Distributed Computation of Large-scale Graph Problems

Peter Robinson

Joint work with:

Hartmut Klauck (NTU)

Danupon Nanongkai (KTH)

Gopal Pandurangan (UH)

Michele Scquizzato (UH)

Large-scale Graph Data

Large Graphs

- Web graph, social networks
- Transportation networks
- Buyer-seller relationship graphs
- ...

Too big to be processed by single commodity machine.

Important Problems

- Minimum spanning trees
- Routing, shortest paths, …
- PageRank
- Community detection
- Connectivity testing, …

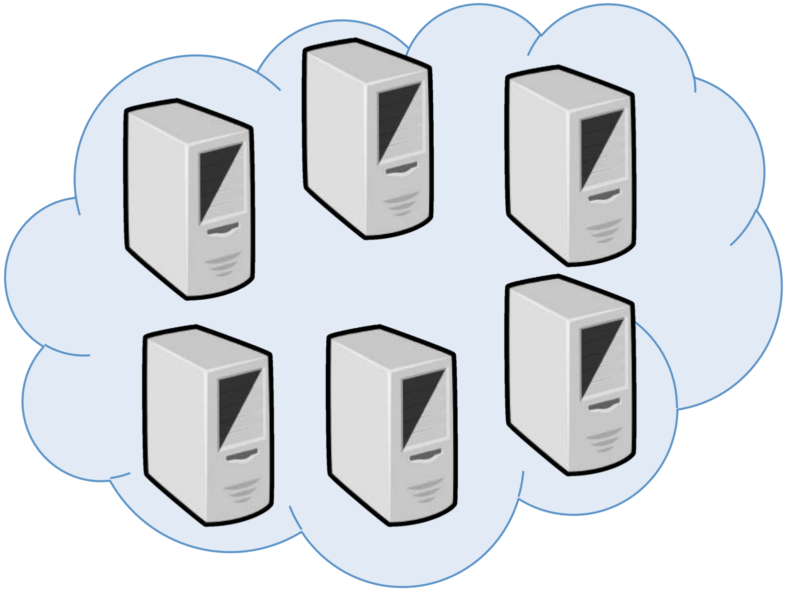
Handling Large Graphs

Approach 1:
Buy more powerful hardware.

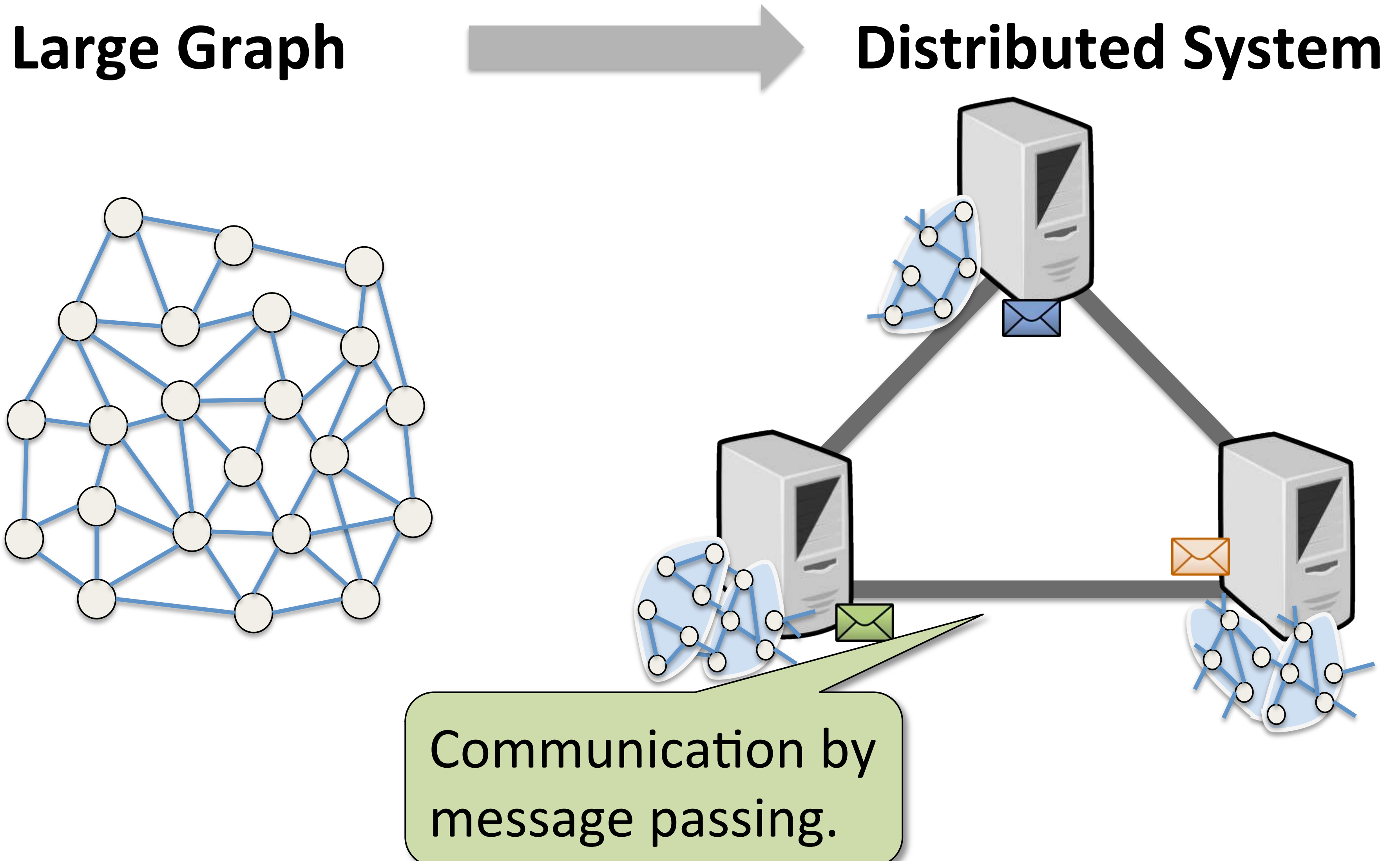
Approach 2:
Build distributed system out of cheaper machines.

Fault-tolerant

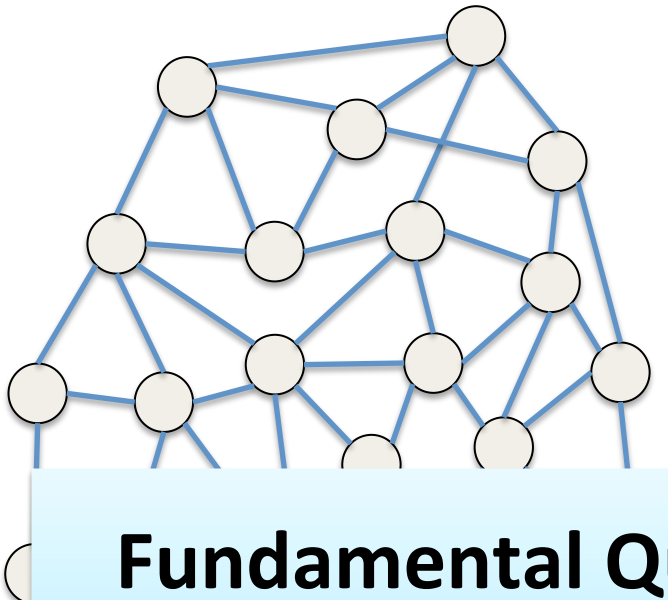
Scalable

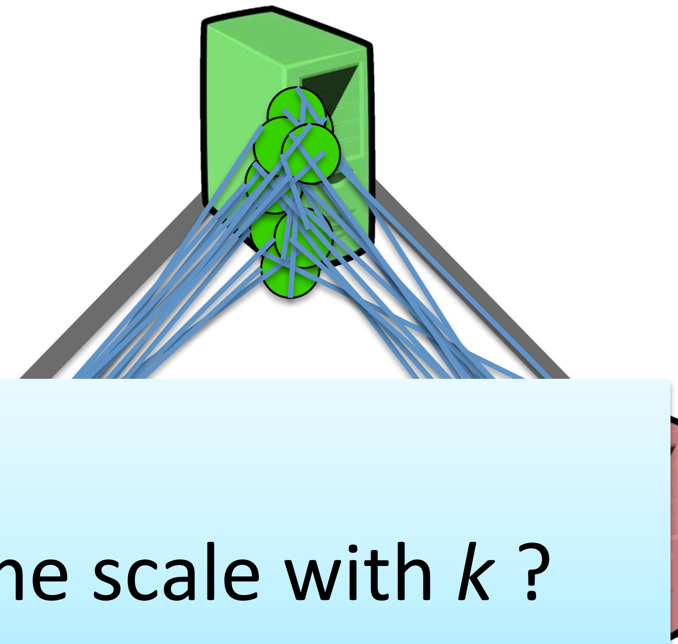


Distributed Large Graph Processing



Partitioning the Input Graph





Fundamental Question:
How does the running time scale with *k* ?

Input Data (Graph)
n vertices, *m* edges

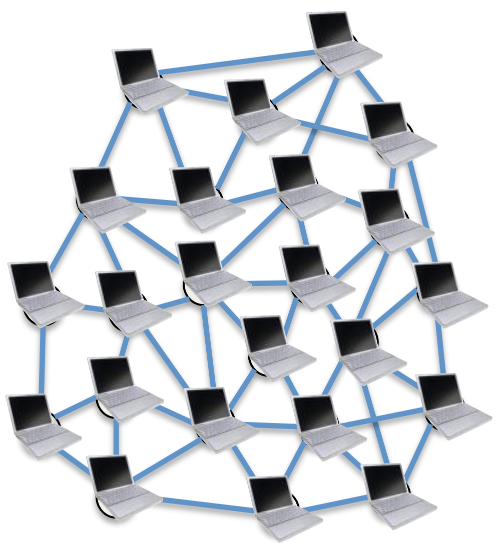
Actual Network
k machines; clique

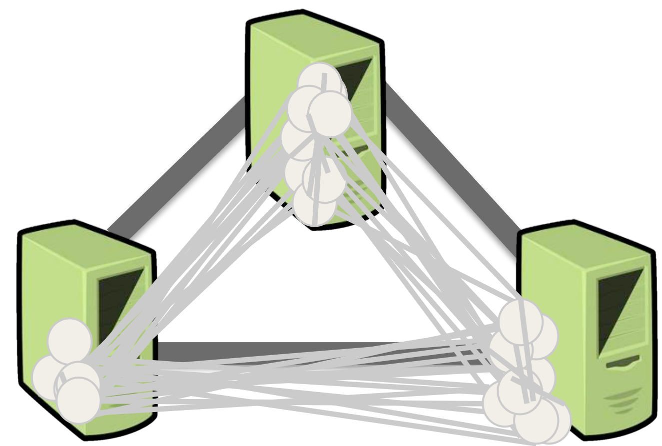
Random Vertex Partitioning

Common practice (e.g. Pregel)

[Woodruff,Zhang'13] Worst case edge partitioning.

Designing algorithms for large graphs





Vertex-Centric Model:

- Vertices "run" algorithm; write code for vertex.
- Input graph = network.
- Classic distributed model

Machine-Centric Model:

- Machines run algorithm; write code for machine.
- Input graph = input data; network = k-clique

Systems for Large Graphs

Pregel & Apache Giraph

- Vertex-centric
- Synchronous message passing

PowerGraph

- Edge-centric model
- Suitable for power-law graphs

GraphLab

- Vertex-centric
- Shared memory abstraction
- Asynchronous

IBM Giraph++

- Extension of Giraph
- machine-centric computation

This talk: Message passing model; vertex/machine-centric

Roadmap

First some preliminaries…

Algorithms

Graph Verification
Connectivity testing

Constructing Trees
BFS Tree, MST

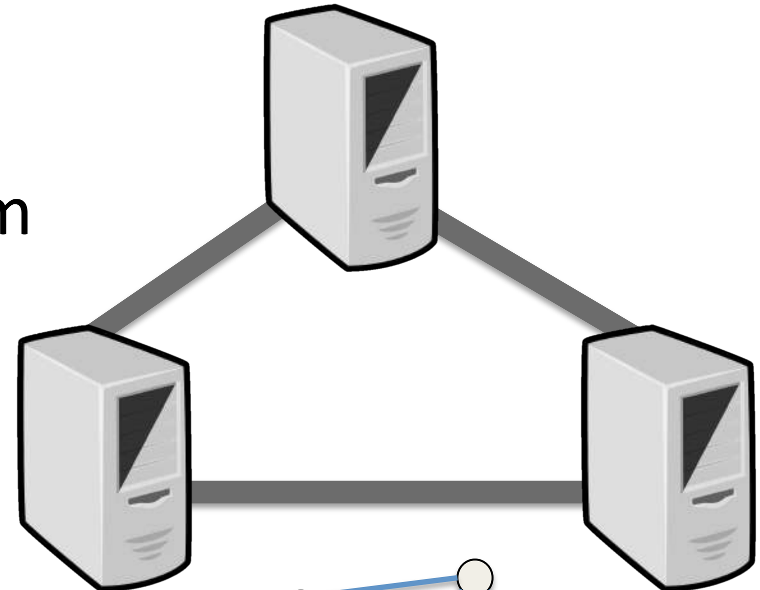
PageRank

Lower Bound Techniques

Communication Complexity

Information Theory

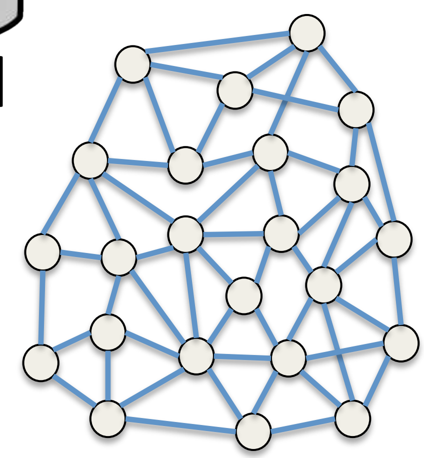
The Distributed *k*-Machine Model



k physical machines running synchronous distributed algorithm

Point-to-point message passing over communication links

Link bandwidth: $O(\log n)$ bits per round



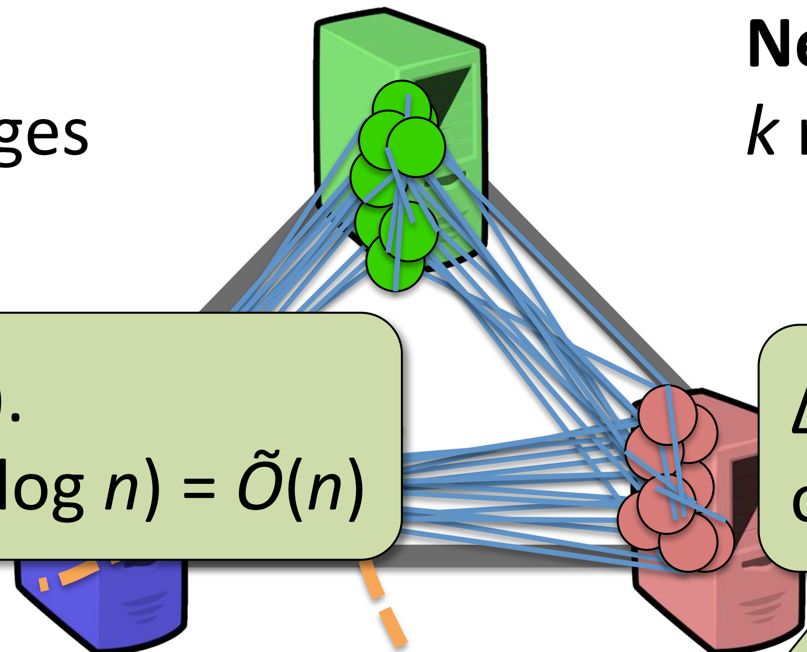
Each machine holds part of large *n*-node input graph.

Machines have local view and no shared memory.

Properties of Random Vertex Partitioning

Input Graph:
n vertices, *m* edges

Network:
k machines



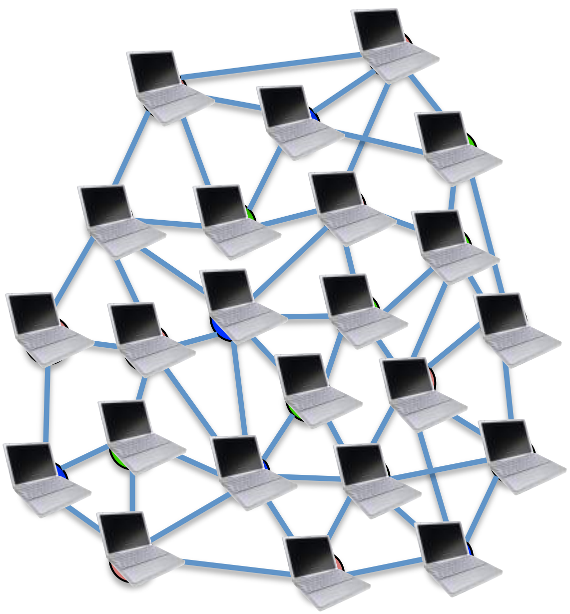
Hides polylog(n).
E.g. $\tilde{O}(n \log^2 n + \log n) = \tilde{O}(n)$

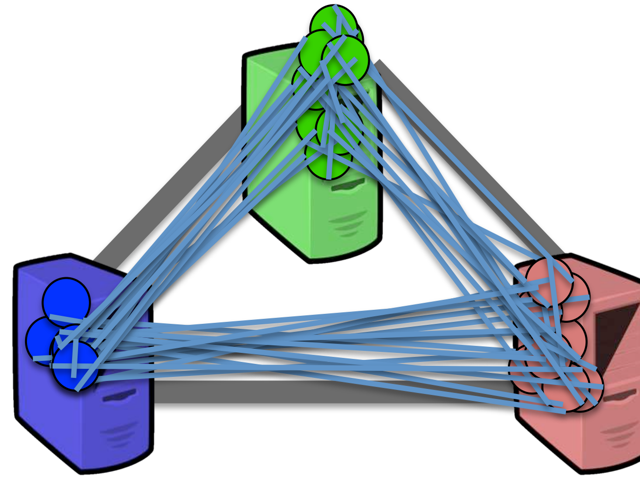
Δ is max degree of input graph

$\tilde{O}(n/k)$ vertices per machine whp;
→ Vertices per machine are balanced.

$\tilde{O}(m/k^2 + \Delta/k)$ edges per link whp;
→ Edges per link are balanced.

Designing Algorithms in k-Machine Model

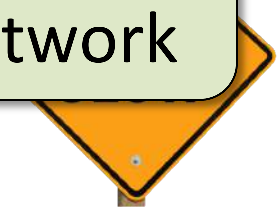




Vertices "run" algorithm:
input graph = distributed network

Trivial Algo: Aggre

Takes $O(n^2/k)$.



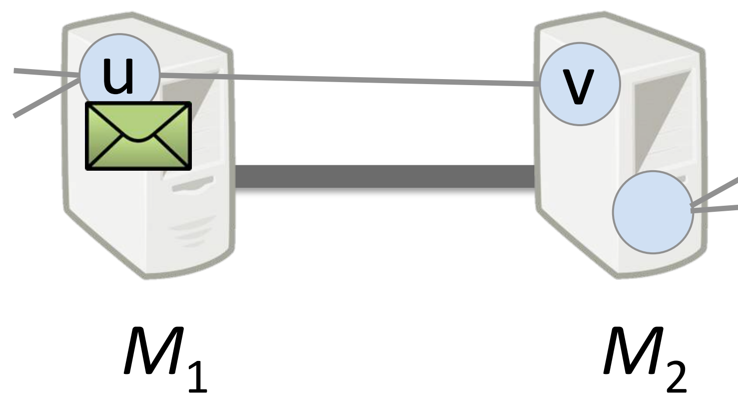
Wealth of algorithms for vertex-centric model.

Simulating Vertex-Centric Algorithms

Suppose we have algorithm for vertex-centric model.

Idea: In *k*-machine model, simulate algorithm on input graph.

Each machine simulates execution on its vertices.



Simulating Message Passing:
When *u* sends  to *v*:
Machine M_1 sends msg (u,v,) to M_2

Simulating Vertex-Centric Algorithms

Performance Measures in Vertex-Centric (VC) Model:

- Message Complexity: M
- Time Complexity: T
- Communication Degree: Δ'

At most Δ' messages sent/rcvd per vertex per round

Conversion Theorem – Part 1

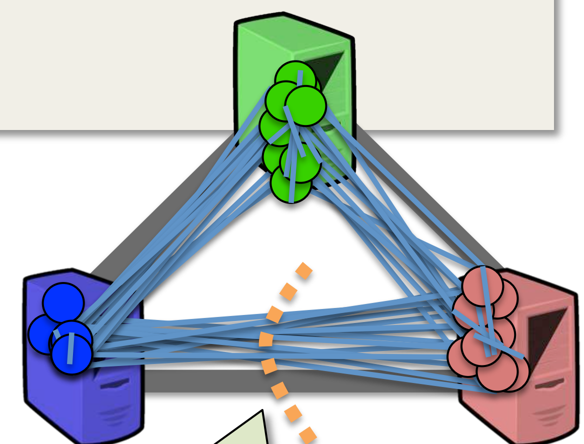
Simulation of vertex-centric algorithm in k-machine model takes $\tilde{O}\left(\frac{M}{k^2} + \frac{T\Delta'}{k}\right)$ rounds.

Efficient algorithm in VC-model ➔ Efficient algorithm in k-machine model

Proof Idea of Conversion Theorem

Conversion Theorem – Part 1

Simulation of vertex-centric algorithm in k-machine model takes $\tilde{O}\left(\frac{M}{k^2} + \frac{T\Delta'}{k}\right)$ rounds.



Recall: Random Vertex Partitioning

→ Edges per link are balanced.

→ In round r, activated edges A_r per link are balanced too!

$M = |A_1| + \cdots + |A_T|$

ges per link.

Simulating 1 round takes $\tilde{O}\left(\frac{|A_r|}{k^2} + \frac{\Delta'}{k}\right)$.

Total time: $\tilde{O}\left(\sum_{i=1}^{T}\left(\frac{|A_r|}{k^2} + \frac{\Delta'}{k}\right)\right)$

Roadmap

Algorithms

Graph Verification
Connectivity testing

Constructing Trees
BFS Tree, MST

PageRank

Lower Bound Techniques

Communication Complexity

Information Theory

Application: Constructing BFS Tree

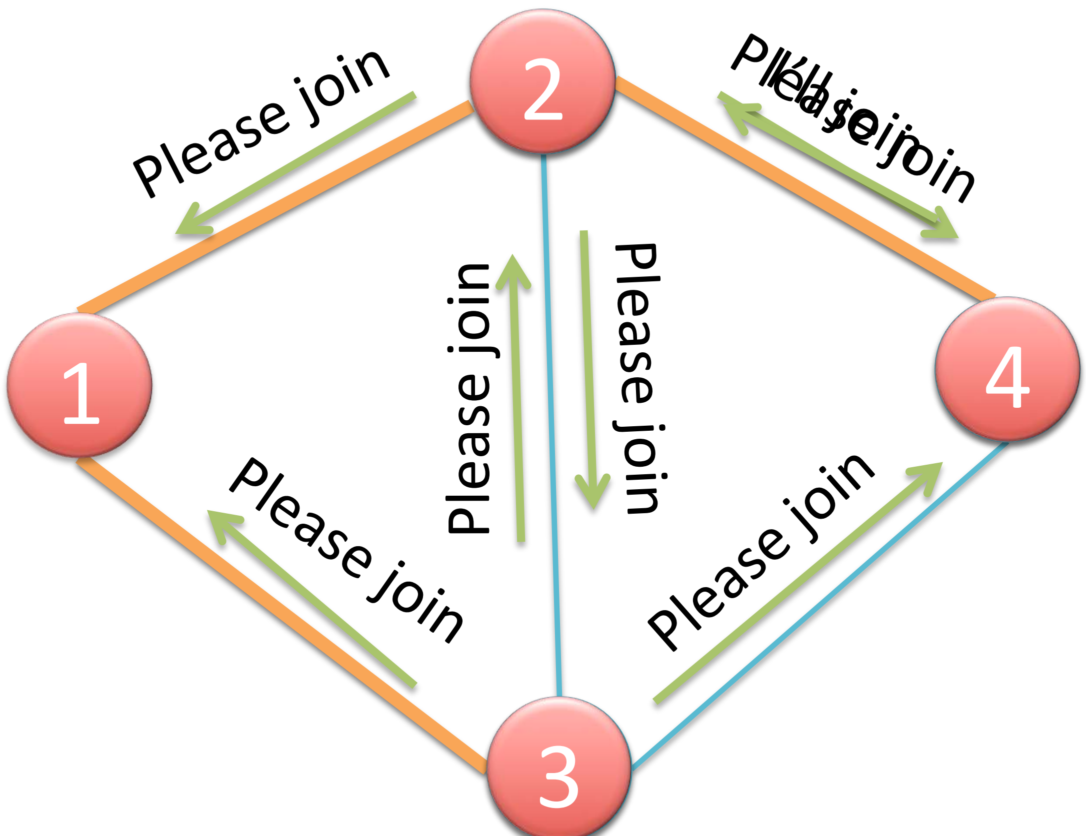
Goal: BFS tree rooted at source node

Vertex-Centric Algorithm:

Application: Constructing BFS Tree

Goal: BFS tree rooted at source node

Vertex-Centric Algorithm:



Application: Constructing BFS Tree

Performance in Vertex-Centric Model:

- Message complexity: $2m$
- Time complexity: diameter $2D$
- Communication degree: $\leq \Delta$

Performance in *k*-Machine Model:

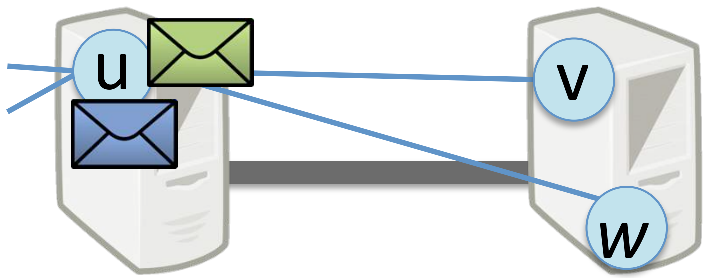
→ $\tilde{O}(m/k^2 + D\Delta/k)$

Conversion Theorem – Part 1

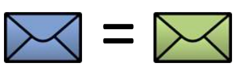
Simulation of vertex-centric algorithm in *k*-machine model takes $\tilde{O}(\frac{M}{k^2} + \frac{T\Delta'}{k})$ rounds.

Our simulation so far:

For each simulated message, we instruct machine to send message.

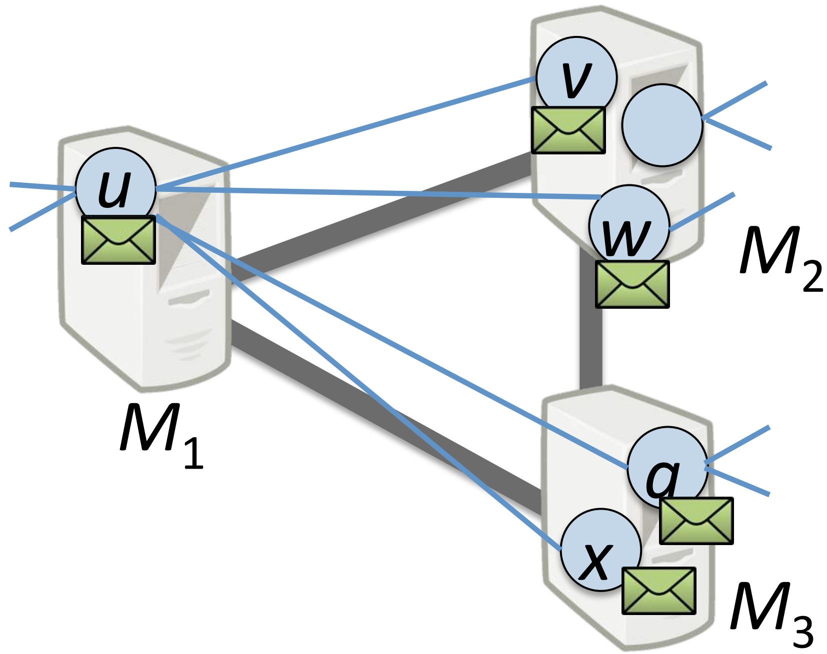


Bandwidth restriction of links is bottleneck.

Not necessary for broadcast algorithms! 

Can we get better bounds for **broadcast algorithms**?

Simulating Broadcast Algorithms



Suppose vertex u broadcasts in some round.

- M_1 sends (u,) to M_2, M_3.
- M_2, M_3 deliver to all local neighbors of u.

→ Simulating 1 broadcast requires $\leq k - 1$ messages.

The Conversion Theorem – Part 2

Performance Measures of Broadcast Algorithms:

- Time Complexity T: running time in vertex-centric model
- Broadcast Complexity B: number of broadcasts

Conversion Theorem – Part 2

Simulation of vertex-centric broadcast algorithm in k-machine model takes $\tilde{O}\left(\frac{B}{k} + T\right)$ rounds.

Intuition: $\tilde{O}(n/k)$ vertices per machine whp.

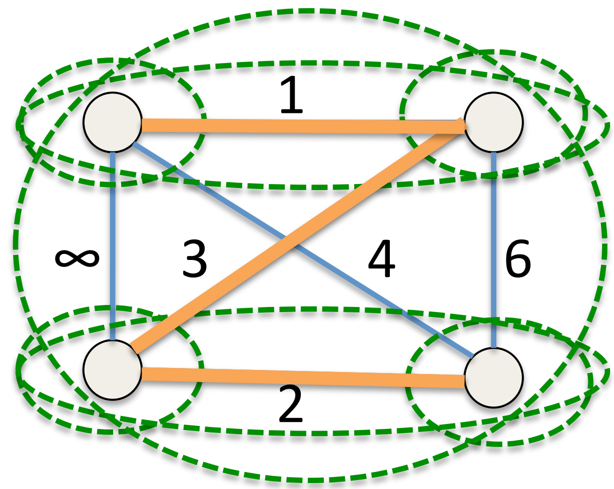
→ Same is true for number of broadcasts B.

Application: Minimum Spanning Tree

Input graph has edge weights.

$O(\log n)$ time algorithm known in vertex-centric clique model.

Pretend input graph is clique:
→ add ∞-weight edges.



Vertex-Centric MST Algorithm:

Initially: every vertex is fragment.

While >1 fragment do:

1. Vertices compute minimum weight outgoing edge (MWOE) of their fragments by broadcast.
2. Add MWOEs to MST.
3. Merge fragments along MWOEs.

Application: Minimum Spanning Tree

Broadcast Complexity?

- Vertices find next outgoing edge of their fragment by broadcasting twice.
- Merging doubles size of fragments.
  → $O(\log n)$ iterations.
- Total number of broadcasts $B = O(n \log n)$.

Conversion Theorem – Part 2

Simulation of vertex-centric broadcast algorithm in machine model takes $\tilde{O}(\frac{B}{k} + T)$ rounds.

→ In machine model: $\tilde{O}(\frac{n \log n}{k} + \log n) = \tilde{O}(n/k)$

Roadmap

Algorithms

Graph Verification
Connectivity testing

Constructing Trees
✓ BFS Tree, MST

PageRank

Lower Bound Techniques

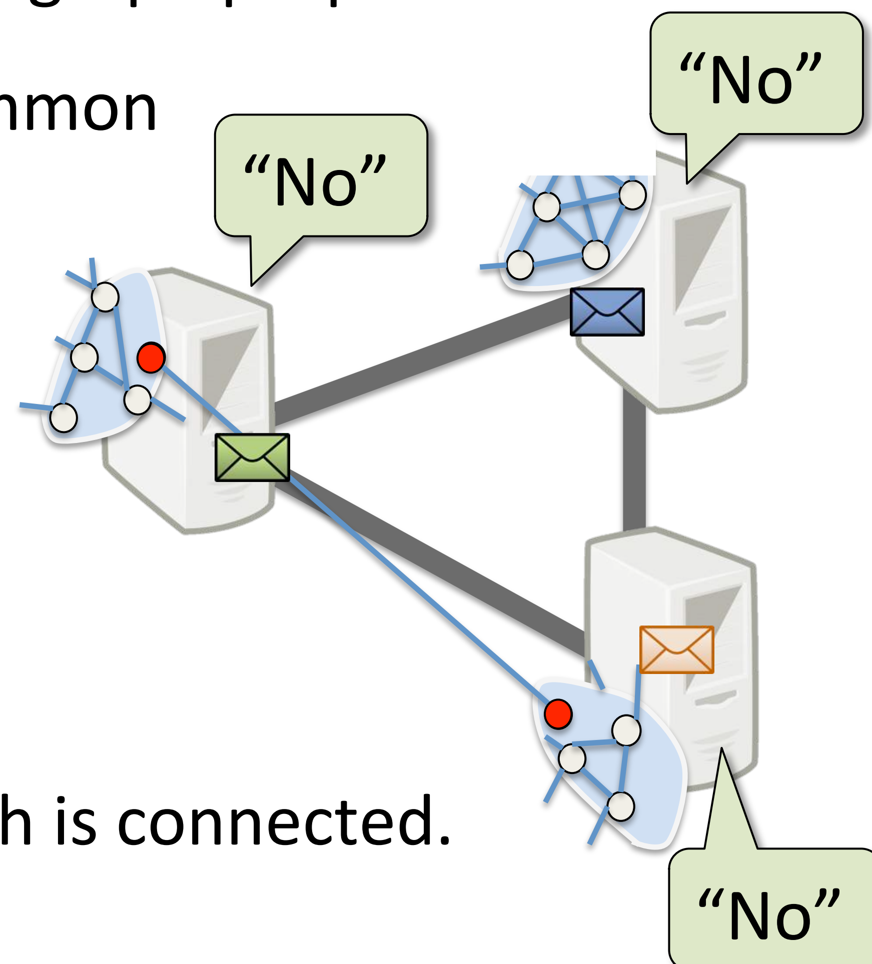
Communication Complexity

Information Theory

Distributed Graph Verification

Goal: Distributed testing of graph properties

Machines must output common answer: "Yes" or "No".

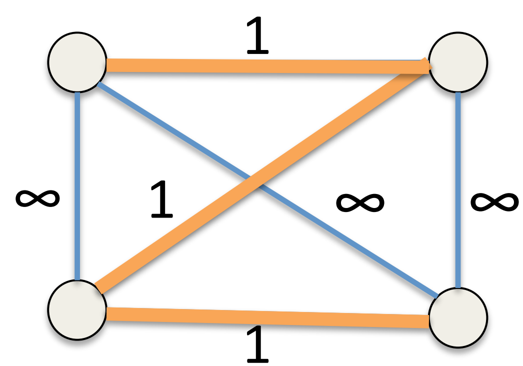


Graph Connectivity:
Output "Yes" iff input graph is connected.

First Attempt: Verification in $\tilde{O}(n/k)$

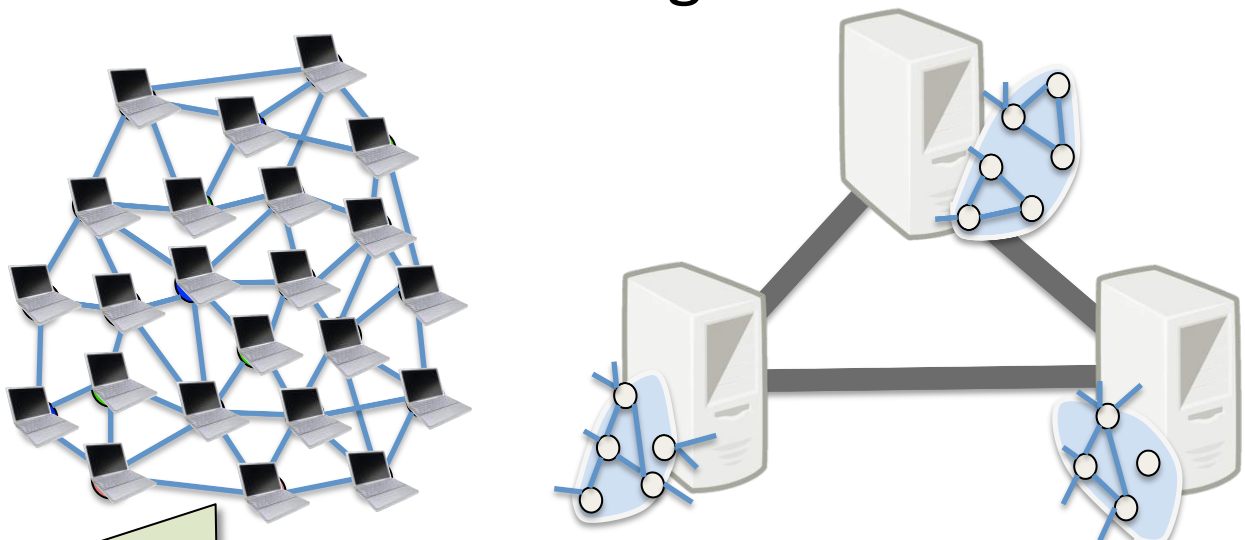
Is input graph connected?

1. Assign ∞ to missing edges.
2. Compute MST.
3. Connected ⇔ MST has finite weight.



Faster Connectivity Testing?

So far: Connectivity testing in $\tilde{O}(n/k)$ rounds based on **vertex-centric** MST algorithm.



Doesn't take advantage of *k*-clique topology.

Can we design faster **machine-centric** algorithms?

Faster Connectivity Testing

$\tilde{O}(n/k^2)$-Time Algorithm

Initially: each vertex is component.

Repeat $\Theta(\log n)$ times:

- For each component **find outgoing edge** to other component.
- **Merge components** into larger components.

Similar to MST alg.

Breaking $\tilde{O}(n/k)$ barrier requires new techniques…

Can we get low messages complexity **per machine**?

Can we merge components efficiently?

Faster Connectivity Testing

$\tilde{O}(n/k^2)$-Time Algorithm

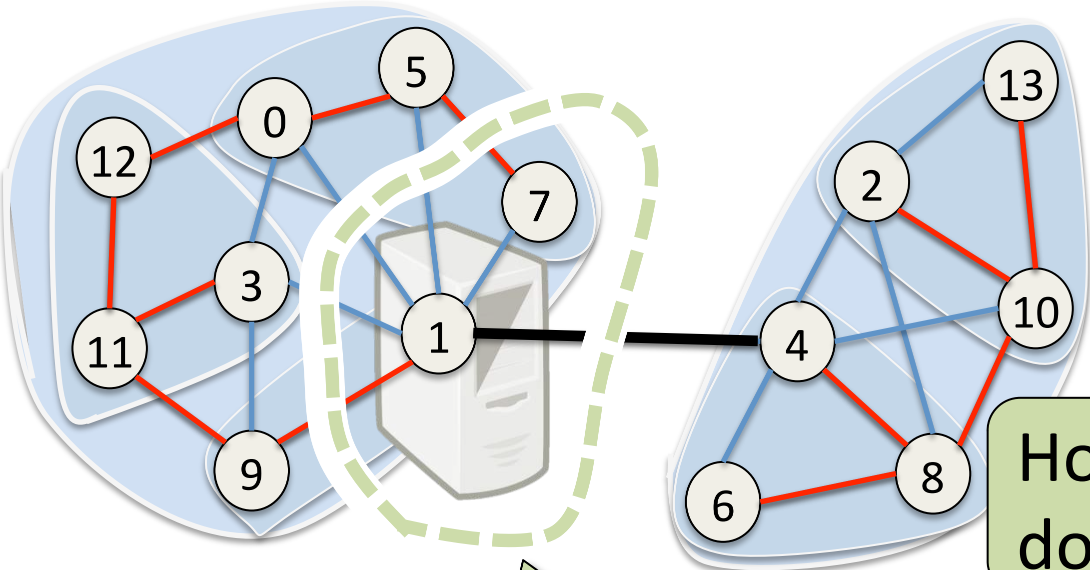
Initially: each vertex is component.

Repeat $\Theta(\log n)$ times:

- For each component **find outgoing edge** to other component.
- **Merge components** into larger components.

Can we get low messages complexity **per machine**?

Finding Outgoing Edges of Components



Finding outgoing edge is easy initially.

Difficult once we have large components.

How much information do we need to find bridge?

Machines have only **local** knowledge!

Finding the … **k…**

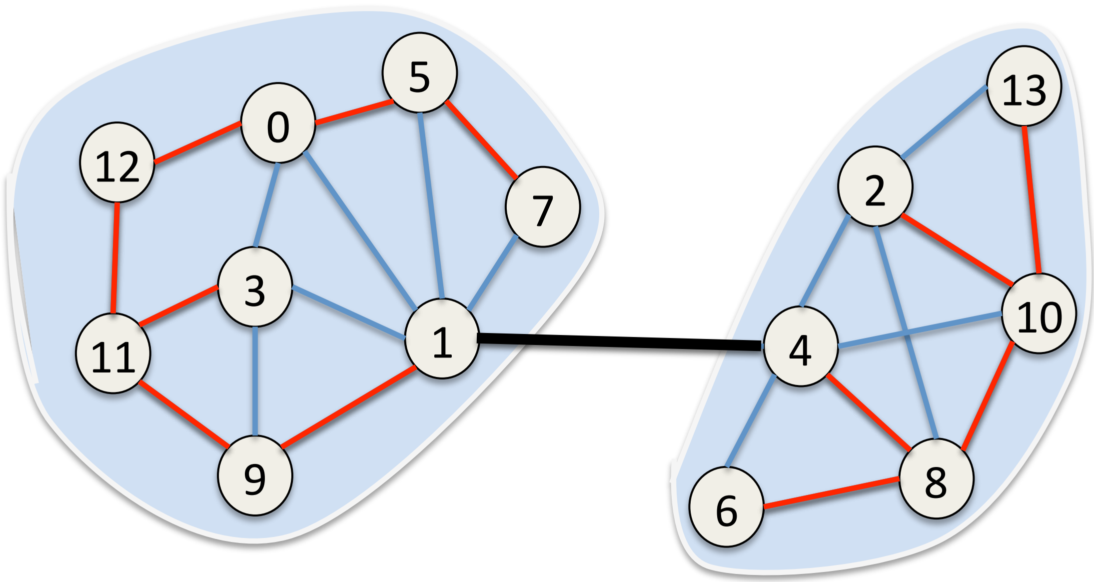
Worst case: … onent.

Most edges lead back into component.

If machine needs $O(n)$ information: takes $O(n/k)$ rounds.

Can we avoid learning about all component members?

Graph Sketches to the Rescue



Sketch of left component: $s = s_0 + s_1 + s_3 + s_5 + s_7 + s_9 + s_{11} + s_{12}$

→ Sample returned by s is bridge edge.

We only need $O(\text{poly} \log n)$ bits to find bridge!

Machines locally compute sketches for their vertices

→ $O(n \text{ poly} \log n)$ messages in total.

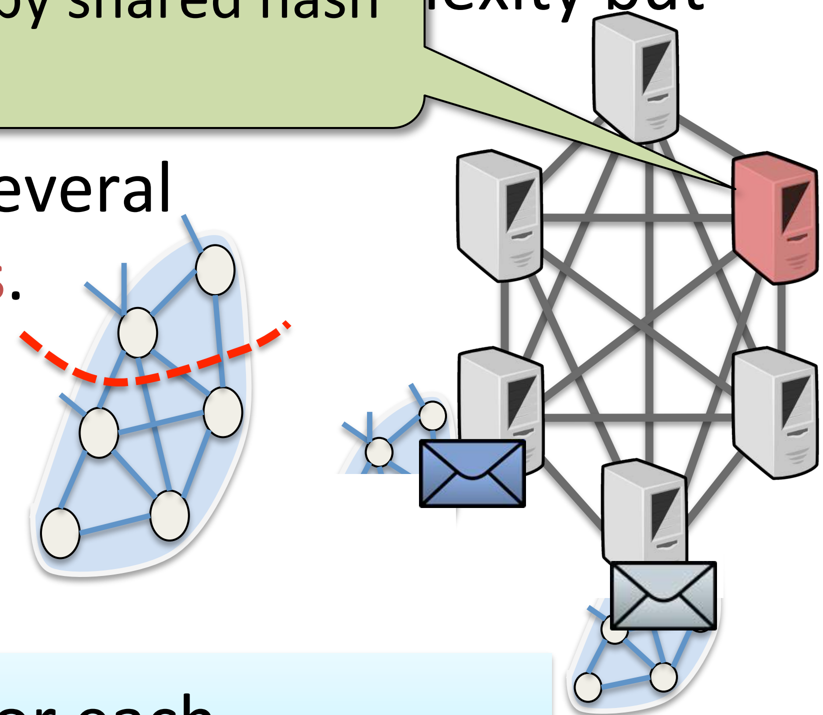
Fast Communication via Random Proxies

Sketches provide overal lexity but load per machine can b

Single component split across several machines into component parts.

For each component:

Choose "almost" random machine as proxy.



Each machine can send sketch for each component part to proxy in $\tilde{O}(n/k^2)$ rounds.

Intuition: Random choice of proxies ensures all k^2 links are used equally. No dependence on graph topology.

Faster Connectivity Testing

$\tilde{O}(n/k^2)$-Time Algorithm

Initially: each vertex is component.

Repeat $\Theta(\log n)$ times:

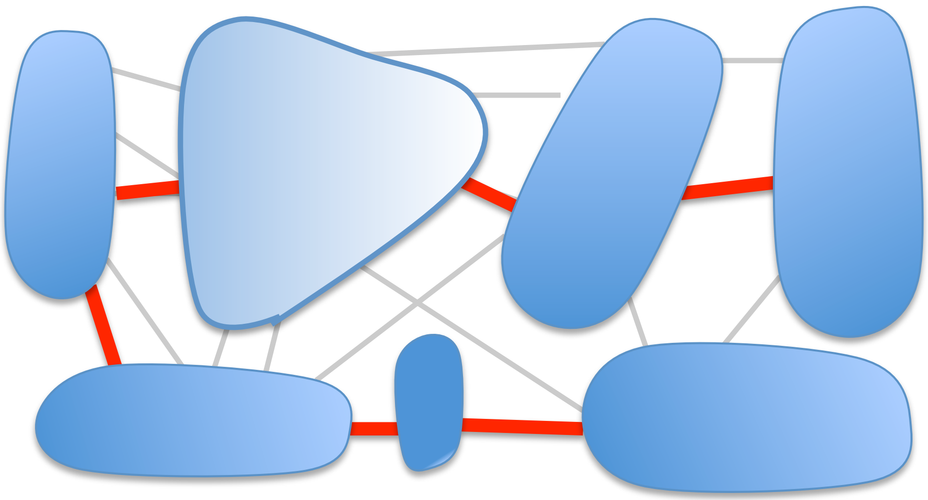
- For each component **find outgoing edge** to other component.
- **Merge components** into larger components.

Can we get low messages complexity **per machine**?

Can we merge components efficiently?

Merging Components

Each component has outgoing edge to other component.



Merge components into single component along chosen edges.

Problem: Induced paths might have Θ(*n*) length!

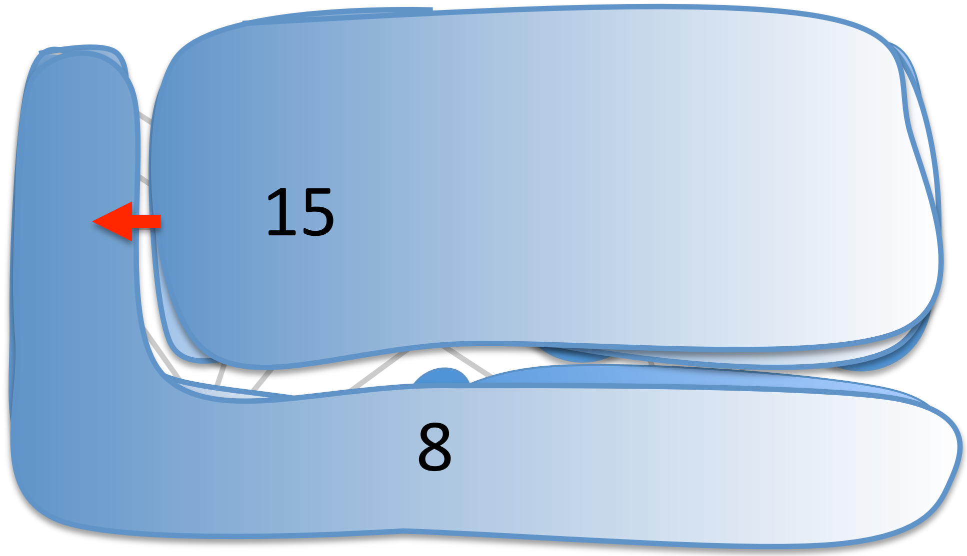
→ Merging Θ(*n*) components too costly…

Building Merge Trees

Goal: Merge all components in $O(\log n)$ steps.

For each component:

1. Choose random rank.
2. Keep outgoing-edge if endpoint rank higher



Ranking yields directed trees of $O(\log n)$ depths.

Repeatedly merge leafs with their parents.

Faster Connectivity Testing

$\tilde{O}(n/k^2)$-Time Algorithm

Initially: each vertex is component.

Repeat $\Theta(\log n)$ times:

- For each component **find outgoing edge** to other component.
- **Merge components** into larger components.

$O(\log n)$ iterations sufficient to identify connected components of input graph.

Each phase takes $\tilde{O}(n/k^2)$ → $\tilde{O}(n/k^2)$ rounds in total.

Roadmap

Algorithms

Graph Verification

✓ Connectivity testing

Constructing Trees

✓ BFS Tree, MST

PageRank

Lower Bound Techniques

Communication Complexity

Information Theory

Time Lower Bound for Connectivity

Connectivity verification takes $\Omega(n/k^2 \log n)$ rounds.

Reduction from Set Disjointness Problem (DISJ) in 2-party communication complexity.

Proof Idea:

1. Show DISJ has high communication complexity

2. Solve DISJ i[...]party model by s[...]
   k-machine [...]orithm.

3. Connectiv[...]f information

Random vertex partitioning!

Bandwidth restriction on links!

The Set Disjointness Problem

Universe: set of *n* elements.

Input: *n*-bit vectors *X*, *Y*.

Alice gets *X*

Bob gets *Y*

Alice and Bob output “yes”

⇔ there is no *i*: *X*[*i*] = *Y*[*i*] = 1.

X	***Y***
0	1
0	0
1	0

Jointly compute function of (X,Y)

ce

Bob

How many bits?

Classic 2-party model:

Alice only knows *X* (nothing of *Y*)

Bob only knows *Y* (nothing of *X*)

The Set Disjointness Problem

Simulation requires X, Y to be assigned randomly.

Input graph **randomly** assigned to machines

Random Partition (RP) Model

- Alice knows all of X.
  Bob knows all of Y.
- Each bit of X, Y is randomly r
  either Alice or Bob with prob

Same as classic model.

Communication complexity of Set Disjointness in random partition model is $\Omega(n)$.

Time Lower Bound for Connectivity

Every Connectivity algorithm takes $\Omega(n/k^2 \log n)$ rounds.

Reduction from Set Disjointness Problem (DISJ) in 2-party communication complexity.

Proof Idea:

✓ 1. Show DISJ has high communication complexity under *random* input partitioning.

2. Solve DISJ in 2-party model by simulating k-machine connectivity algorithm.

3. Connectivity requires lots of information → many rounds.

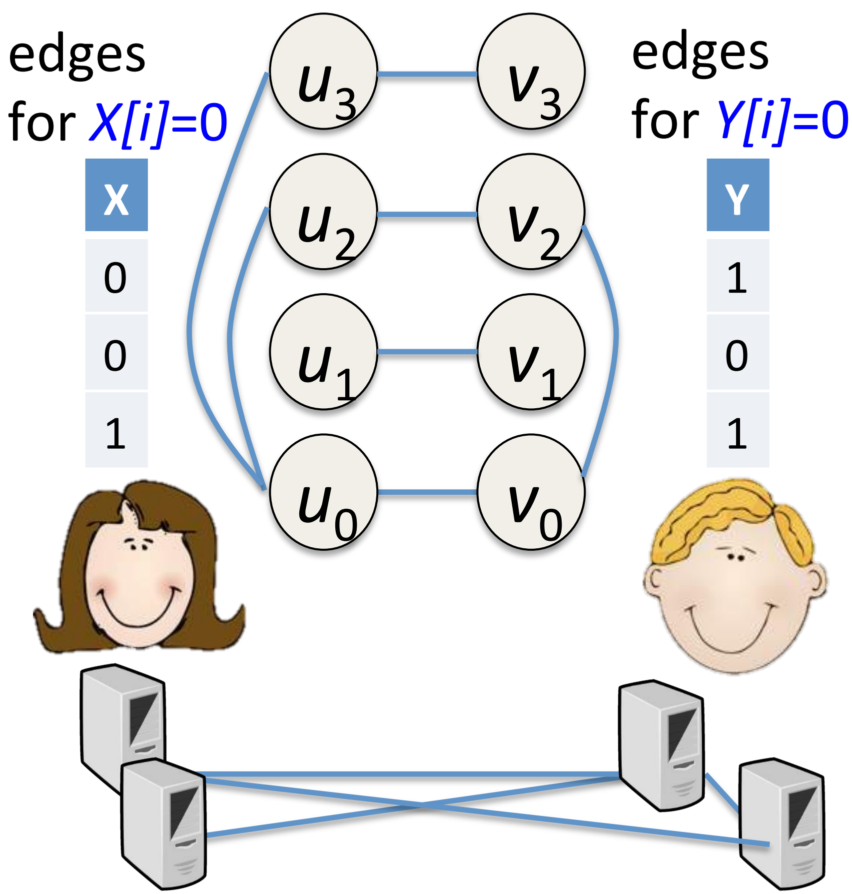
Solving Disjointness via Connectivity

Simulate Connectivity algorithm of *k*-machine model in 2-party model.

Input: DISJ instance.
Randomly assigned vectors: X, Y.
Alice and Bob:

- Construct Graph(X,Y).
- Simulate k/2 machines each
- Create vertex partition via shared randomness
- If u_0,v_0 on same machine:
  return “No”
- Run Connectivity algorithm:
  Use output to decide DISJ

connected ⇔ *X*, *Y* disjoint



Time Lower Bound for Connectivity

Every Connectivity algorithm takes $\Omega(n/k^2 \log n)$ rounds.

Reduction from Set Disjointness Problem (DISJ) in 2-party communication complexity.

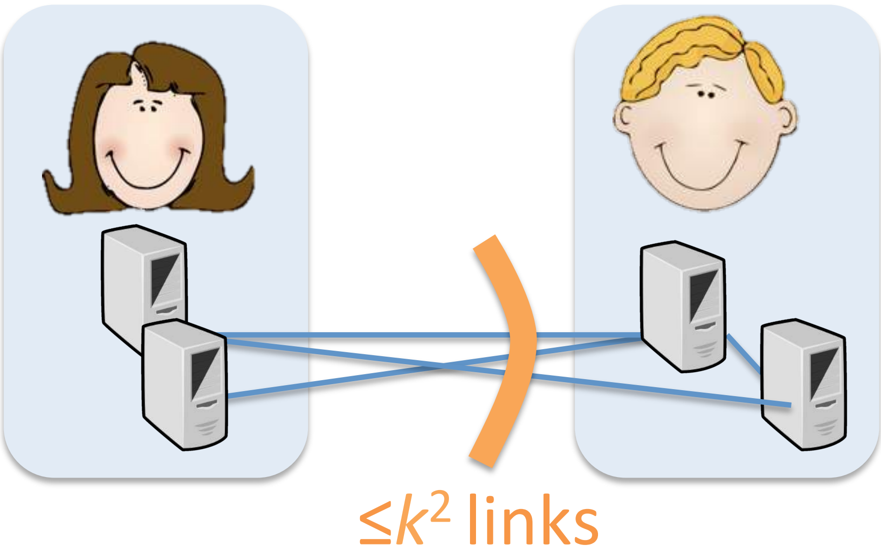
Proof Idea:

✓ 1. Show DISJ has high communication complexity under *random* input partitioning.

✓ 2. Solve DISJ in 2-party model by simulating k-machine connectivity algorithm.

3. Connectivity requires lots of information → many rounds.

High Communication → Many Rounds



Alice and Bob each simulate ½k machines.

Connectivity algorithm solves Set Disjointness.

Communication complexity of Set Disjointness in random partition model is $\Omega(n)$.

How much

Bob’s machines?

Each round of simulation generates $\leq k^2 \log n$ bits.

→ Connectivity algorithm takes $\Omega(n/k^2 \log n)$ rounds.

Roadmap

Algorithms

Graph Verification

✓ Connectivity testing

Constructing Trees

✓ BFS Tree, MST

PageRank

Lower Bound Techniques

✓ Communication Complexity

Information Theory

Distributed PageRank Computation

Goal: Machines output PageRank for their vertices.

Distributed Vertex-Centric PageRank Algorithm:

Each vertex starts $\Theta(\log n)$ random walks
(Generates $\Theta(\log n)$ tokens.)

Random walk step = send token

At each step of token: terminate with prob ε
continue with prob $1 - \varepsilon$

All walks terminate in
$O(\log n / \varepsilon)$ steps w.h.p.

→ Vertex u outputs $\text{PageRank}(u) = \#(\text{visits to } u)\ \varepsilon / \Theta(n \log n)$

Distributed PageRank Computation

Distributed Vertex-Centric PageRank Algorithm:

Each vertex starts $\Theta(\log n)$ random walks.
(Generates $\Theta(\log n)$ tokens.)

Conversion Theorem

Simulation of A on input graph in k-machine model takes $\tilde{O}(\frac{M}{k^2} + \frac{T\Delta'}{k})$ rounds.

Total message complexity: $M = \Theta(n \log^2 n)$

Total time complexity: $T = O(\log n)$

Communication degree: $\Delta' = n - 1$

In the k-machine model: $\tilde{O}(\frac{n \log^2 n}{k^2} + \frac{n \log n}{\epsilon k}) = \tilde{O}(\frac{n}{\epsilon k})$

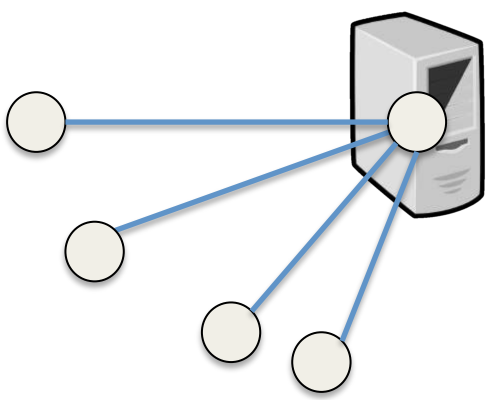
Faster PageRank Computation

Distributed Vertex-Centric PageRank Algorithm:

Each vertex starts $\Theta(\log n)$ random walks.
(Generates $\Theta(\log n)$ tokens.)

Random walk step = send token to random neighbor

At each step of token: terminate with prob ε
continue with prob $1 - \varepsilon$



Tokens per machine $\Theta(n)$.
→ $\Theta(n/k)$ rounds unavoidable?

Faster PageRank Computation

[work in progress]

Distributed Machine-Centric PageRank Algorithm:

Each vertex starts $\Theta(\log n)$ random walks.
(Generates $\Theta(\log n)$ tokens.)

Random walk step:

- Combine tokens to *u*'s neighbors on same machine by sending only their count.
- Send tokens via proxy machines

At each step of token: terminate with prob ε
continue with prob $1 - \varepsilon$

Faster PageRank Computation
[work in progress]

T_u = expected number of tokens at vertex u
(in specific round r).

Group vertices into bins wrt T_u:

$B_i = \{ u \mid k / 2^{i+1} \leq T_u \leq k / 2^i \}$

$|B_i| \leq \tilde{O}(2^{i+1} n / k)$

B_i-sets are distributed randomly.

→ Each machine M has $\tilde{O}(2^{i+1} n / k^2)$ vertices from B_i.

M has per-round-capacity of k–1 tokens.

→ Sending all tokens for $v \in B_i$ requires $1/2^i$-fraction of capacity.

→ Sending all tokens of B_i takes $2^{-i} 2^{i+1} n / k^2 = \tilde{O}(n/k^2)$ time.

Roadmap

Algorithms

Graph Verification
✓ Connectivity testing

Constructing Trees
✓ BFS Tree, MST

✓ **PageRank**

Lower Bound Techniques

✓ Communication Complexity

Information Theory

Lower Bound on Finding Spanning Trees

Spanning Tree Construction:

- Each machine outputs list of incident tree edges.
- **Goal**: machine outputs form spanning tree.

How fast can we find a spanning tree of the input graph?

Huh!? I just showed you $\tilde{O}(n/\boldsymbol{k}^2)$ algorithm for connectivity (and ST)

$\tilde{O}(n/k)$ rounds optimal for constructing **any** spanning tree!

Assumption: **both** machines holding endpoint vertices output edge if edge is in ST.

The Hard Input Graph

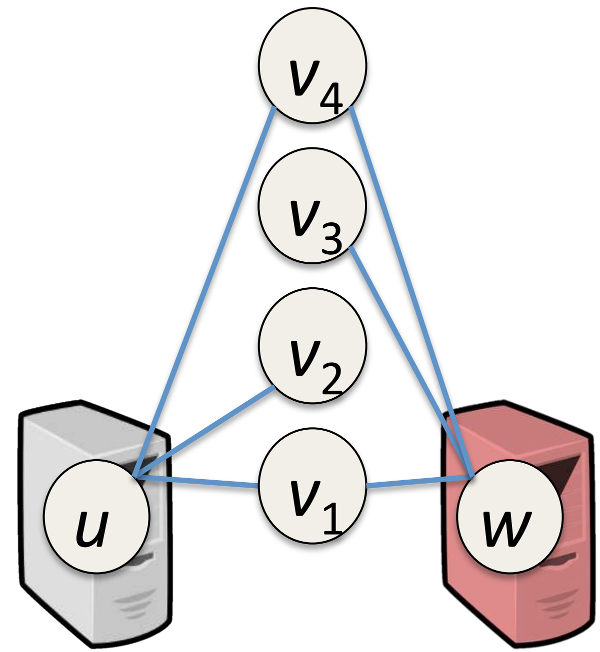
Vertices:

- outer vertices: u, w
- n inner vertices $v_1,\ldots,v_n$

Edges:

- Chosen by random n-bit vectors X, Y.
- Restriction: $X[i] + Y[i] \geq 1$.

Ensures connectivity



n=4 X={1,1,0,1}

Y={1,0,1,1}

Intuition:

Every spanning tree has ≤ $n/2$ edges of either u or w.

→ High uncertainty wrt which edges to include in ST.

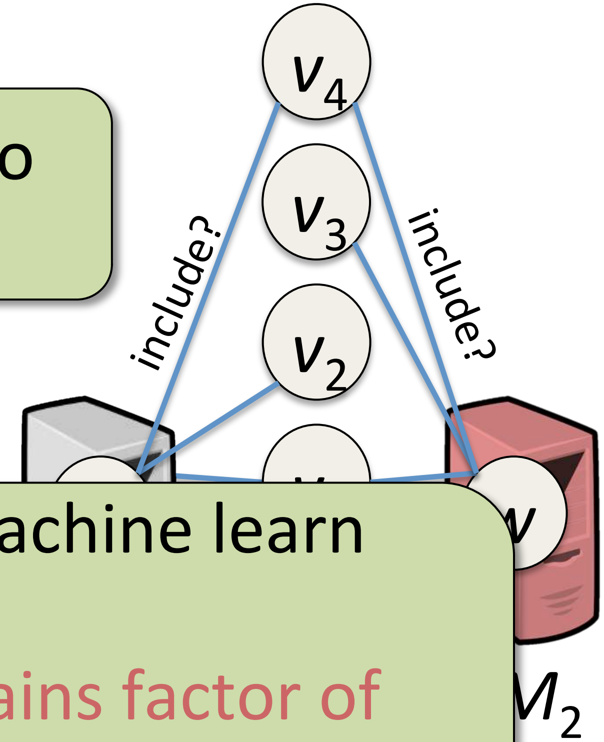
Information Theoretic Lower Bound

After partitioning graph:

u, w likely to be o

machines; probab

And M_2 needs to learn about X.



M_1 needs to learn about Y to

corre

How much information can machine learn per round?

- Synchronous rounds
- Distinct ports

Gains factor of $O(1 / \log n \log k)$.

M_1 g

M_1's

H(Y

Eventually: H(Y | InitialKnowl$_1$) < ½ n

→ I(Y; Transcr$_1$ | InitialKnowl$_1$) = $\Omega(n)$.

→ $\Omega(n / (k \log^2 n \log k))$ rounds.

More on Information Theoretic LBs

Works best for problems where output per machine is large.

More involved LB proof for **triangle enumeration** problem [work in progress]

If there are *t* triangles in input graph (sampled from Rusza-Szemeredi graph), some machine outputs ≥ *t* / *k*.

Show: initial knowled

information about ex

Best upper bound: $\tilde{O}(n^2 / k^{5/3})$
(D.Dolev, Lenzen, Peled DISC'12)

→ $\Omega(m / (k^2 \log^2 n \log k))$ rounds for triangle enumeration

Wrap-up

Is there a conversion theorem for getting $\tilde{O}(n / k^2)$ or $\tilde{O}(m / k^2)$ type bounds?

Machine-centric vs Vertex-centric algorithm design

Fault-Tolerance?

Impact of partitioning of graph data?

Theory meets Practice: Implementing algorithms in Apache Giraph, Spark/GraphX