Wander Join: Online Aggregation via Random Walks

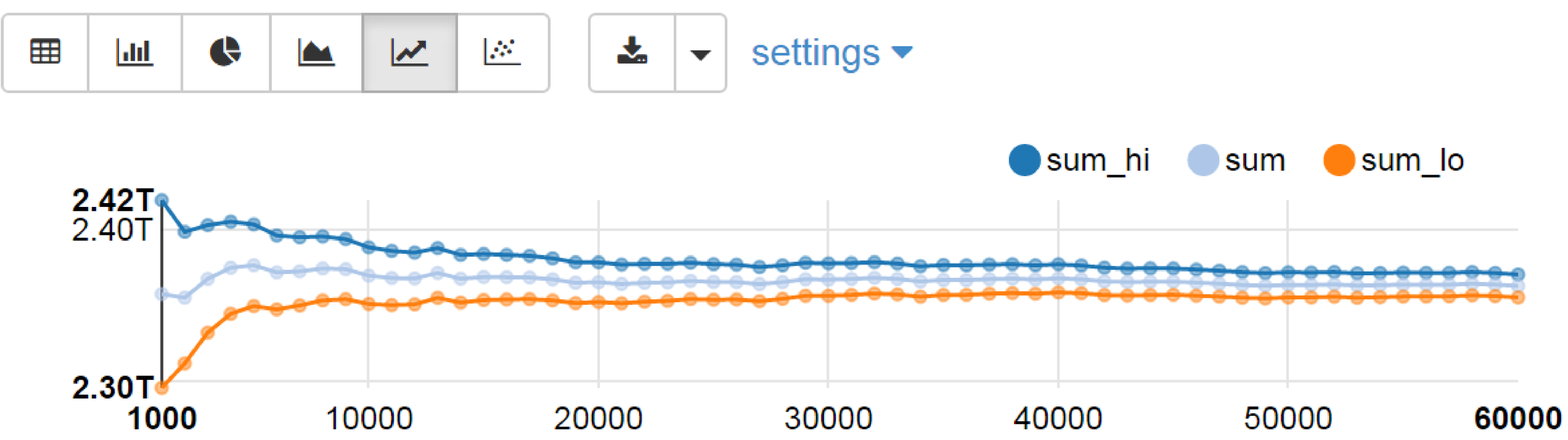
Feifei Li (U. Utah), *Bin Wu* (HKUST), *Ke Yi* (HKUST), *Zhuoyue Zhao* (SJTU)

Abstract

Joins are expensive, and online aggregation over joins was proposed to mitigate the cost, which offers users a nice and flexible tradeoff between query efficiency and accuracy in a continuous, online fashion. We introduce a new approach, *wander join*, to the online aggregation problem by performing random walks over the underlying join graph.

Online Aggregation

```
SELECT ONLINE SUM(l_extendedprice * (1 - l_discount))
FROM customer, orders, lineitem
WHERE   c_mktsegment = 'BUILDING'
    AND c_custkey = o_custkey
    AND l_orderkey = o_orderkey
WITHTIME 60000 CONFIDENCE 95 REPORTINTERVAL 1000;
```



Wander Join

Query
Total revenue of all orders from customers in China.

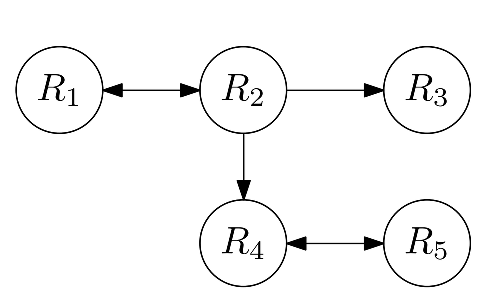
Nation	CID
US	1
US	2
China	3
UK	4
China	5
US	6
China	7
UK	8
Japan	9
UK	10

BuyerID	OrderID
4	1
3	2
1	3
5	4
5	5
5	6
3	7
5	8
3	9

OrderID	ItemID	Price
4	301	$2100
2	304	$100
3	201	$300
4	306	$500
3	401	$230
1	101	$800
2	201	$300
5	101	$200
4	301	$100

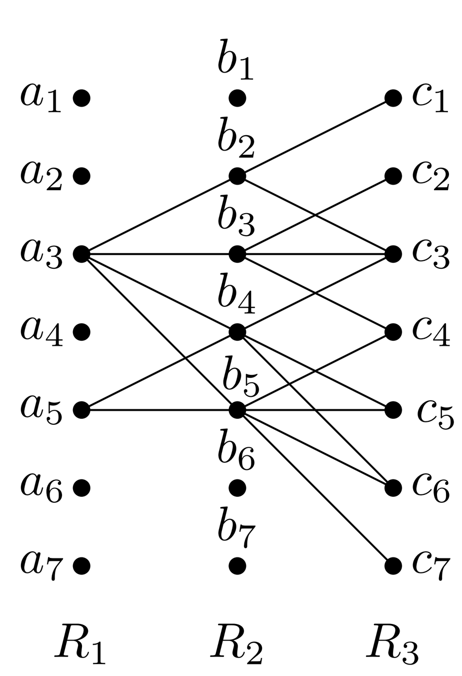
Unbiased estimator: $\frac{\$500}{\text{sampling prob.}} = \frac{\$500}{1/3 \cdot 1/4 \cdot 1/3}$

Walk Plan Optimizer



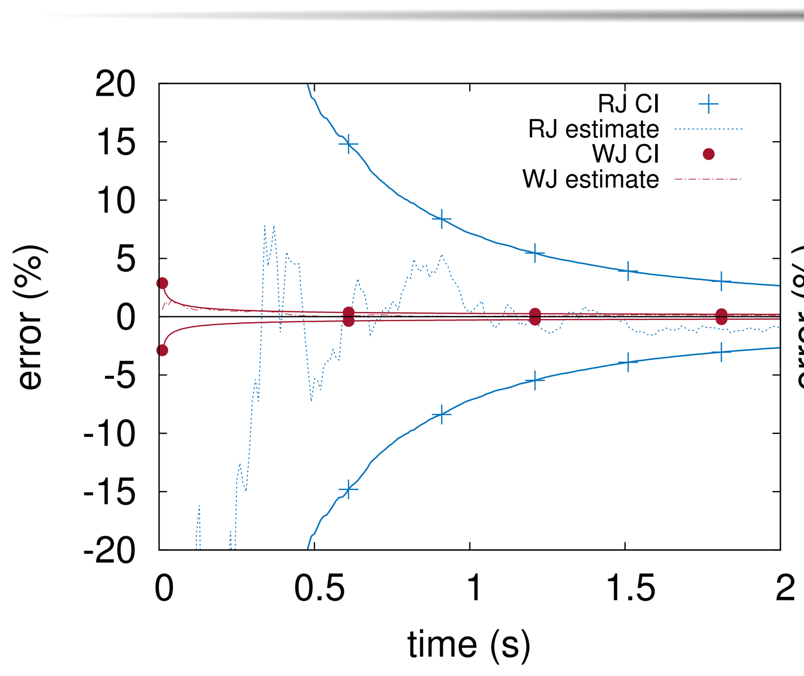
R_1, R_2, R_3, R_4, R_5
R_1, R_2, R_4, R_3, R_5
R_1, R_2, R_4, R_5, R_3
R_2, R_1, R_3, R_4, R_5
R_2, R_1, R_4, R_3, R_5
R_2, R_1, R_4, R_5, R_3
R_2, R_3, R_1, R_4, R_5
R_2, R_3, R_4, R_1, R_5
R_2, R_3, R_4, R_5, R_1
R_2, R_4, R_1, R_3, R_5
R_2, R_4, R_1, R_5, R_3
R_2, R_4, R_3, R_1, R_5
R_2, R_4, R_3, R_5, R_1
R_2, R_4, R_5, R_1, R_3
R_2, R_4, R_5, R_3, R_1

All possible walk plans.

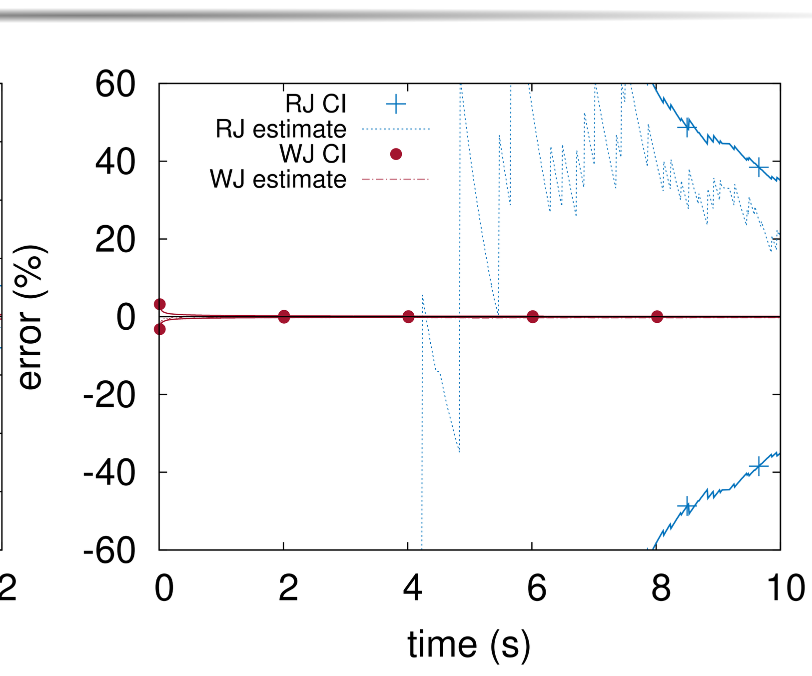


Structure of the join data graph has a significant impact on the performance of different walk plans.

Standalone implementation.

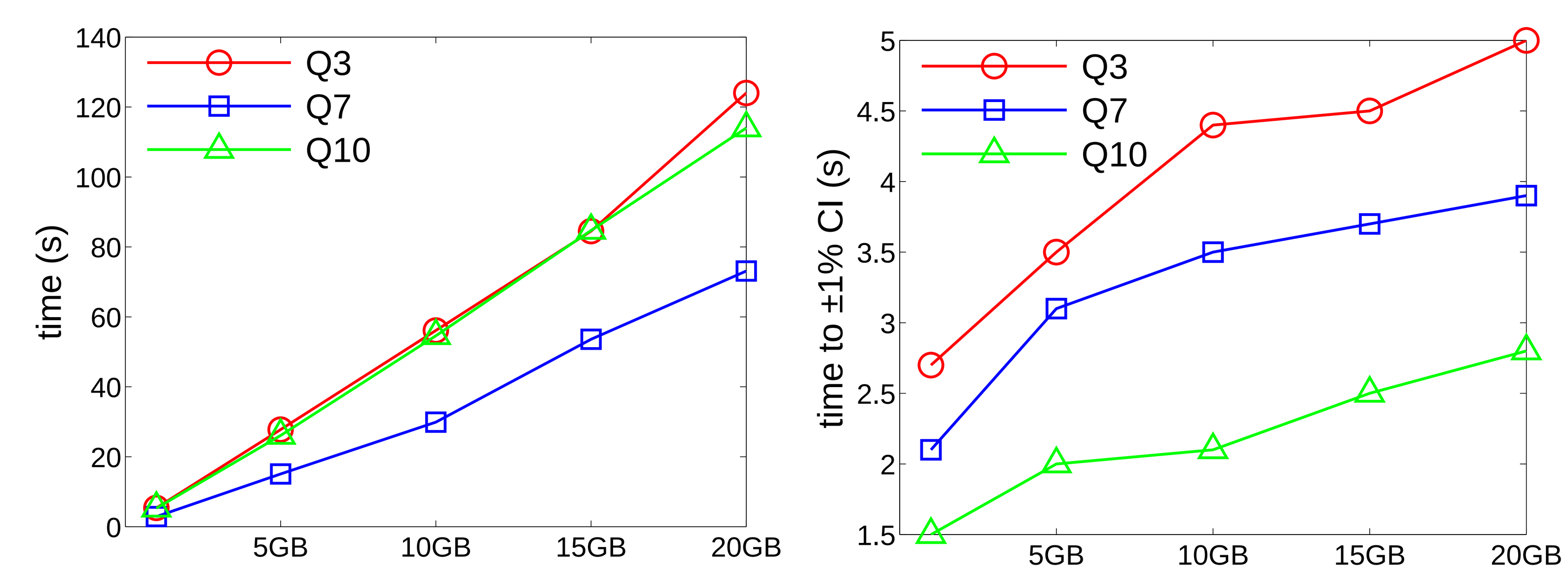


Q3



Q7

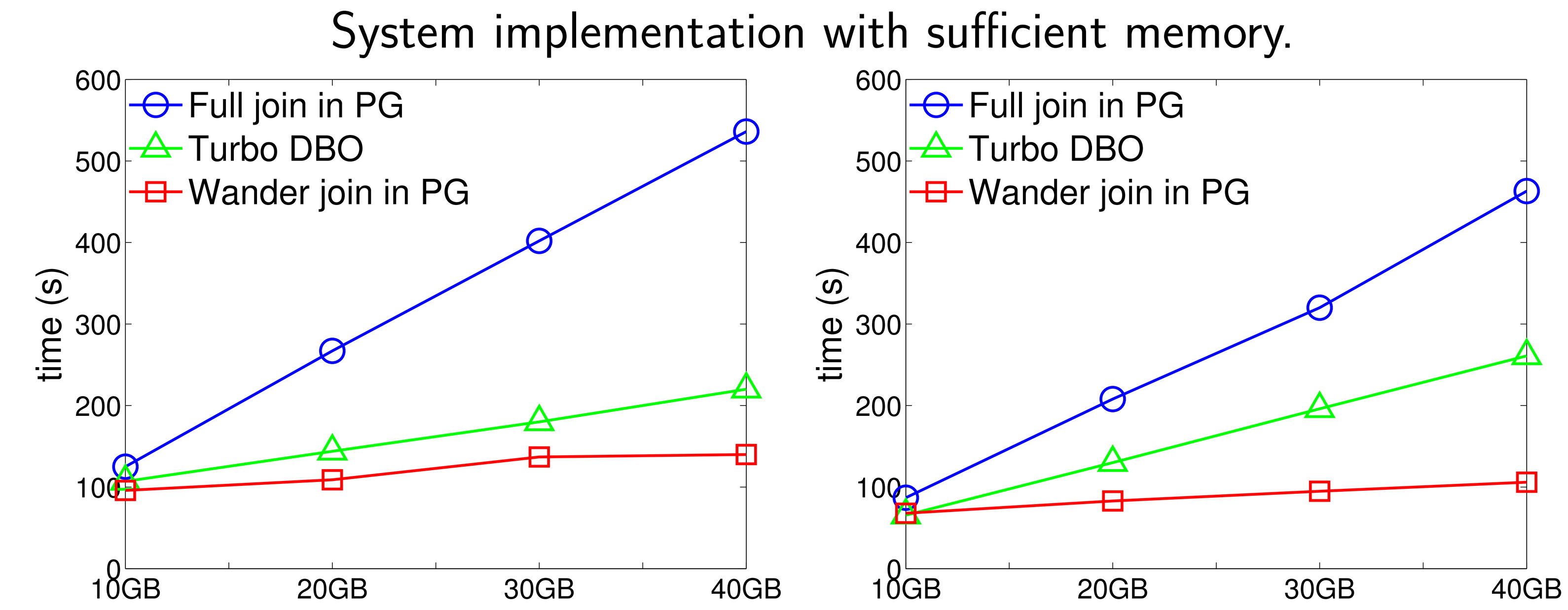
System implementation



PostgreSQL full join

Wander join in PostgreSQL

System implementation with sufficient memory.



Q3

Q7

System implementation with limited memory.

Comparison

Wander join		Ripple join
Independent but non-uniform	Sampling methodology	Uniform but non-independent
Needed	Index	Not needed
Easy $O(n)$ Time	Confidence interval computation	Complicated $O(n^k)$ Time k : # of tables
~3 seconds ~100 seconds	Convergence time(20 GB) Internal memory (target CI 1%) External memory (target CI 5%)	~50 seconds ~100 seconds
Logarithmic	Scalability	Linear
Parser Join processor Index lookup	Component affected	Almost everything
PostgreSQL (finished) Oracle (in progress) SparkSQL (in progress)	System implementation	Informix (CONTROL project) DBO

Accuracy achieved in 1/10 of System X's full running time.

	size(GB)	sufficient memory			limited memory		
		System X[1]	PG+WJ[4]		System X	PG+WJ	
			CI[2]	AE[3]		CI	AE
Q3	10	32.24	1.18	0.09	107.27	15.9	7.8
	20	74.29	0.78	0.43	249.94	11.1	4.3
	30	65.17	0.84	0.40	428.39	9.6	4.5
	40	90.23	0.76	0.26	707.04	8.1	4.7
Q10	10	40.43	0.75	0.01	146.57	13.7	1.2
	20	98.96	0.47	0.02	326.67	8.7	2.1
	30	109.19	0.46	0.05	697.06	6.9	1.3
	40	138.87	0.42	0.06	829.97	5.2	0.5

Accuracy achieved in $1/10$ of System X's running time for computing the full join.

[1] System X: full join time on System X (seconds).
[2] CI: half width of the confidence interval (%).
[3] AE: actual error (%).
[4] PG+WJ: Our version of PostgreSQL with Wander Join implemented inside the PostgreSQL engine.