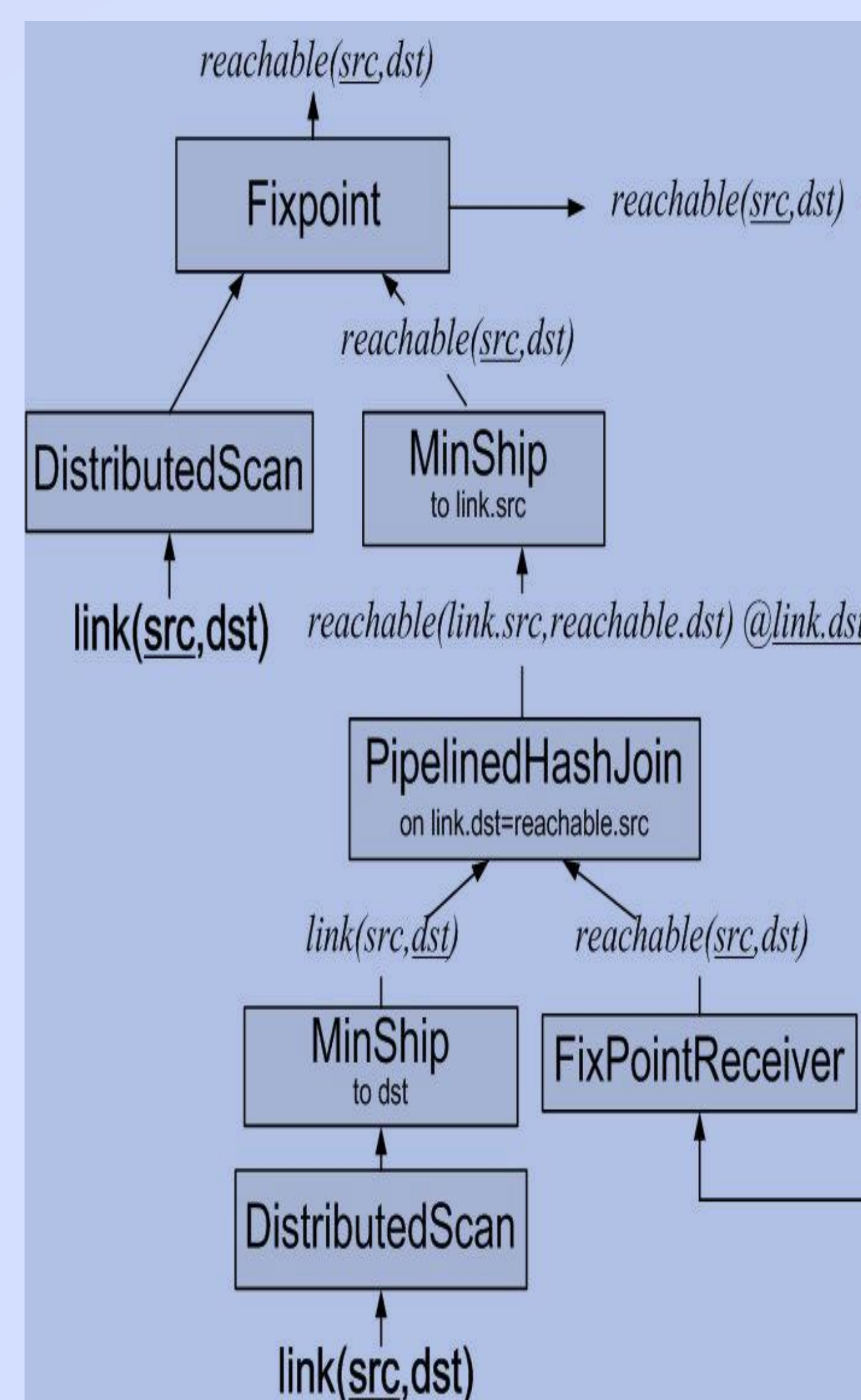
$\pi_A(R)$: Given tuples t_1, t_2, \dots, t_n that project to the same tuple t , annotate t with $P(t_1) \vee P(t_2) \dots \vee P(t_n)$.

Absorption Law: $a \wedge (a \vee b) \equiv a \vee (a \wedge b) \equiv a$

Implementation: Binary Decision Diagrams(BDD)



Distributed Optimization Techniques



Fixpoint

We reach a **fixpoint** when we can no longer derive any new results that affect the Absorption Provenance of any tuple!

MinShip

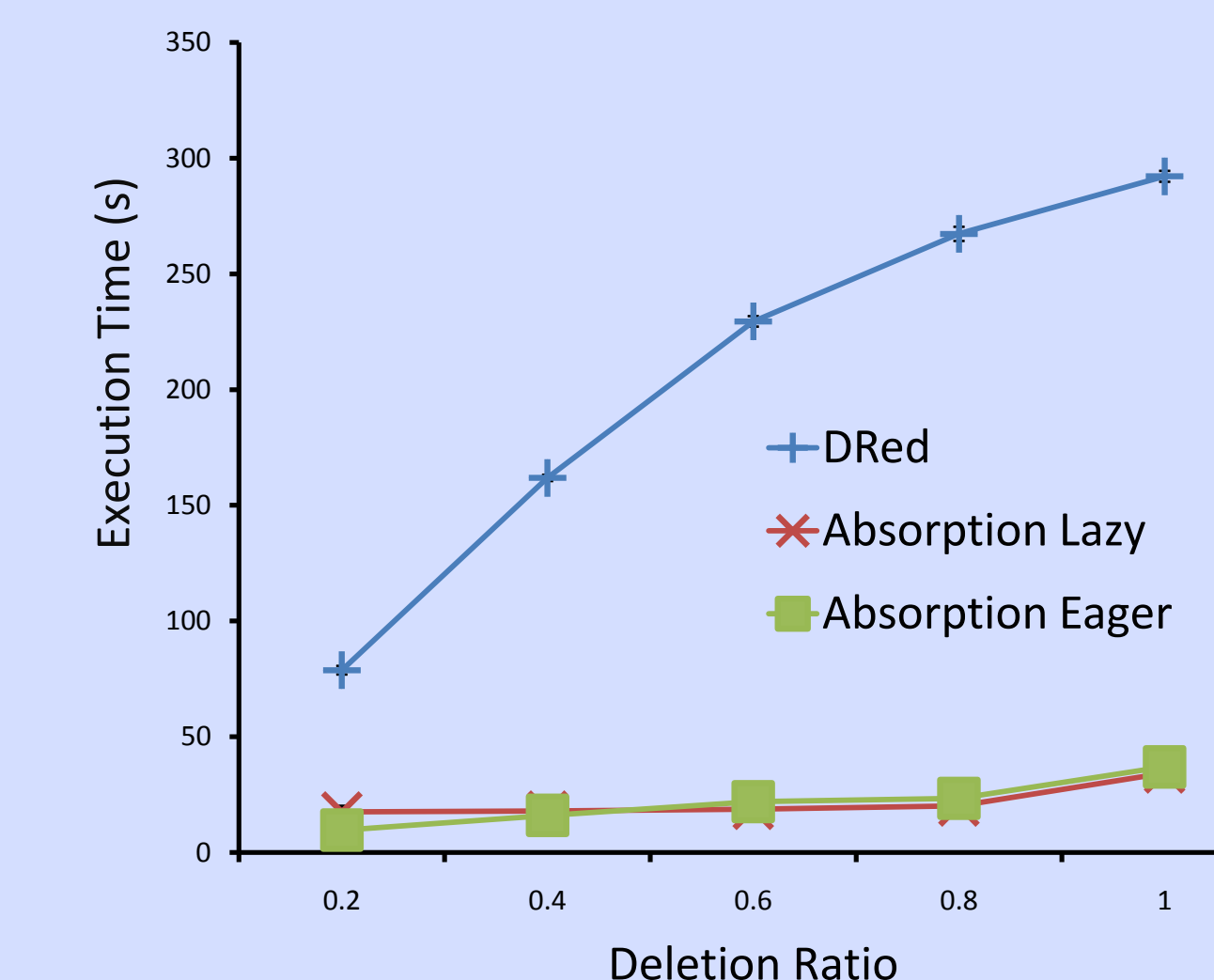
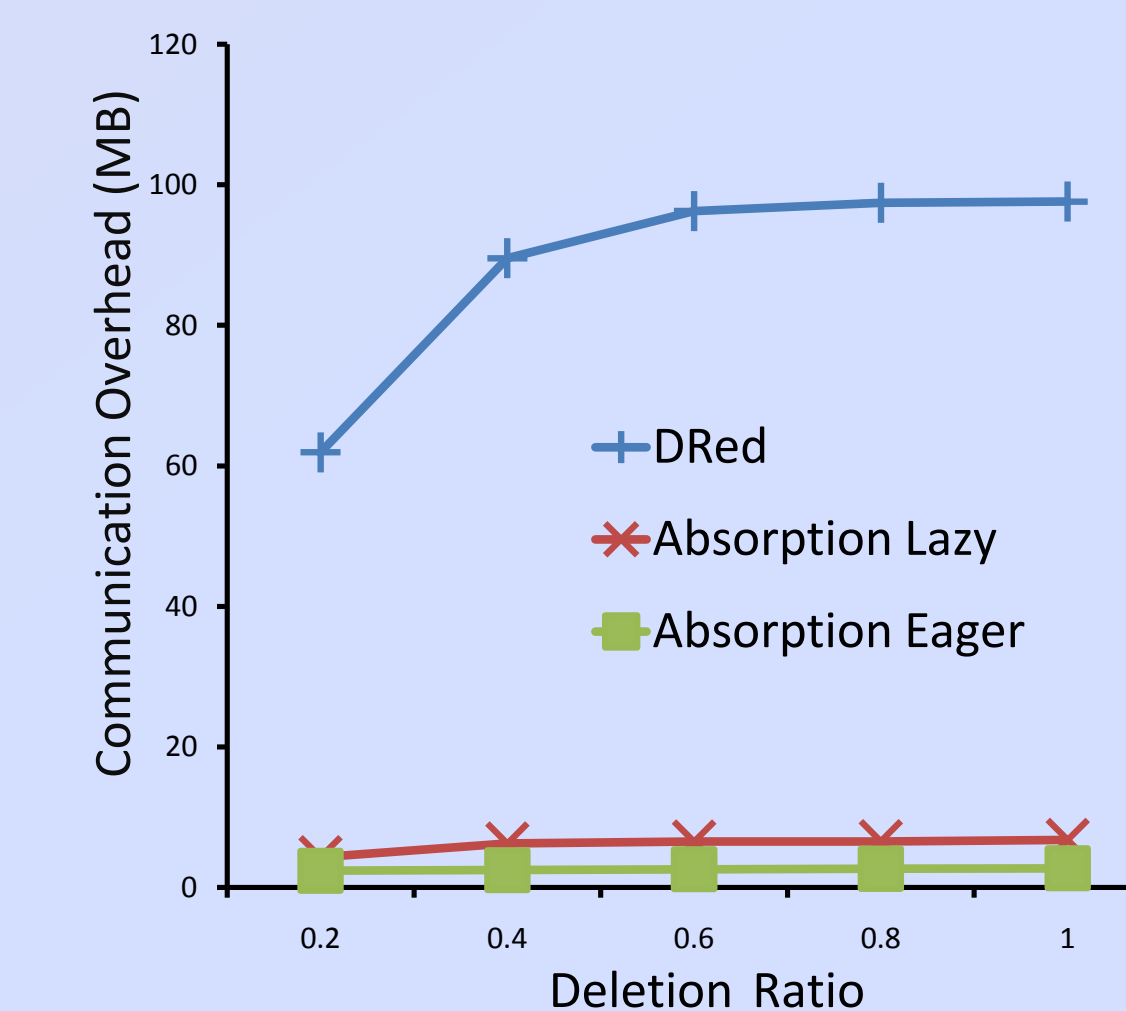
We only propagate the first derivation, and buffer the subsequent derivation. If the buffer size is large enough, this is called **Lazy Propagation**.

Aggregate Selection

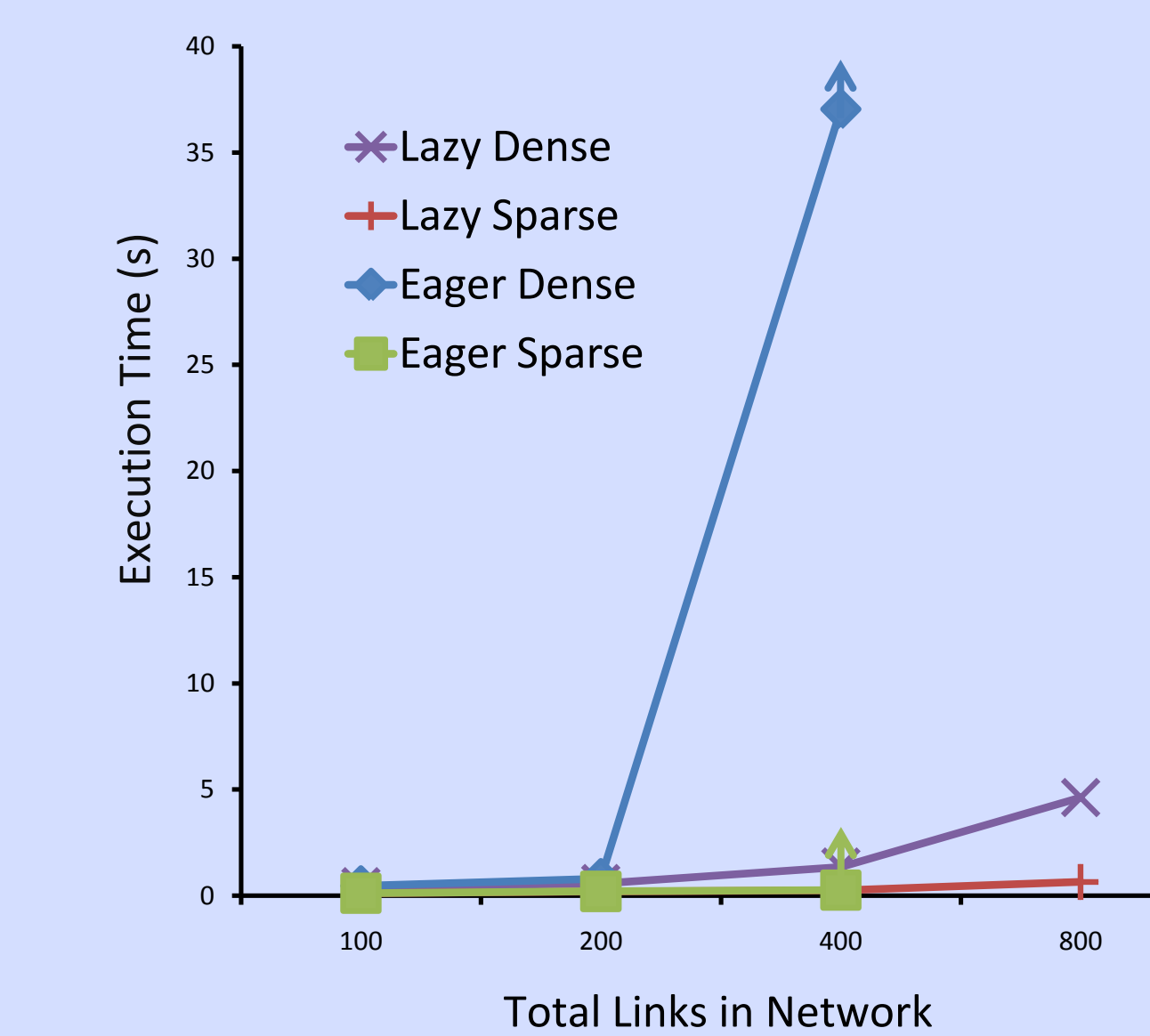
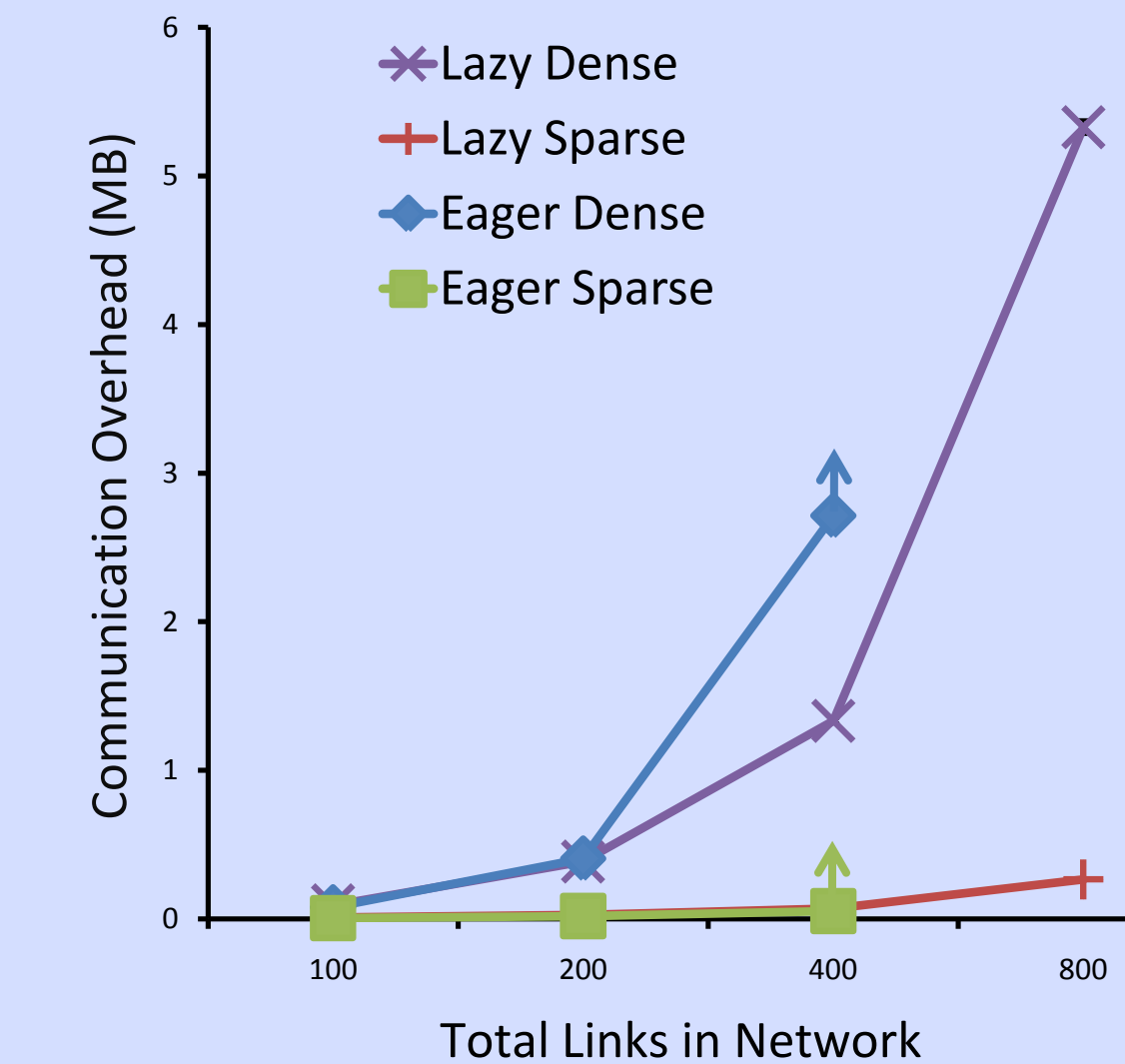
We always push down selections before aggregations, and design a provenance-aware module to handle insertions as well as deletions.

Performance Evaluation

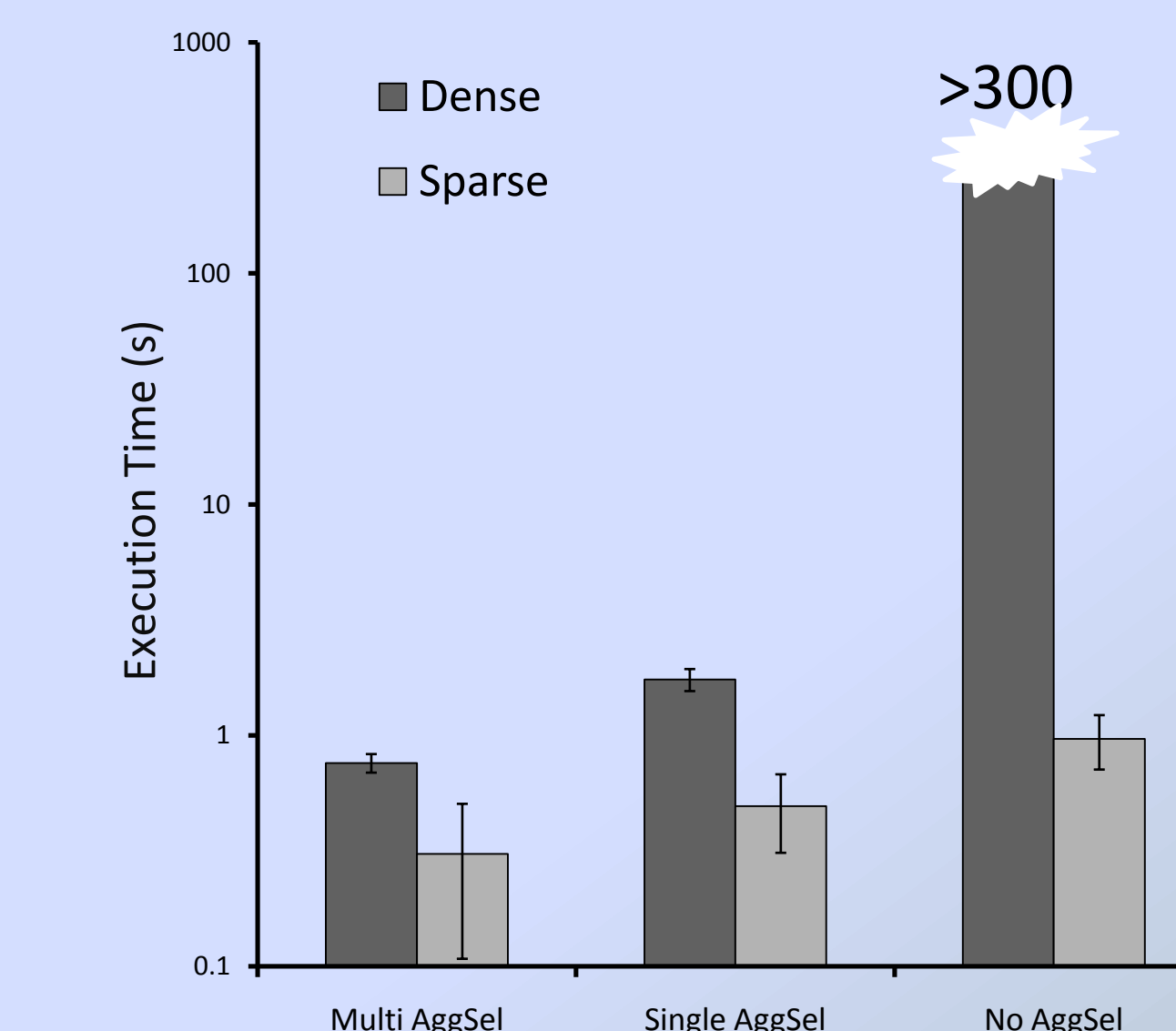
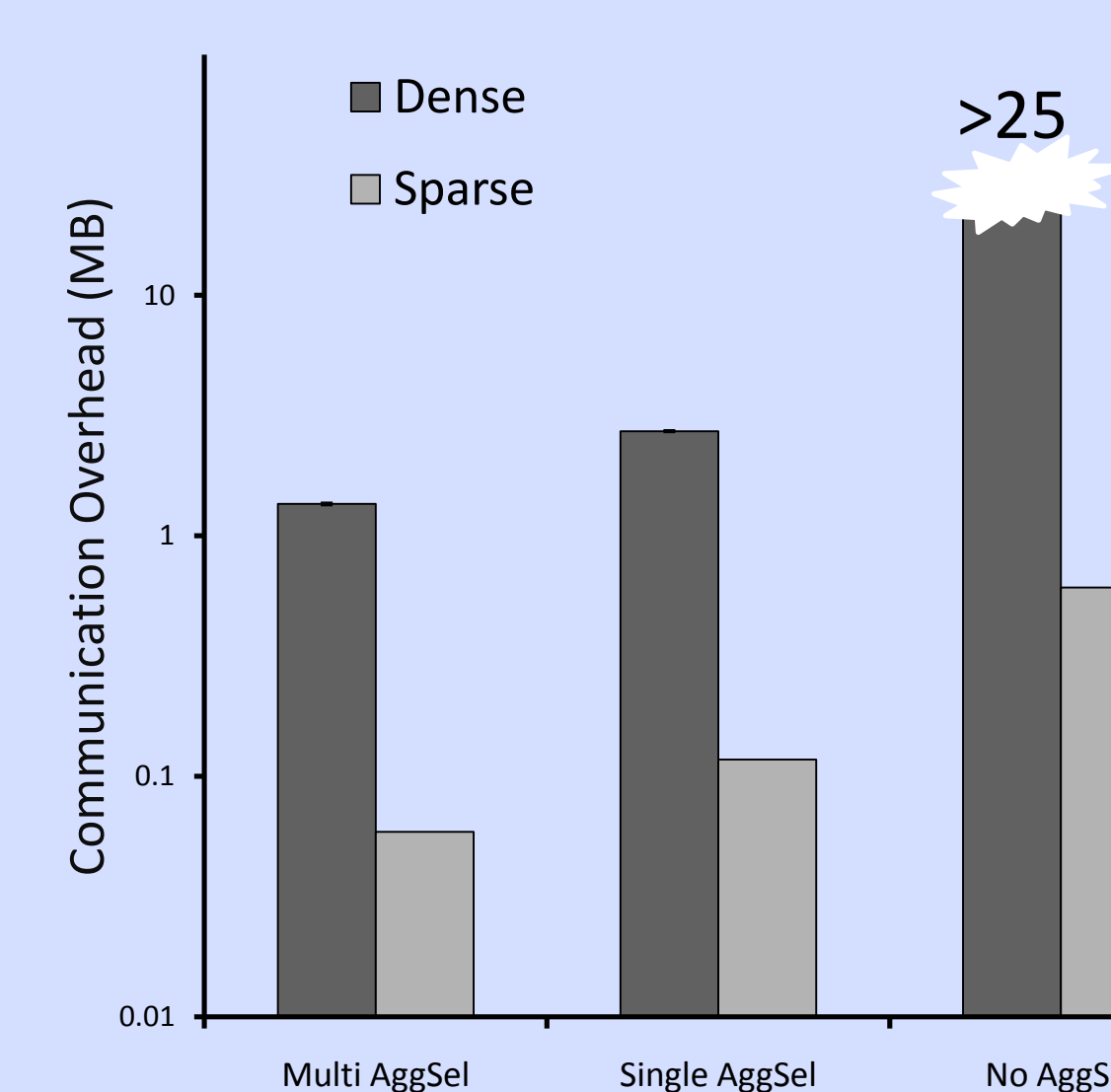
Incremental Maintenance



Scaling data sets



Aggregate Selection



Publications

Mengmeng Liu, Nicholas E. Taylor, Wenchao Zhou, Zachary G. Ives, Boon Thau Loo, "Recursive Computation of Regions and Connectivity in Networks" (full paper), to appear in ICDE 2009, Shanghai, China.