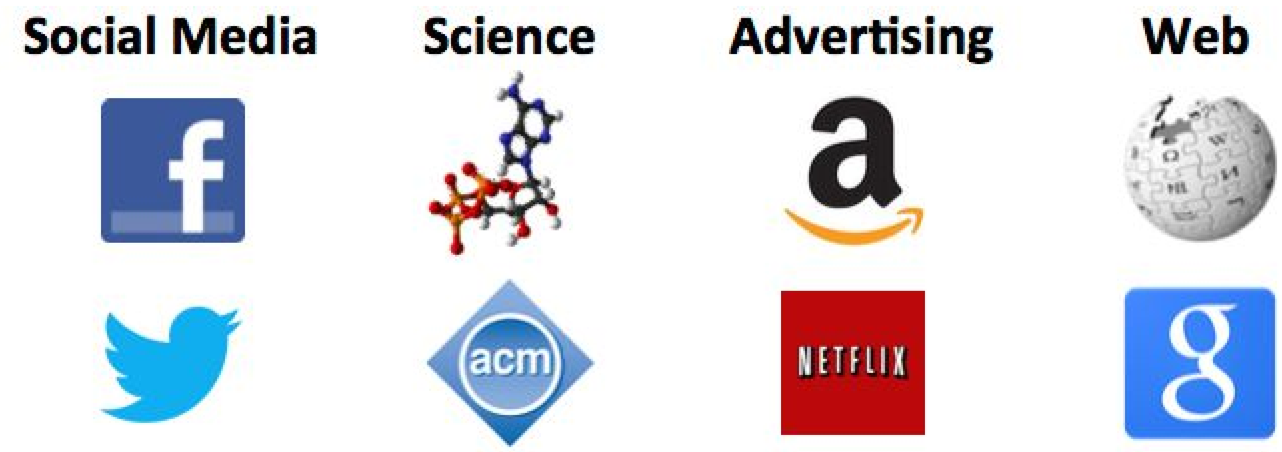
PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs

Gonzales et al.

James Trever

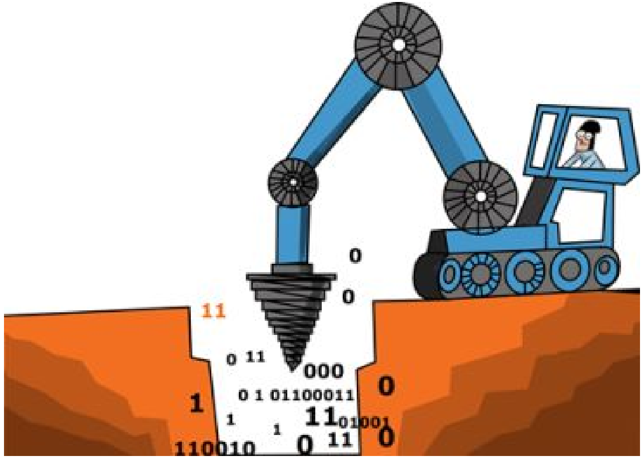
What are Graphs?

Graphs are everywhere and used to encode relationships



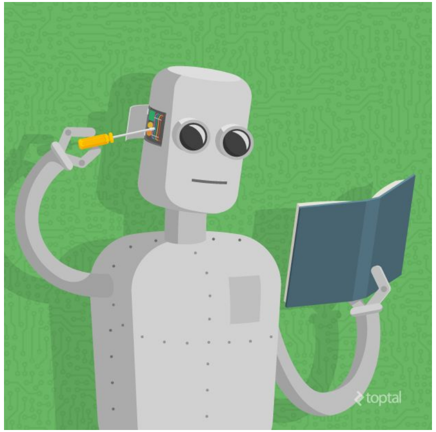
So what are they used for?

- Targeted ads
- Natural Language Processing
- Identifying influential people and information



Data Mining

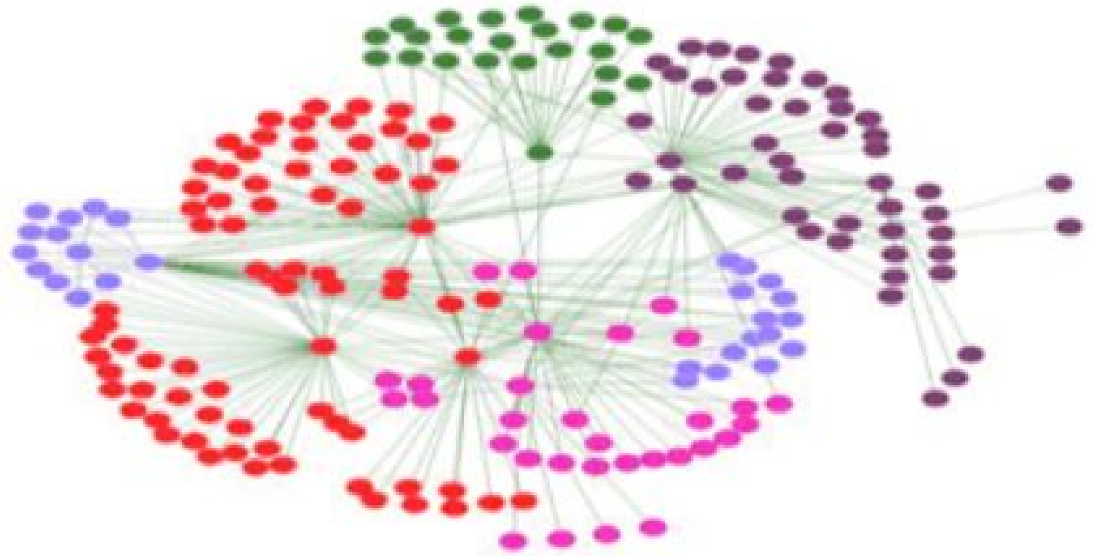
Machine Learning



Natural Graphs

Graphs derived from real world phenomena

Challenges with Natural Graphs



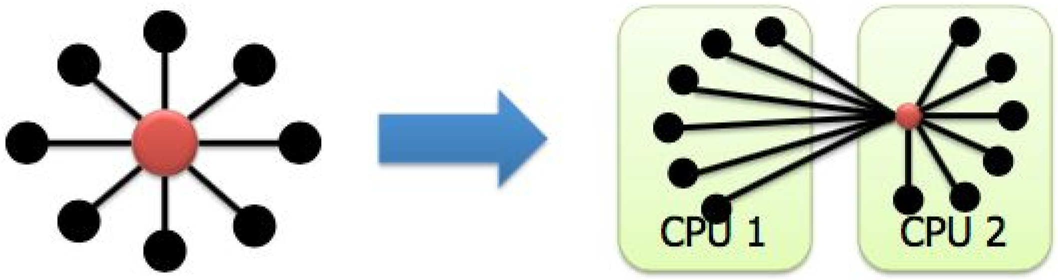
Power-Law Degree Distribution

#	Name (Screen Name)	Location	URL	Followers	Following	Updates	Joined
1.	 KATY PERRY @katyperry		http://t.co/TUWZkUEN3o	76,701,097	157	6,691	81 months ago
2.	 Justin Bieber @justinbieber		http://t.co/2B1YApLM35	68,765,135	237,436	29,760	80 months ago
3.	 Barack Obama @BarackObama	Washington, DC	http://t.co/O5Woad92z1	65,259,895	639,547	14,163	Details...
4.	 Taylor Swift @taylorswift13		http://t.co/AjT5TRgs35	64,195,315	240	3,976	84 months ago
5.	YouTube @YouTube	San Bruno, CA	http://t.co/F3fLcfnBVf	55,801,499	896	14,982	97 months ago

Graph-Parallel Abstraction

- A Vertex-Program, designed by the user, runs on every vertex
- Vertex-Programs interact with one another along their edges
- Multiple Vertex-Programs are run simultaneously

Challenges with Natural Graphs



- Power-Law Graphs are very difficult to partition/cut
- Often incurs a large communication or storage overhead

Existing Systems

Pregel

&

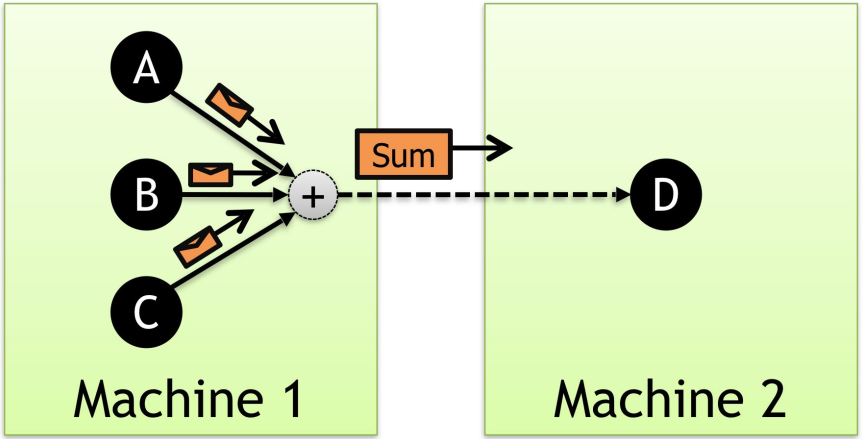
GraphLab

Pregel

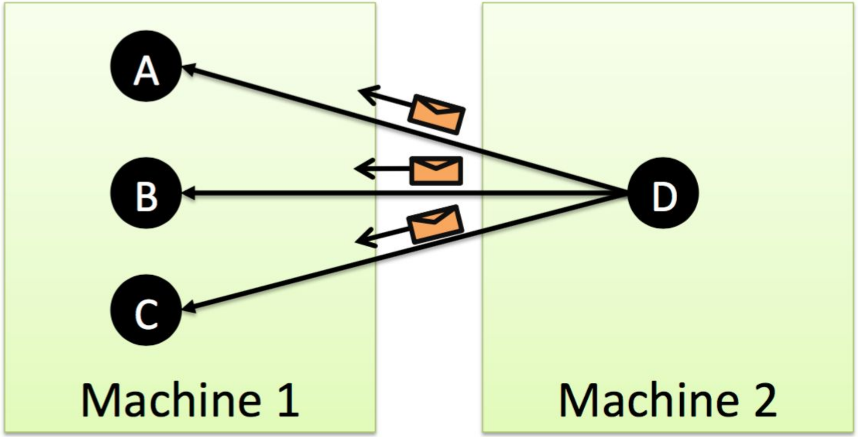
- Bulk Synchronous Message Passing Abstraction
- Uses messages to communicate with other vertices
- Waits until all vertex programs have finished before starting the next “super step”
- Uses message combiners

```
Message combiner(Message m1, Message m2) :
  return Message(m1.value() + m2.value());
void PregelPageRank(Message msg) :
  float total = msg.value();
  vertex.val = 0.15 + 0.85*total;
  foreach(nbr in out_neighbors) :
     SendMsg(nbr, vertex.val/num_out_nbrs);
```

Pregel



Fan-In



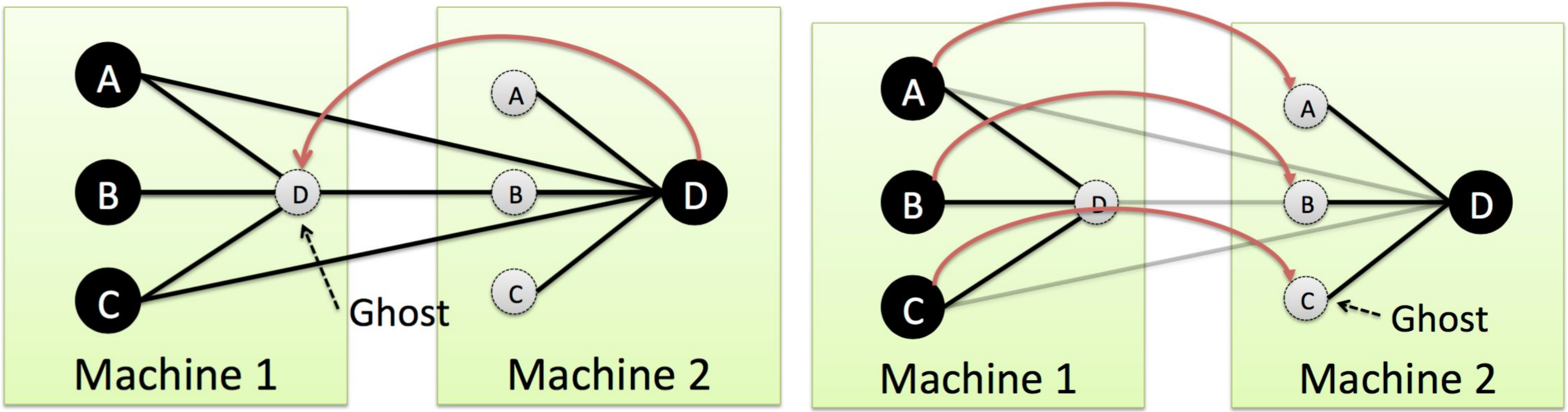
Fan-Out

GraphLab

- Asynchronous Distributed Shared-Memory Abstraction
- Vertex-Programs have shared access to distributed graph with data stored on each vertex and edge and can access the current vertex, adjacent edges and adjacent vertices irrespective of edge direction
- Vertex-Programs have the ability to schedule other vertices’ execution in the future

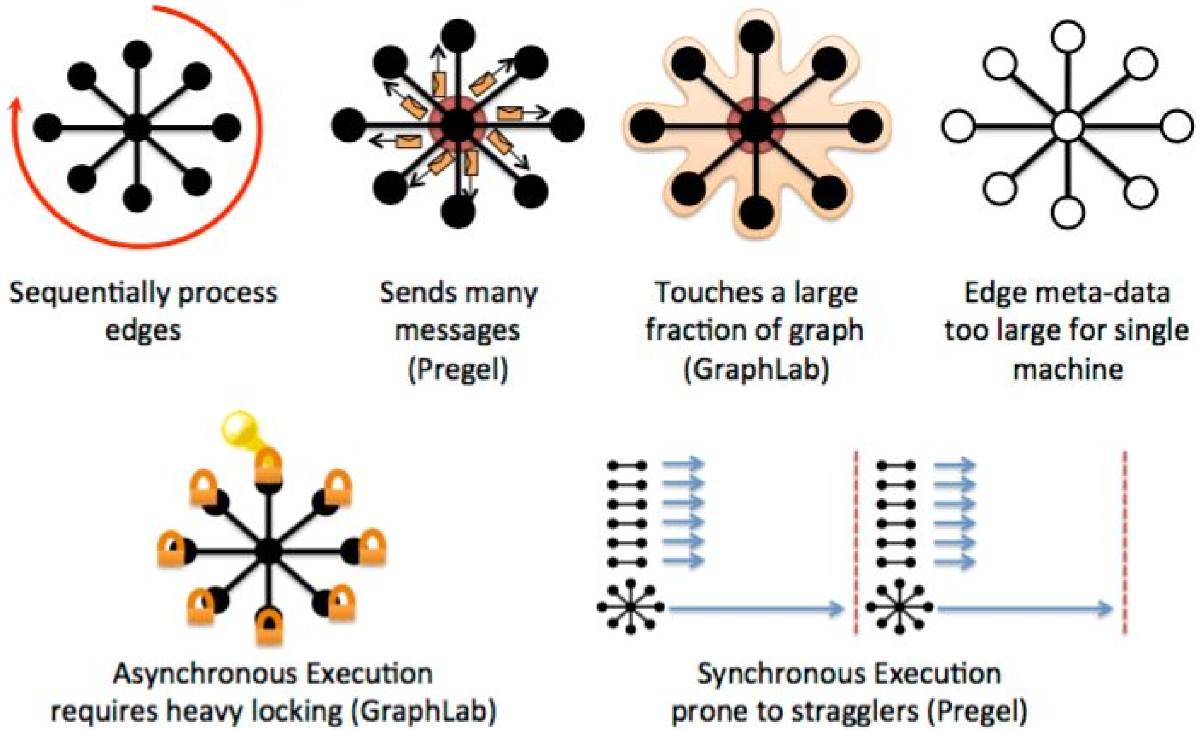
```
void GraphLabPageRank(Scope scope) :
  float accum = 0;
  foreach (nbr in scope.in_nbrs) :
     accum += nbr.val / nbr.nout_nbrs();
  vertex.val = 0.15 + 0.85 * accum;
```

GraphLab



GraphLab Ghosting

Challenges with Natural Graphs

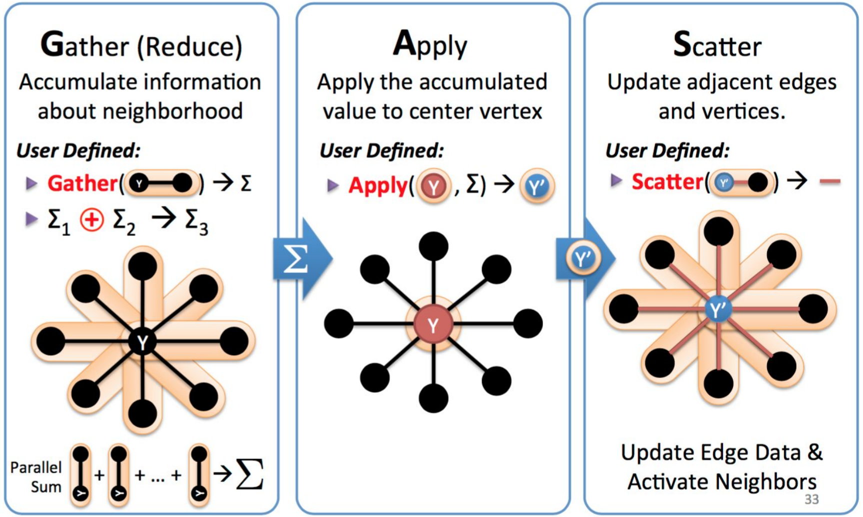


PowerGraph

PowerGraph

- GAS Decomposition
  - Distribute Vertex-Programs
  - Parallelise high degree vertices
- Vertex Partitioning
  - Distribute power-law graphs more efficiently

GAS Decomposition

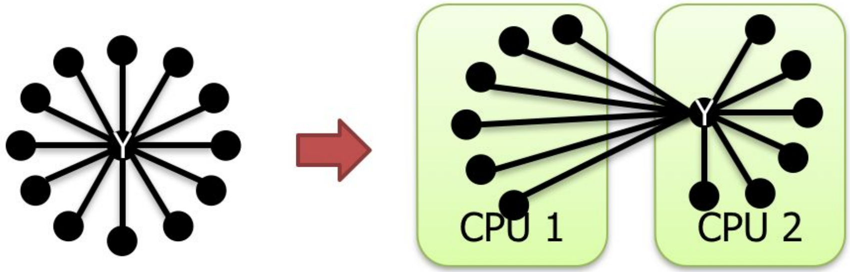


interface *GASVertexProgram*(u) {
  // Run on gather_nbrs(u)
  gather(D_u, $D_{(u,v)}$, D_v) $\rightarrow$ *Accum*
  sum(*Accum left*, *Accum right*) $\rightarrow$ *Accum*
  apply(D_u, *Accum*) $\rightarrow$ D_u^{new}
  // Run on scatter_nbrs(u)
  scatter(D_u^{new}, $D_{(u,v)}$, D_v) $\rightarrow$ ($D_{(u,v)}^{\text{new}}$, *Accum*)
}

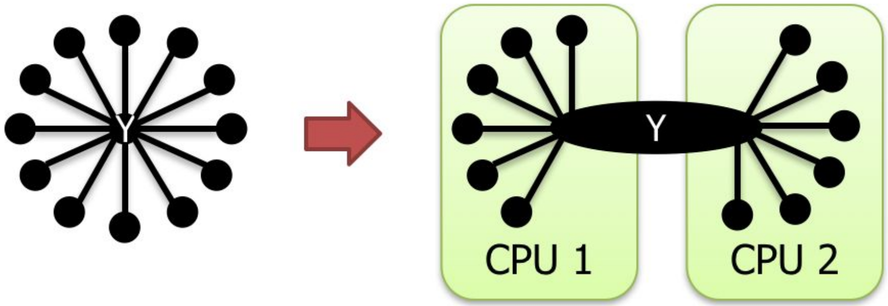
Algorithm 1: Vertex-Program Execution Semantics

Input: Center vertex u
if *cached accumulator* a_u *is empty* **then**
  foreach *neighbor* v *in gather_nbrs(u)* **do**
    $a_u \leftarrow \text{sum}(a_u, \text{gather}(D_u, D_{(u,v)}, D_v))$
  end
end
$D_u \leftarrow \text{apply}(D_u, a_u)$
foreach *neighbor* v *scatter_nbrs(u)* **do**
  $(D_{(u,v)}, \Delta a) \leftarrow \text{scatter}(D_u, D_{(u,v)}, D_v)$
  if a_v *and* Δa *are not Empty* **then** $a_v \leftarrow \text{sum}(a_v, \Delta a)$
  else $a_v \leftarrow$ Empty
end

Vertex Partitioning

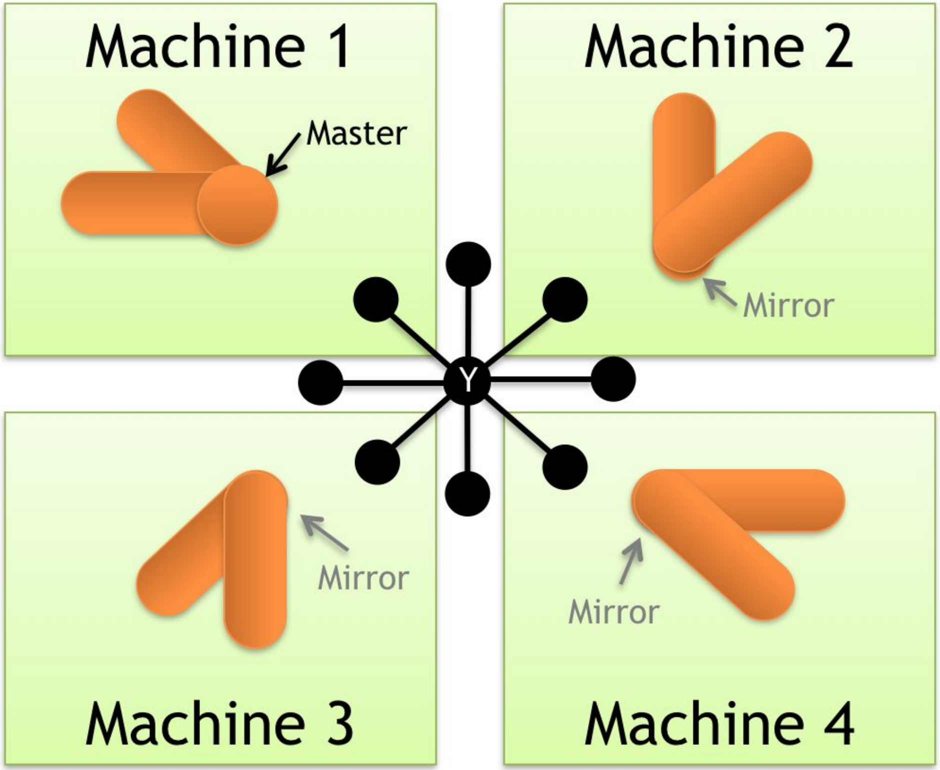


Edge Cuts



Vertext Cuts

Vertex Partitioning

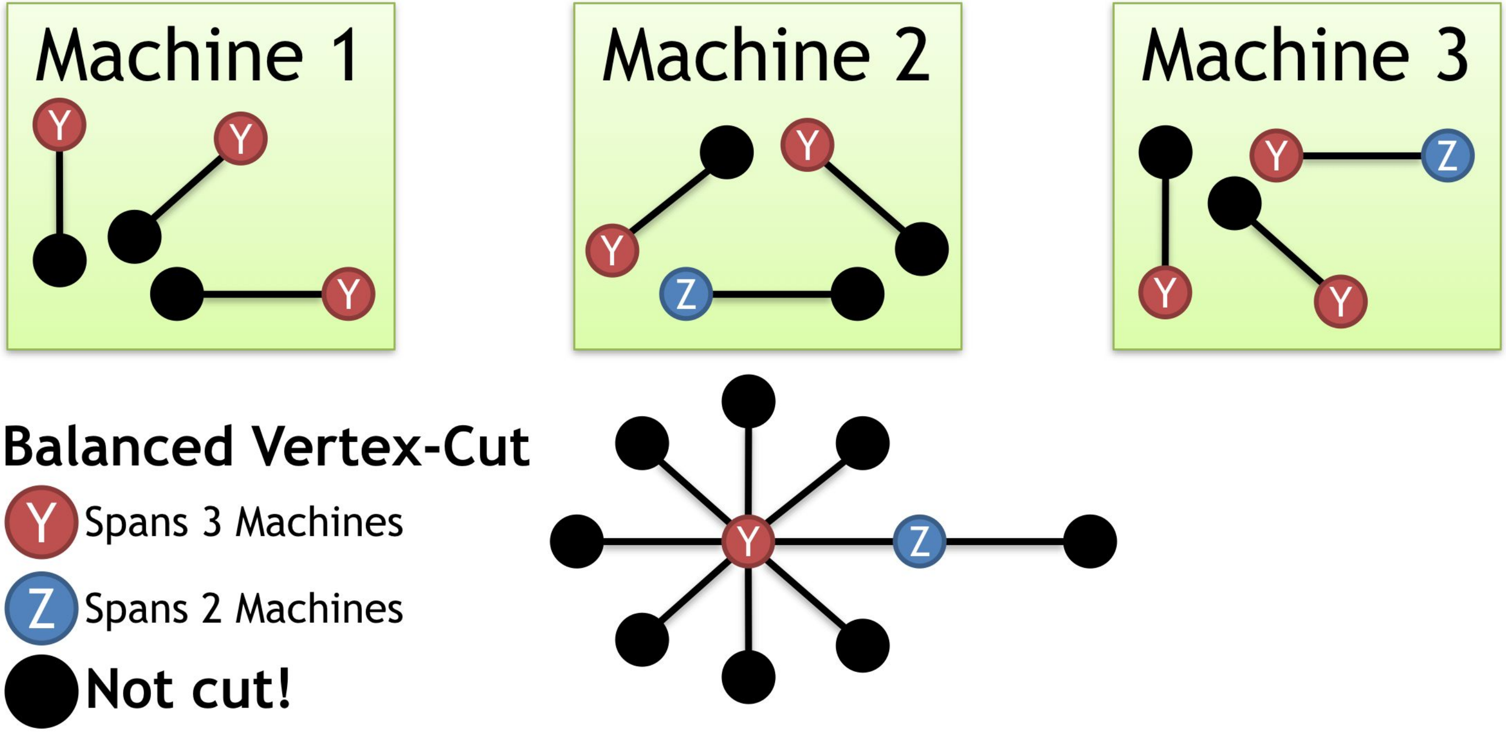


How the vertices are partitioned

- Evenly assign edges to machines

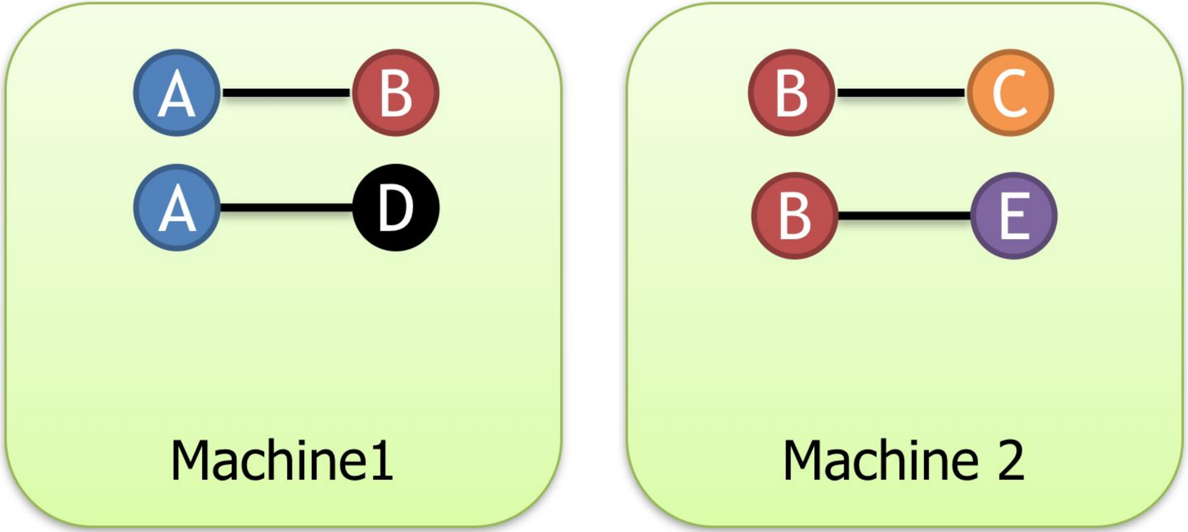
- 3 different approaches
    - Random edge placement
    - Greedy placement
        - Coordinated edge placement
        - Oblivious edge placement

Random Edge Placements



Greedy Edge Placements

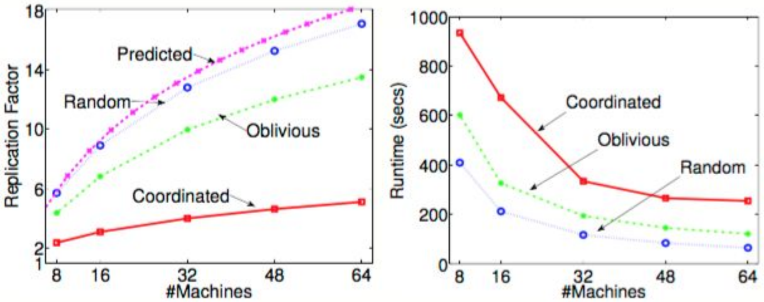
- Place edges on machines that already have the vertices in that edge
- If there are multiple options, choose the less loaded machine



Greedy Edge Placements

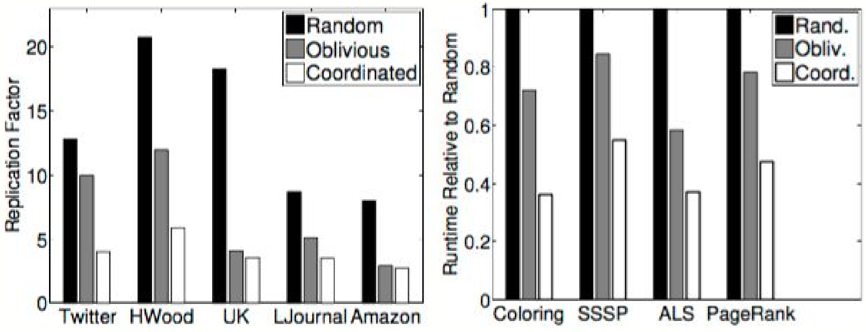
- Minimises the expected number of machines spanned
- Coordinated:
  - Requires coordination to place each edge
  - Slower but has higher quality cuts
- Oblivious:
  - Approximate greedy objective without coordination
  - Faster but lower quality cuts

Experiments - Graph Partitioning



(a) Replication Factor (Twitter) (b) Ingress time (Twitter)

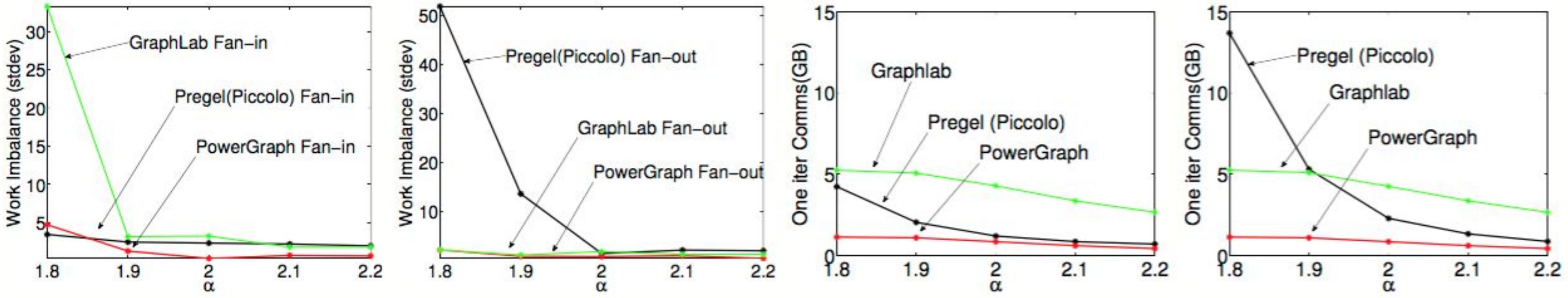
Figure 8: **(a,b)** Replication factor and runtime of graph ingress for the Twitter follower network as a function of the number of machines for random, oblivious, and coordinated vertex-cuts.



(a) Actual Replication (b) Effect of Partitioning

Figure 7: **(a)** The actual replication factor on 32 machines. **(b)** The effect of partitioning on runtime.

Experiments - Synthetic Work Imbalance and Communication



(a) Power-law Fan-In Balance (b) Power-law Fan-Out Balance (c) Power-law Fan-In Comm. (d) Power-law Fan-Out Comm.

Figure 9: **Synthetic Experiments: Work Imbalance and Communication. (a, b)** Standard deviation of worker computation time across 8 distributed workers for each abstraction on power-law fan-in and fan-out graphs. **(b, c)** Bytes communicated per iteration for each abstraction on power-law fan-in and fan-out graphs.

Experiments - Synthetic Runtime

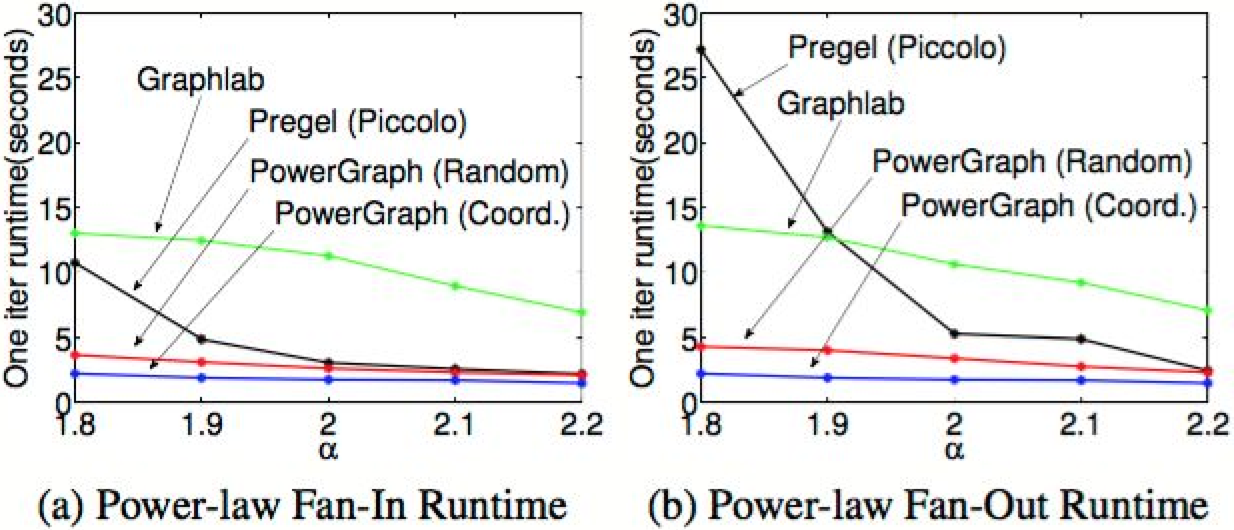


Figure 10: **Synthetic Experiments Runtime. (a, b)** Per iteration runtime of each abstraction on synthetic power-law graphs.

Experiments - Machine Learning

PageRank	Runtime	$\|V\|$	$\|E\|$	System
Hadoop [22]	198s	–	1.1B	50x8
Spark [37]	97.4s	40M	1.5B	50x2
Twister [15]	36s	50M	1.4B	64x4
PowerGraph (Sync)	3.6s	40M	1.5B	64x8

Triangle Count	Runtime	$\|V\|$	$\|E\|$	System
Hadoop [36]	423m	40M	1.4B	1636x?
PowerGraph (Sync)	1.5m	40M	1.4B	64x16

LDA	Tok/sec	Topics	System
Smola et al. [34]	150M	1000	100x8
PowerGraph (Async)	110M	1000	64x16

Table 2: Relative performance of PageRank, triangle counting, and LDA on similar graphs. PageRank runtime is measured per iteration. Both PageRank and triangle counting were run on the Twitter follower network and LDA was run on Wikipedia. The systems are reported as number of nodes by number of cores.

Other Features

- 3 different execution modes:
  - Bulk Synchronous
  - Asynchronous
  - Asynchronous Serialisable
- Delta Caching

Critical Evaluation

- Lots of talk of performance, not many tests comparing systems
- Delta caching only briefly touched on
- Future work lacks detail
- Lots of unbacked up claims
- Greedy edge placement not very clear
- No mention of fault tolerance

Bibliography

J. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin: Powergraph: distributed graph-parallel computation on naturalgraphs. OSDI, 2012.

And his original presentation found here:

http://www.cs.berkeley.edu/~jegonzal/talks/powergraph_osdi12.pptx