Walking in the Cloud: Parallel SimRank at Scale

Zhenguo Li[1], Yixiang Fang[2], Qin Liu[3], Jiefeng Cheng[1], Reynold Cheng[2], John C.S. Lui[3]

[1]Huawei Noah's Ark Lab, [2]HKU, [3]CUHK

To appear in PVLDB'16

SimRank [1]

❑ **Graph data grows rapidly**
1. Internet of Things
2. World Wide Web

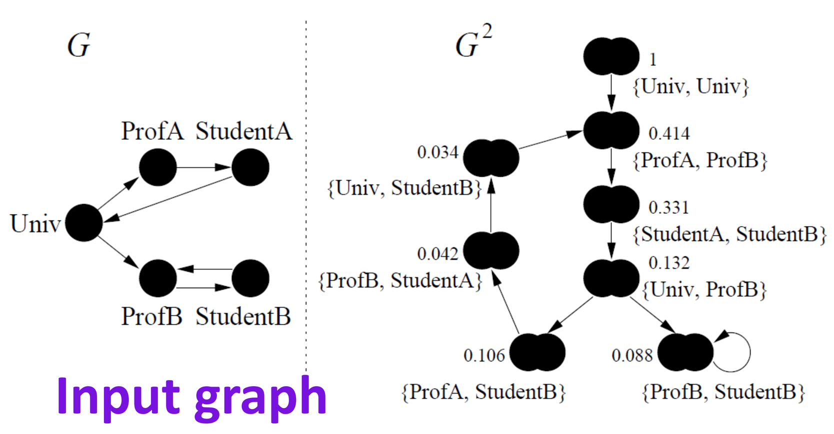
❑ **Similarity is fundamental**
1. Information retrieval
2. Recommender system
3. Churn prediction

❑ **SimRank - two objects are similar if referenced by similar objects**

$$s(i,j)=\begin{cases}1, & i=j\\ \dfrac{c}{|In(i)||In(j)|}\sum_{i'\in In(i),\, j'\in In(j)} s(i',j'), & i\neq j\end{cases}$$

s(i,j): similarity of nodes *i* and *j*
In(i): in-neighbors of *i*
c: decay factor, $0<c<1$

Similarity Propagation



- It captures human perception of similarity
- It outperforms other similarity measures, such as co-citation

❑ **Three fundamental queries**
1. Single-pair query – return similarity of two nodes
2. Single-source query – return similarity of every node to a node
3. All-pair query – return similarity between every two nodes

❑ **Challenges in SimRank computation**
1. High complexity: $O(n^3)$ time, $O(n^2)$ space
2. Heavy computational dependency (hard to be parallelized)
3. Not allow querying similarities individually

CloudWalker – Big SimRank, instant response

❑ **Contribution**
1. Enable parallel SimRank computation
2. Test on the largest graph, clue-web(|V|=1B, |E|=43B)

❑ **Problem**

SimRank Decomposition $S=cP^{\top}DP+D$

P: the transition matrix on graph

D: the diagonal correction matrix to be estimated

$$S=D+cP^{\top}DP+c^2P^{2\top}DP^2+\cdots$$

1. how to compute *D* for big graph ?
2. how to query efficiently given *D* ?

❑ **Offline indexing** $x=[D_{11},D_{22},\cdots,D_{nn}]^{\top}$

1. **Key observation:** self-similarity is 1.0

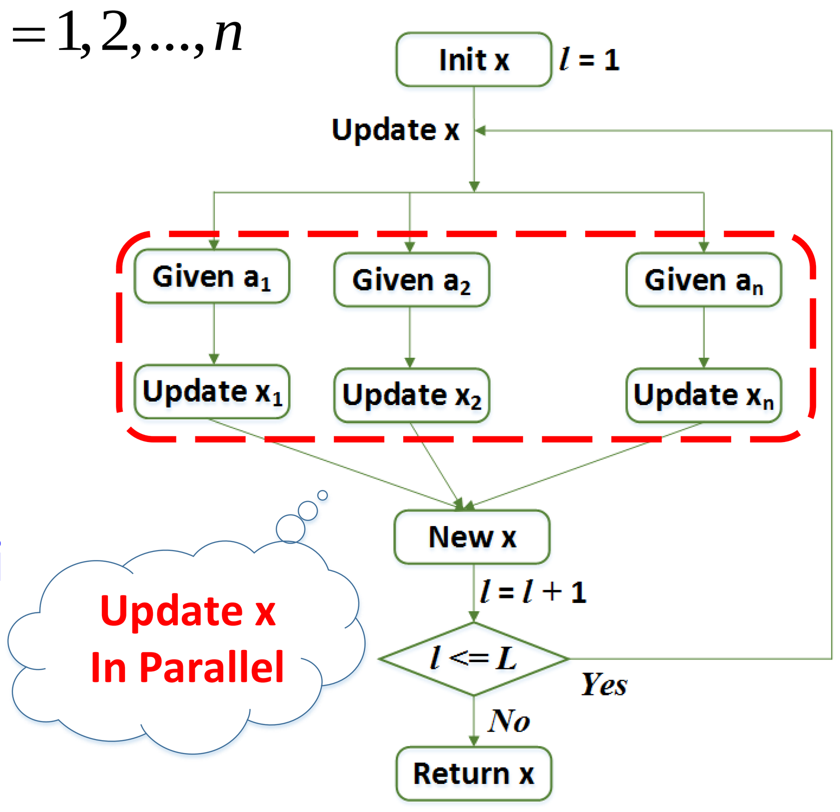
   Indexing linear system $a_i^{\top}x=1, i=1,2,\ldots,n$

   here $a_i=\sum_{t=0}^{T}c^{t-1}(P^{\top})^{t-1}e_i\circ P^{t-1}e_i$

2. Generate a_i's by Monte Carlo simulation, in parallel

3. Solve the linear system via Jacobi method, in parallel

$$x_i^{(k+1)}=\frac{1}{a_{ii}}\Big(1-\sum_{j\neq i}a_{ij}x_j^{(k)}\Big)$$



Jacobi Method

To compute a_i, we obtain P^te_i using Monte Carlo Simulation
1. Place *R* random walkers on node *i*
2. Each walker walks *t* steps along in-links
3. Count the distribution of walkers

❑ **Online queries**

✔ **MCSP: Monte Carlo simulation for single-pair query**
- constant time complexity: $O(TR)$

✔ **MCSS: Monte Carlo simulation for single-source query**
- constant time complexity: $O(T^2R\log d)$

✔ **MCAP: Monte Carlo simulation for all-pair query**
- use MCSS repeatedly; time complexity: $O(nT^2R\log d)$

Implementation on Spark

Why Spark?
- General-purpose in-memory cluster computing
- Easy-to-use operations for distributed applications

Two implementation models
- Broadcasting: Graph stored in each machine
- RDD (Resilient Distributed Dataset): Graph stored in an RDD

Experiments

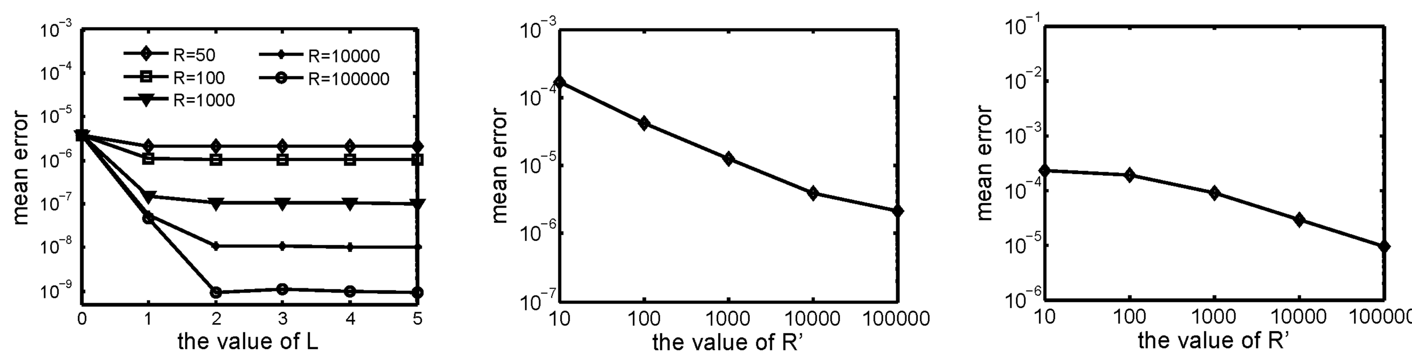
✔ **Setup: cluster, datasets, and default parameters**
- 10 nodes (each with 16 cores, 377GB RAM, 20TB disk)

Dataset	Nodes	Edges	Size
wiki-vote	7.1K	103K	476.8KB
wiki-talk	2.4M	5M	45.6MB
twitter-2010	42M	1.5B	11.4GB
uk-unioni	131M	5.5B	48.3GB
clue-web	1B	42.6B	401.1GB

Parameter	Value	Meaning
c	0.6	decay factor of SimRank
T	10	# of walk steps
L	3	# of iterations in Jacobi method
R	100	# of walkers in simulating a_i
R'	10,000	# of walkers in MCSP and MCSS

10x larger than the largest graph reported on SimRank

✔ **Effectiveness: CloudWalker converges quickly**



wiki-vote dataset

✔ **Broadcasting is more efficient, but RDD is more scalable**

Broadcasting

Dataset	D	MCSP	MCSS
wiki-vote	7s	0.004s	0.042s
wiki-talk	59s	0.046s	0.179s
twitter-2010	975s	0.049s	0.281s
uk-union	3323s	0.025s	0.292s

RDD

Dataset	D	MCSP	MCSS
wiki-vote	50s	2.7s	2.9s
wiki-talk	620s	8.5s	13.9s
twitter-2010	8424s	11.8s	22.3s
uk-union	6.4h	13.1s	27.2s
clue-web	110.2h	64.0s	188.1s

✔ **CloudWalker outperforms state of the art**

Preprocessing, single-pair and single-source queries

Dataset	FMT [2]			LIN [3]			CloudWalker		
	Prep.	SP.	SS.	Prep.	SP.	SS.	Prep.	SP.	SS.
wiki-vote	43.4s	30.4ms	42.5s	187ms	0.61ms	5.3ms	7s	4ms	42ms
wiki-talk	N/A	N/A	N/A	N/A	N/A	N/A	59s	46ms	180ms
twitter-2010	-	-	-	14376s	3.17s	11.9s	975s	49ms	281ms
uk-union	-	-	-	8291s	9.42s	21.7s	3323s	25ms	291ms
clue-web	-	-	-	-	-	-	110.2h	64.0s	188s

[1] G. Jeh and J. Widom. Simrank: a measure of structural-ctontext similarity. *KDD*'02.
[2] D. Fogaras and B. Racz. Scaling link-based similarity search. *WWW*'05.
[3] T. Maehara, et al. Efficient simrank computation via linearization. *CORR*'14.