The First Europe-China Workshop on Big Data Management
May 16, 2016, Helsinki, Finland

I/O-Efficient Big Graph Computation

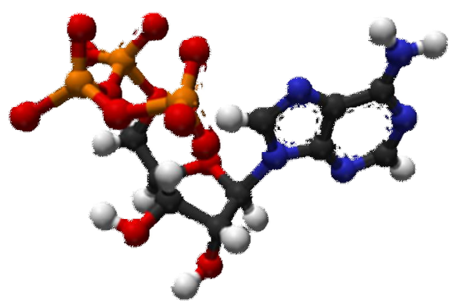
Ge Yu
Northeastern University, China
yuge@cse.neu.edu.cn

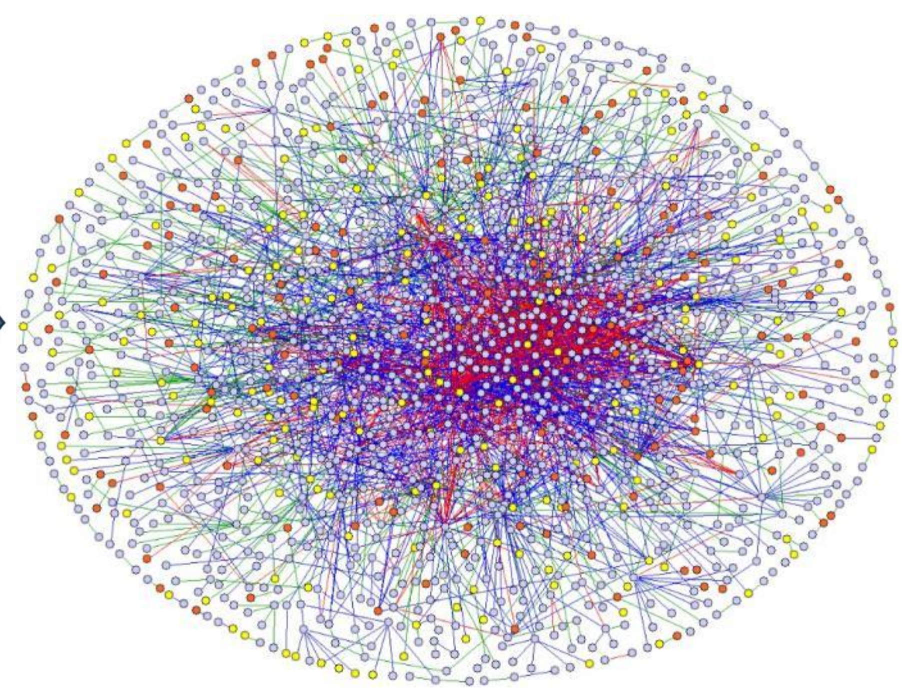
Outline

1. Background & Challenges
2. Push & Pull Style Message Optimization
3. Open source Hybrid-Graph System
4. Work in Progress

1. Background

- Big Graph
  - Social network, scientific computations, …
  - Billion vertices & trillion edges, still growing
  - Iterative algorithms: PageRank, Shortest Path





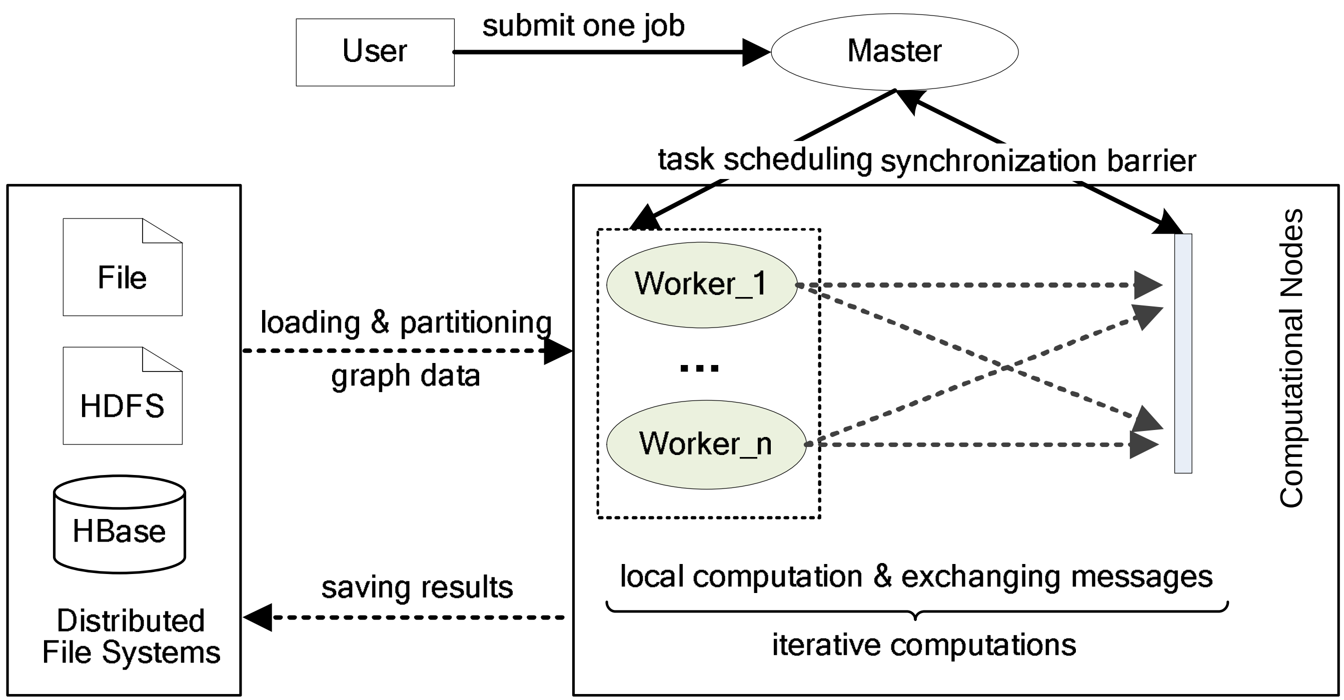
Background

- How to handle “Big Graph” ?
  - Graph data
    “There are 10.4 billion daily active users on Facebook” (Dec, 2015)
  - Message data

Solution	Benefits	Problems	Systems
Single-machine+ Disk	Ease of Management	Poor scalability	GraphChi, TurboGraph
Cluster + memory	High efficiency	Scalability vs. Expense	Pregel, GPS, GraphLab
Cluster + disk	Scalability	I/O-inefficiency	Gbase, Giraph, MOCgraph, Pregelix

Background

- BSP based Distributed Graph Processing Systems
  – Pregel[1], Hama[2], Giraph[3], GPS[4], ...
  – Spark[5], Bagel[5], GraphX[6]
  – GraphLab[7], PowerGraph[8]



Background

- Example of Computation Expense
  - Twitter-2010 (|V|=41.7million, |E|=1.4billion)

    graph data: 13GB

    message data: >=13GB
  - Amazon EC2

Storage Type	Unit Price (per month)	Twitter-2010 (13+13GB)
Memory	$9.36/GB	$244
SSD	$0.10/GB	$2.6
HDD	$0.05/GB	$1.3

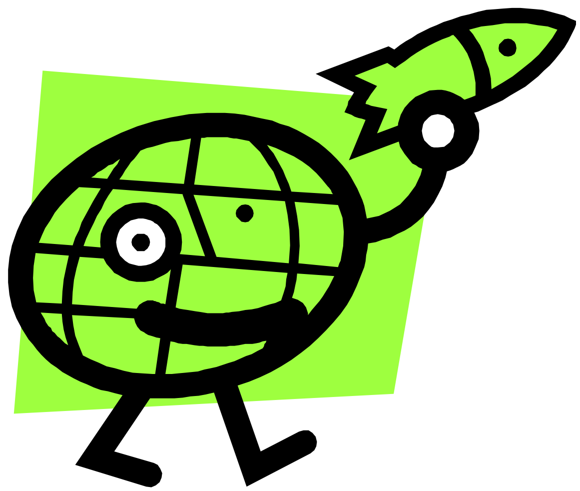
Background

- Optimizations for distributed memory systems
  - Partition:
    METIS[9], LDG[10], vertex-cut[8], load balance[11,12]
  - Message Management:
    combiner[1], LALP[4]
  - Convergence:
    asynchronous[7,8]/block-centric update[13,14]
  - Fault-tolerance:
    checkpoint[1], vertex replication[15]

All these optimizations are NOT designed for I/O efficient computation in disk-resident environments!

Challenges

- I/O-inefficiency for Disk-based systems
  - Graph data IO Cost
    read & write vertices
    read edges
  - Message data IO Cost
    read & write



- Other problems
  - Graph partitioning
  - Convergence
  - Fault-tolerance
  - Data update and incremental maintenance

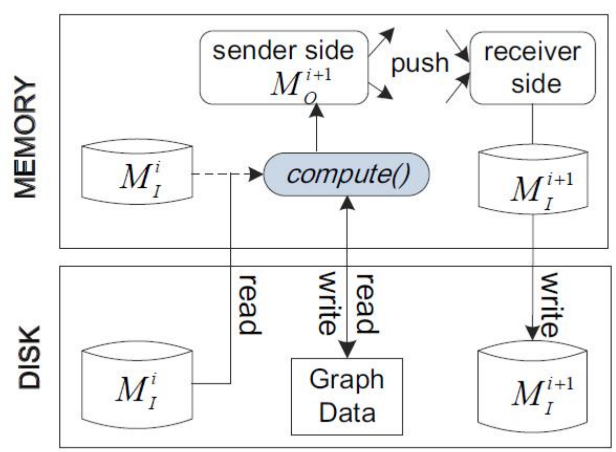
2. Push vs. Pull

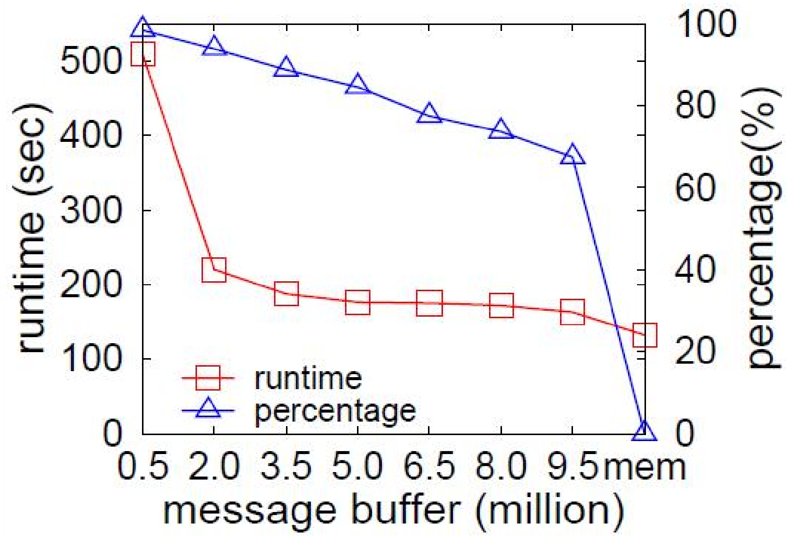
- Two Existing data operations : Push and Pull
- PUSH Approach *(e.g. PageRank on Giraph)*

  <u>*Messages are generated when updating vertices at the i-th iteration, but used in the (i+1)-th iteration*</u>
  - I/O of graph data: seq. reads & writes
  - I/O of message data: random writes & seq.reads

Favorite scenario:

More Messages in memory

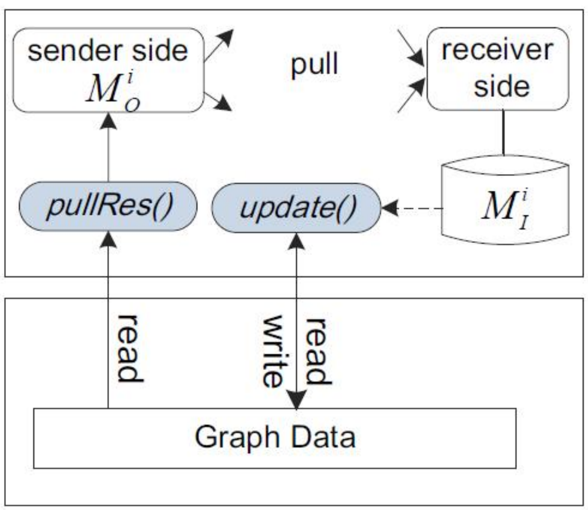




Push vs. Pull

- **PULL** Approach *(e.g. PageRank on modified PowerGraph)*
  Messages are generated on demand of vertex updates and then consumed immediately at the (i+1)-th iteration
  – I/O of graph data: random reads & seq. writes
  – I/O of message data: none

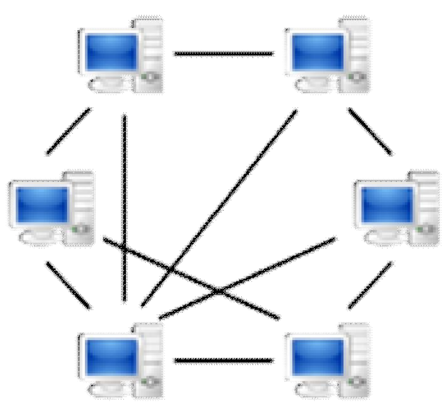
Favorite scenario:
More Graph data in memory



scenarios	PageRank		
	livej	wiki	orkut
original	3.0	3.6	5.0
ext-mem	3.1	4.1	5.9
ext-edge	3.9	4.8	7.1
ext-edge-v3	4.5	5.0	7.1
ext-edge-v2.5	654.7	960.4	1187.8

Existing systems

- Distributed systems
  - Memory-based
    PUSH/PULL
  - Disk-based
    PUSH



Name	PUSH	PULL	DISK
Giraph++	✓		
Blogel	✓		
GiraphX	✓		
GPS	✓		
GRE	✓		
Mizan	✓		
Naiad	✓		
Pregel	✓		
Trinity	✓		
Faunus	✓		✓
PEGASUS	✓		✓
Gbase	✓		✓
Giraph	✓		✓
GraphX	✓		✓
MOCgraph	✓		✓
Hama	✓		✓
Pregelix	✓		✓
Surfer	✓		✓
Chronos	✓	✓	
Kineograph	✓	✓	
Pregel+	✓	✓	
Kylin		✓	
Seraph		✓	
GraphLab PowerGraph		✓	
LFGraph		✓	

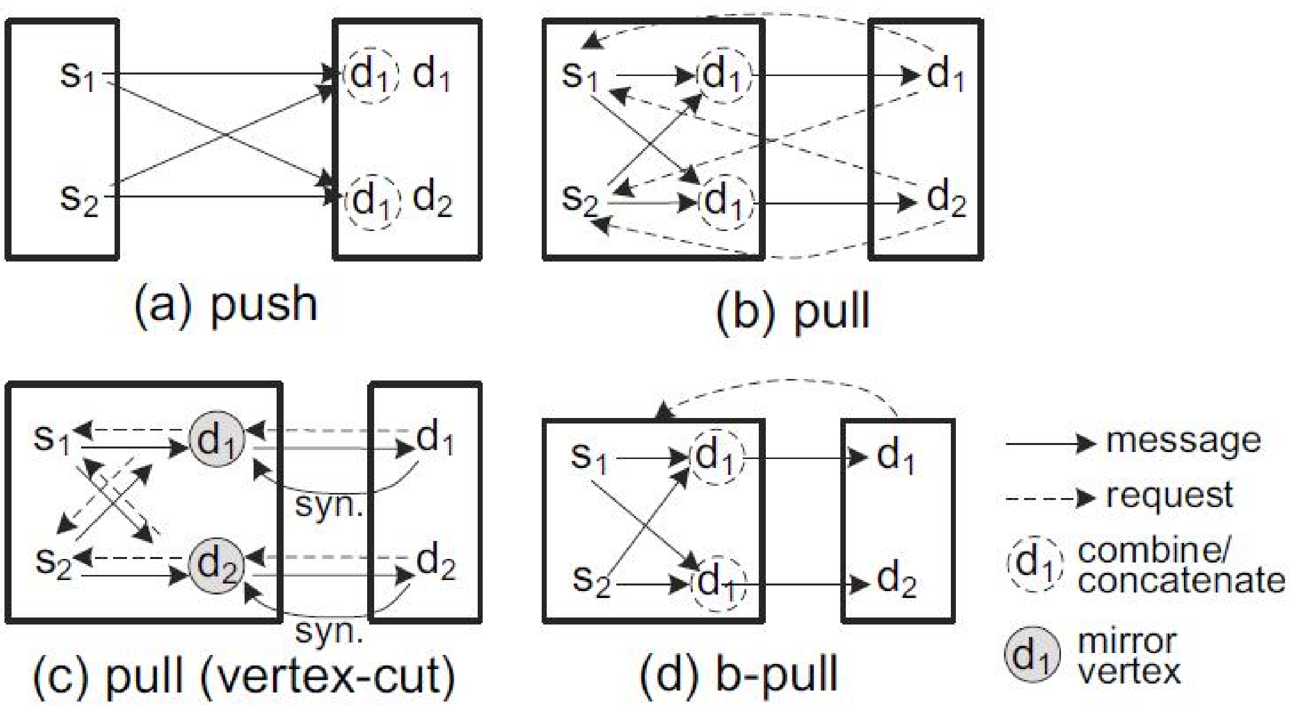
Our Hybrid Solution

- Switching between push and pull adaptively.
- Challenges
  - Existing pull-based approaches are I/O-inefficient.
  - How to efficiently combine push and pull?
- Contributions
  - b-pull: pull messages in block-centric, not vertex-centric
  - VE-BLOCK: data structure for disk-based graph data
  - Hybrid engine: a seamless switching mechanism an effective performance prediction model

- Z. Wang, Y. Gu, Y. Bao, G. Yu, J. Yu. Hybrid Pulling/Pushing for I/O-Efficient Distributed and Iterative Graph Computing. to appear in SIGMOD 2016.

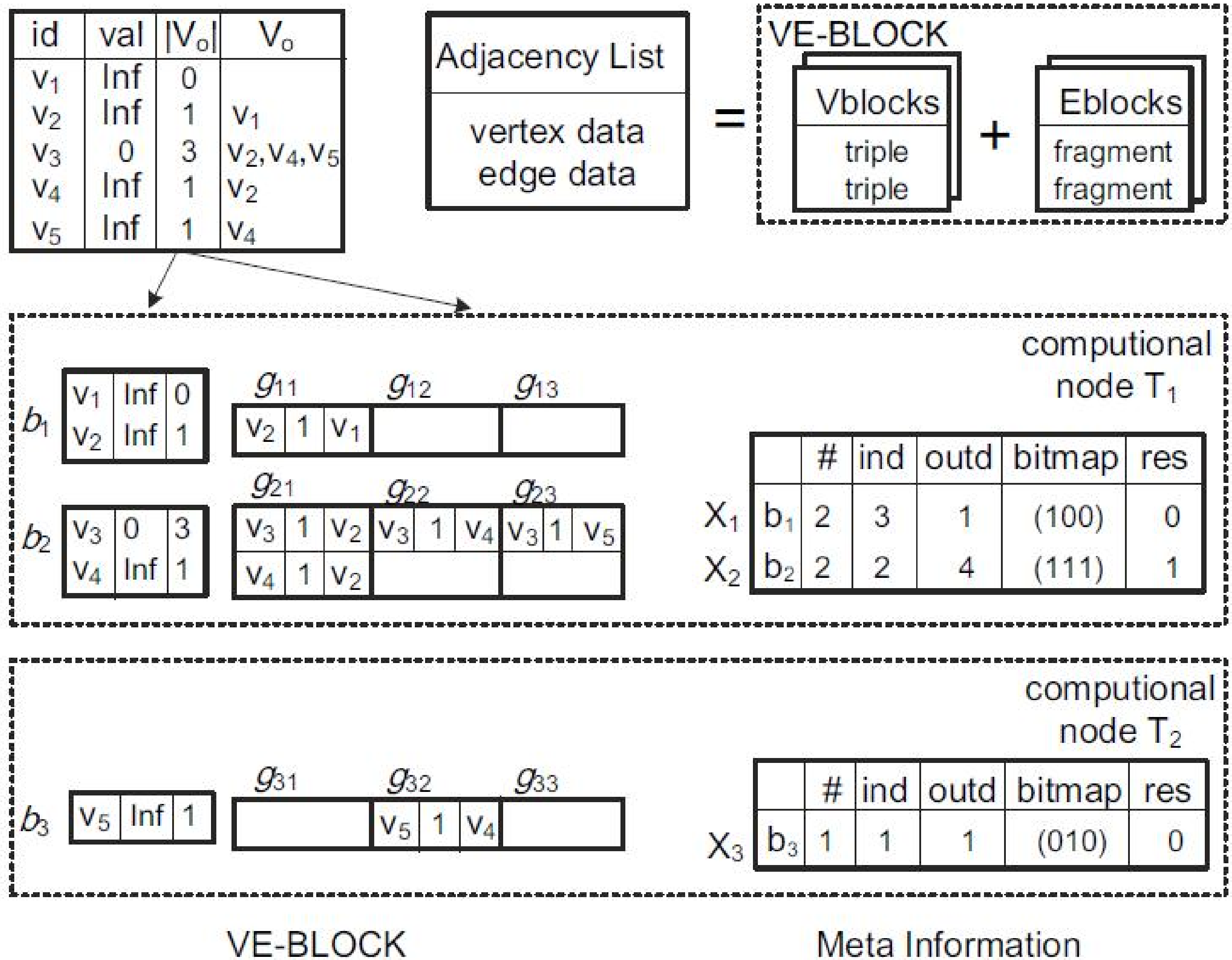
Block-centric Pull (b-pull)

- Strength
  - avoid message data read & write
  - reduce source vertex access costs
  - reduce pull request communication costs



Graph Storage: VE-BLOCK

- VE-BLOCK
  - VBlocks
  - Eblocks
  - Metadata



Block-centric Pull (b-pull)

- Pulling messages in VBlocks
  - 1) Receiver side:

    sending pull requests for one local Vblock b_i
  - 2) Sender side:

    producing and sending messages for b_i on demand

    by reading Vblocks b_j and Eblocks g_{ji}
  - 3) Receiver side:

    consuming messages immediately and updating values of vertices in b_i

  One iteration = Repeat 1)-3) for every Vblock b_i

Hybrid-Decomposition

- Decomposing push and b-pull
  - push: load() à update() à pushRes()
  - b-pull: pullRes() à update()

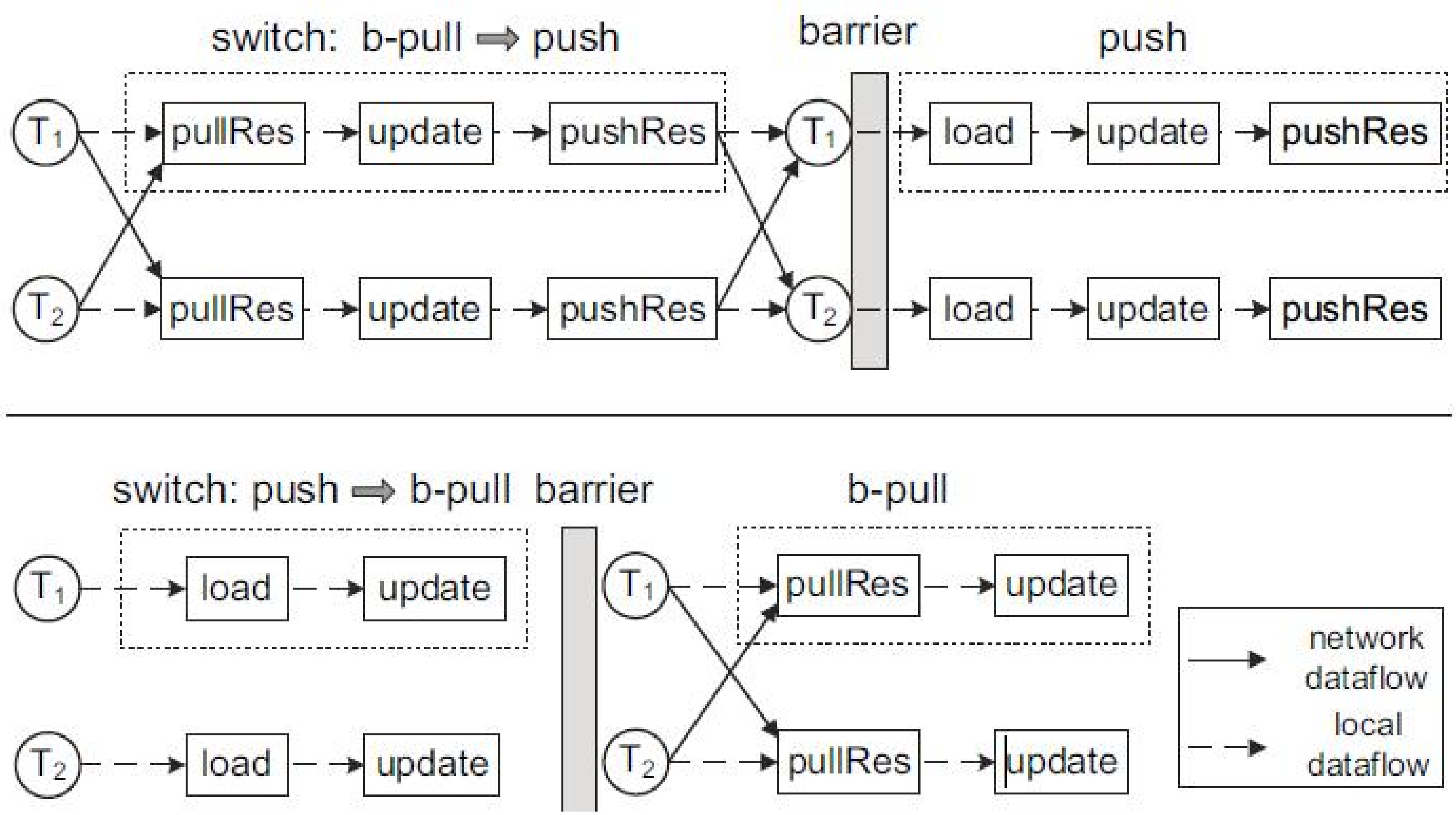
  shared update()

- Graph storage
  - push: Vblocks + edges (in adjacency list)
  - b-pull: Vblocks + edges (in Eblocks)

  shared Vblocks and two replicas of edges

Hybrid-Switching Operations

- Switching between b-pull and push
  - b-pull à push: pulling & pushing messages
  - push à b-pull: no messages are generated



Hybrid-Switching Timing

- Performance metric: Q=Q(push)-Q(b-pull)

$$Q^t = \frac{\mathcal{M}_{co}Byte_m}{s_{net}} + \frac{IO(M_{disk})}{s_{rw}} - \frac{IO(V_{rr}^t)}{s_{rr}} + \frac{IO(E^t) + IO(M_{disk}) - IO(\mathcal{E}^t) - IO(F^t)}{s_{sr}}$$

$Q^t > 0$, b-pull, otherwise, push

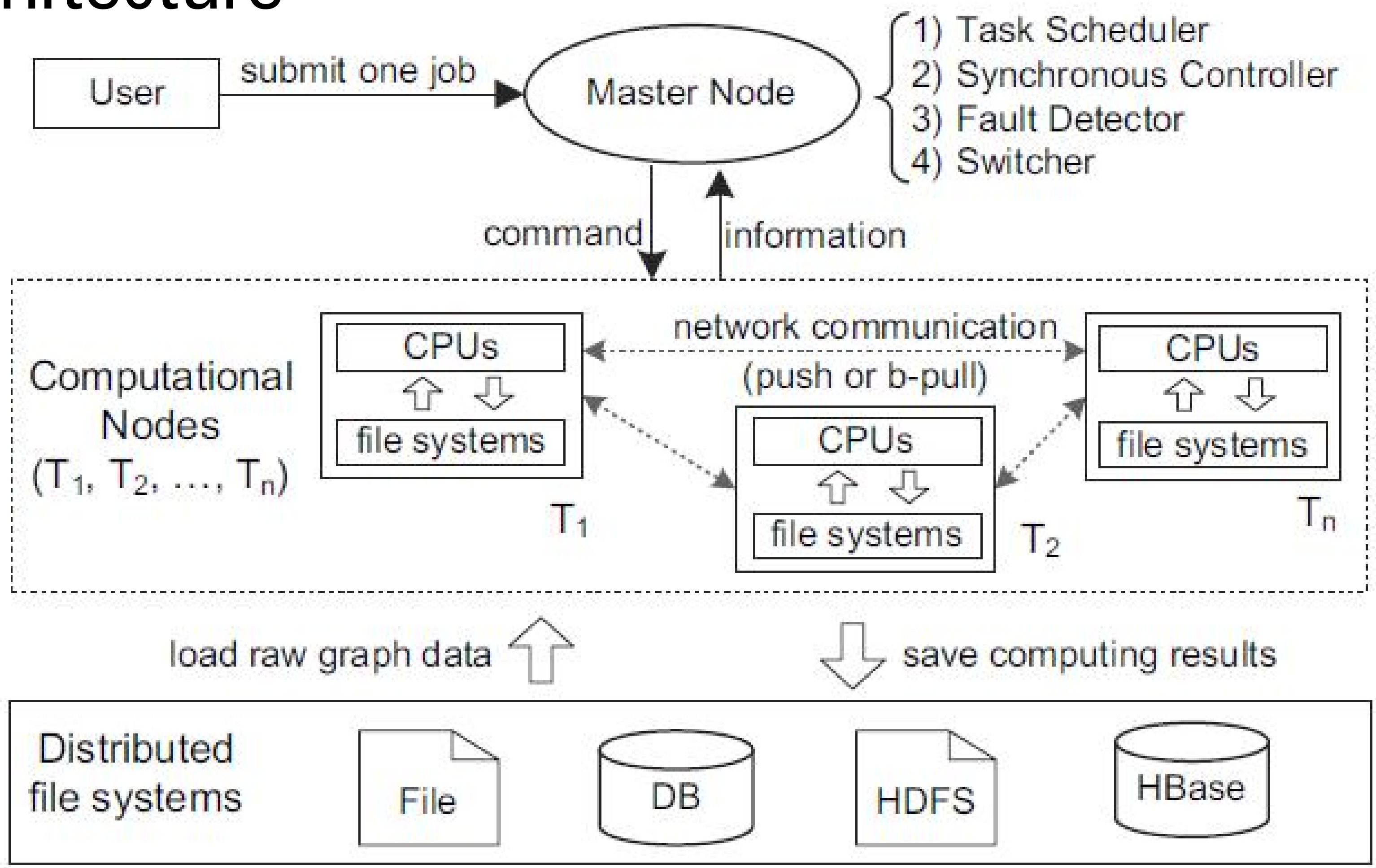
- **How to predict Q?**

Metrics collected at the t-th superstep are regarded as the predicted values on superstep (t+x). (based on the methods in Ref [12] Z. Shang et al., ICDE2013)

- **Default Prediction interval x=2**
  - Prediction accuracy ∝ 1/x, x ∈ N^+
  - x >= 2, to balance the cost and gain

4. HybridGraph System

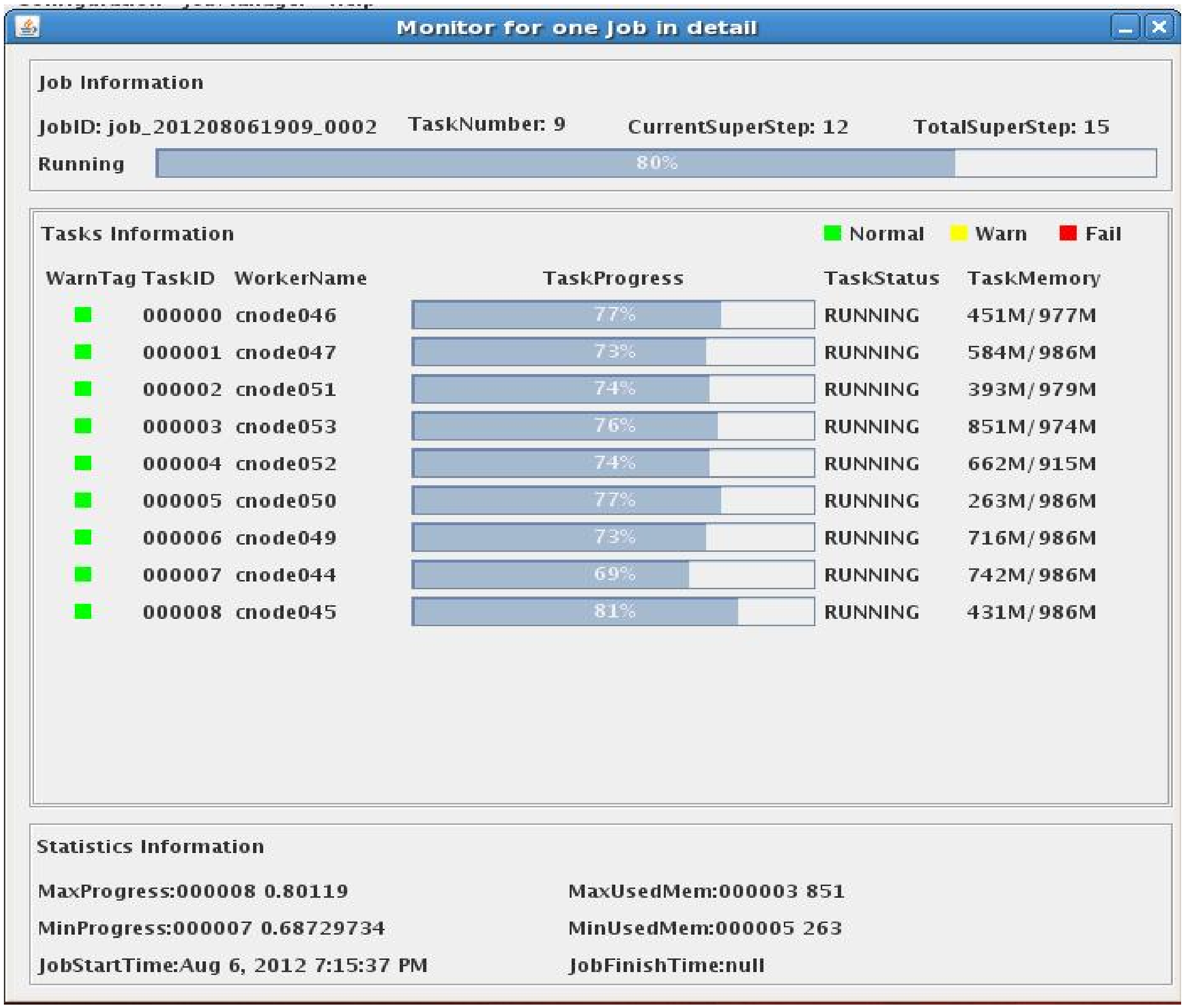
- Architecture



- https://github.com/HybridGraph/HybridGraph

HybridGraph System

- Console of Running one job



Evaluation

- Compared solutions
  - push: Giraph
  - pushM: MOCgraph (message online computing)
  - pull: GraphLab PowerGraph (vertex-cut optimizations)
  - b-pull: our block-centric pull
  - hybrid: our hybrid solution on top of push and b-pull
- Cluster (31 nodes)
  - Local cluster: HDDs
  - Amazon cluster: SSDs

Table 3: Configurations of two clusters

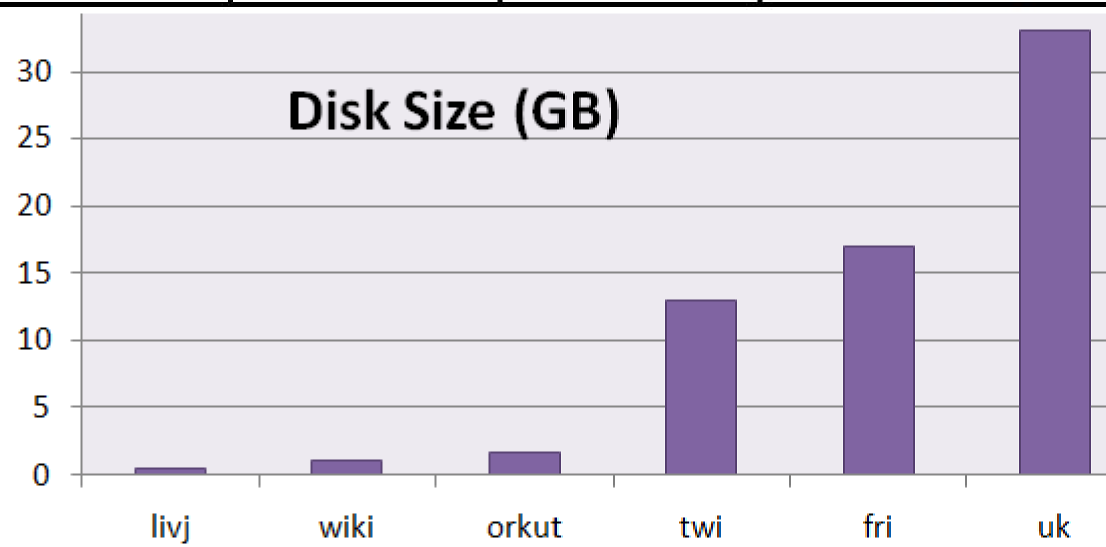
Cluster	RAM	Disk	$s_{rr}/s_{rw}/s_{sr}$ †	s_{net} ‡
local	6.0GB	500GB	1.177/1.182/2.358MB/s	112MB/s
amazon	7.5GB	30GB	18.177/18.194/18.270MB/s	116MB/s

Evaluation

- Dataset

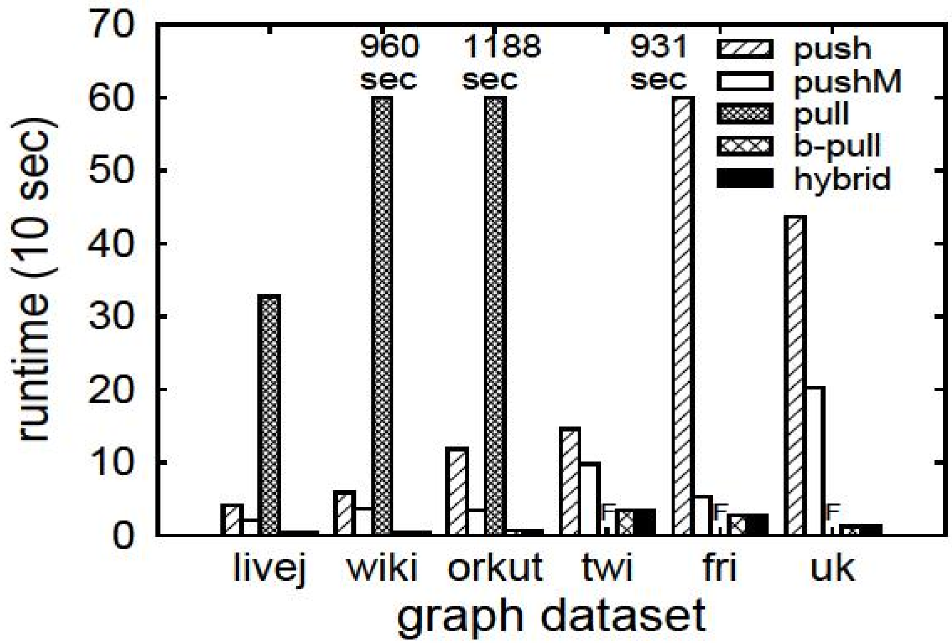
Table 4: Real Graph Datasets (M: million)

Graph	Vertices	Edges	Degree	Type	Disk size
livej[5]	4.8M	68M	14.2	Social networks	0.50GB
wiki[6]	5.7M	130M	22.8	Web graphs	0.98GB
orkut[7]	3.1M	234M	75.5	Social networks	1.59GB
twi[8]	41.7M	1,470M	35.3	Social networks	12.90GB
fri[9]	65.6M	1,810M	27.5	Social networks	17.00GB
uk[10]	105.9M	3,740M	35.6	Web graphs	33.02GB

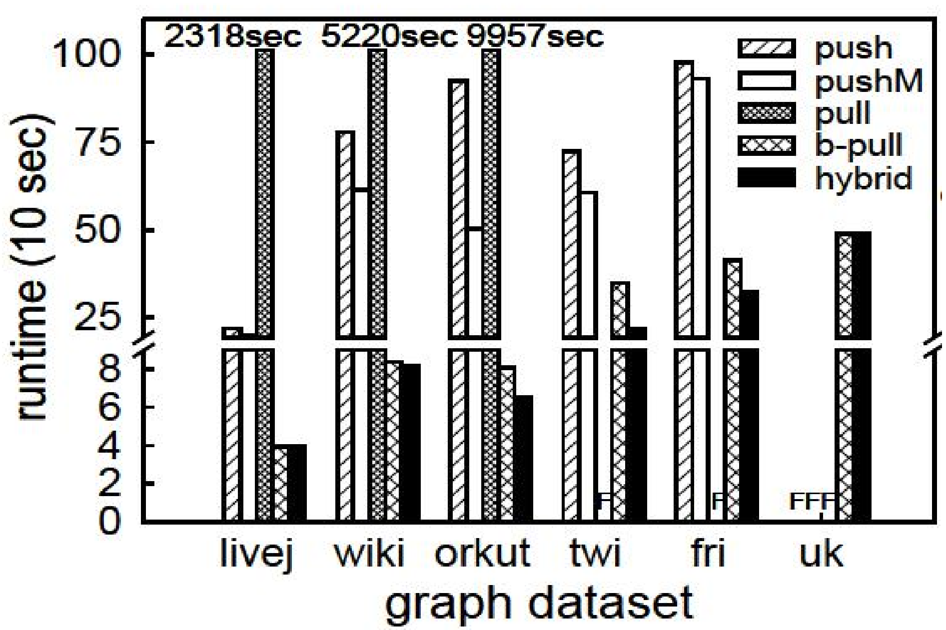


Evaluation

- Limited memory (local cluster with HDDs)
  - b-pull vs. push: up to 35x
  - b-pull vs. pushM: up to 16x
  - b-pull vs. pull: up to 239x
  - hybrid vs. b-pull: up to 1.6x



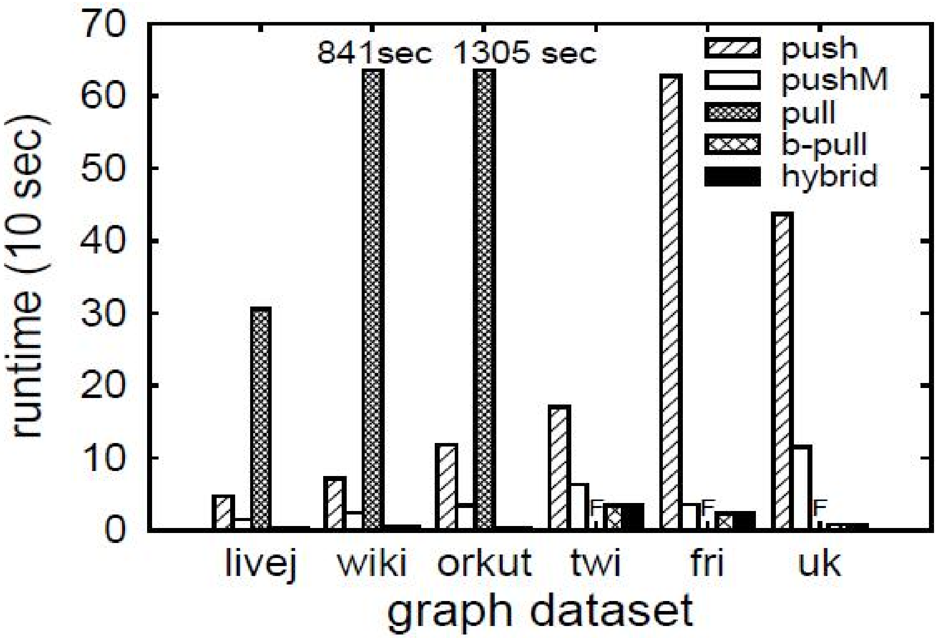
(a) Runtime of PageRank



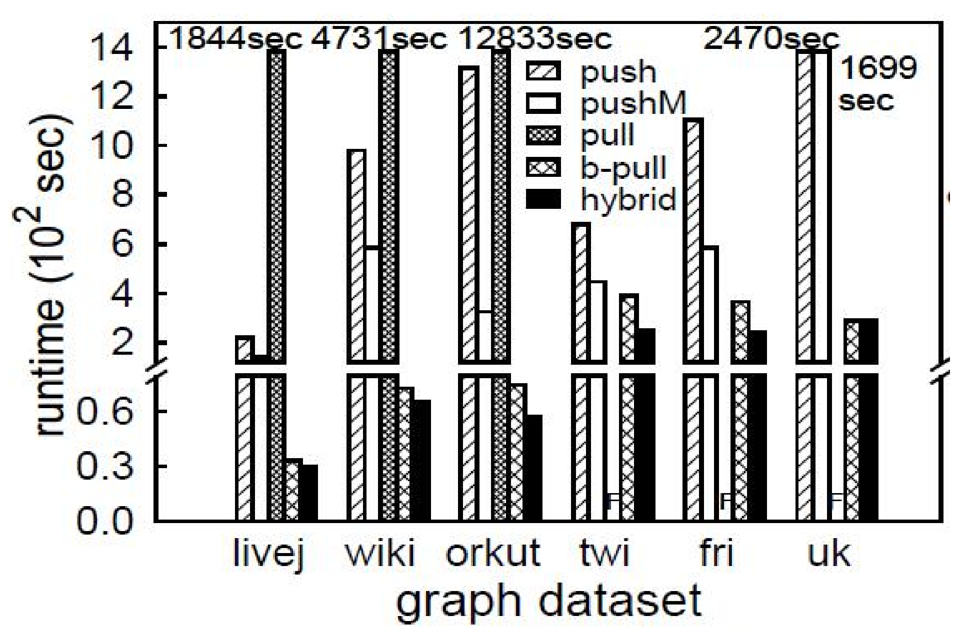
(b) Runtime of SSSP

Evaluation

- Limited memory (Amazon cluster with SSDs)
  - b-pull vs. push: up to 56x
  - b-pull vs. pushM: up to 15x
  - b-pull vs. pull: up to 343x
  - hybrid vs. b-pull: up to 1.5x



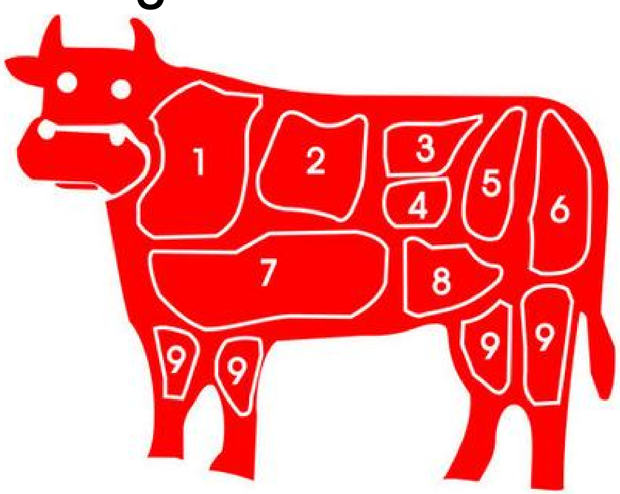
(a) Runtime of PageRank



(b) Runtime of SSSP

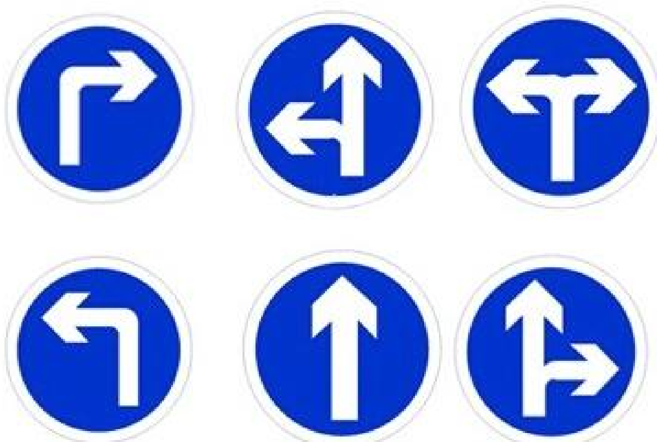
5. Work in progress (1)

- Partitioning graph for I/O-efficient computation
  - Besides reducing # of cut-edges and balancing load, also considering reducing I/O costs of graph data index
  - Building a reverse graph <u>G</u> and Clustering vertices on <u>G</u> based on the similarities of outgoing edges (for I/O costs)
  - Assigning clustered vertex blocks among machines using METIS (for cut-edges and load balance)



Work in progress (2)

- Prioritized block scheduling for b-PULL
  - Asynchronous vertex update to speed up message propagation
  - Prioritized block scheduling instead of vertex scheduling to avoid random access to vertices
  - Estimating block priority based on dynamical block-dependency graph

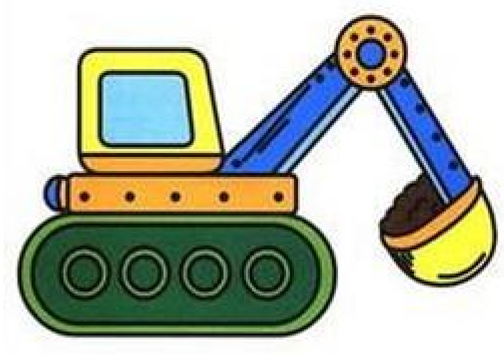


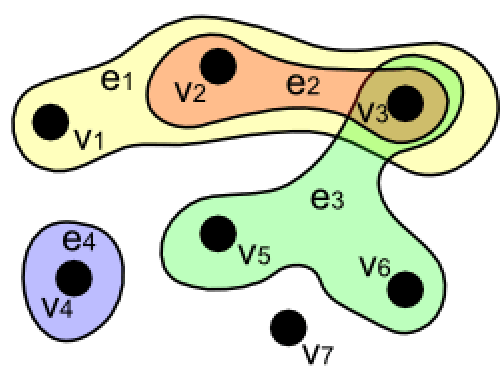
Work in progress (3)

- Lightweight Fault-tolerance using b-pull
  - Archiving local historical data (vertices, not messages)
  - Writing distributed checkpoint data in parallel with updating vertices (not a blocking way)
  - For failure recovery, re-pulling messages for lost vertices and updating lost vertex values based on b-pull

Work in progress (4)

- More functions for various applications
  - Offering a library implementing the typical iterative algorithms(graph computation, matrix analysis and data mining)
  - Friendly interface and visualization
  - Incremental computation and index maintenance for graph data updates
  - Support for temporal graphs and hypergraphs





References

[1] Malewicz G, Austern M H, Bik A J C, et al. Pregel: a system for large-scale graph processing. SIGMOD2010.

[2] https://hama.apache.org/

[3] http://giraph.apache.org/

[4] Salihoglu S, Widom J. Gps: A graph processing system. SSDBM2013.

[5] http://spark.apache.org/

[6] Gonzalez J E, Xin R S, Dave A, et al. Graphx: Graph processing in a distributed dataflow. OSDI2014..

[7] Low Y, Bickson D, Gonzalez J, et al. Distributed GraphLab: a framework for machine learning and data mining in the cloud. VLDB2012.

[8] Gonzalez J E, Low Y, Gu H, et al. Powergraph: Distributed graph-parallel computation on natural graphs. OSDI 2012.

[9] Karypis G, Kumar V. Analysis of multilevel graph partitioning. Supercomputing1995.

[10] Stanton I, Kliot G. Streaming graph partitioning for large distributed graphs. KDD2012.

[11] Khayyat Z, Awara K, Alonazi A, et al. Mizan: a system for dynamic load balancing in large-scale graph processing. EuroSys2013.

[12] Shang Z, Yu J X. Catch the wind: Graph workload balancing on cloud. ICDE2013.

[13] Tian Y, Balmin A, Corsten S A, et al. From think like a vertex to think like a graph. VLDB2013.

[14] Yan D, Cheng J, Lu Y, et al. Blogel: A block-centric framework for distributed computation on real-world graphs. VLDB2014.

[15] Pundir M, Leslie L M, Gupta I, et al. Zorro: Zero-cost reactive failure recovery in distributed graph processing. SoCC.

Thanks!