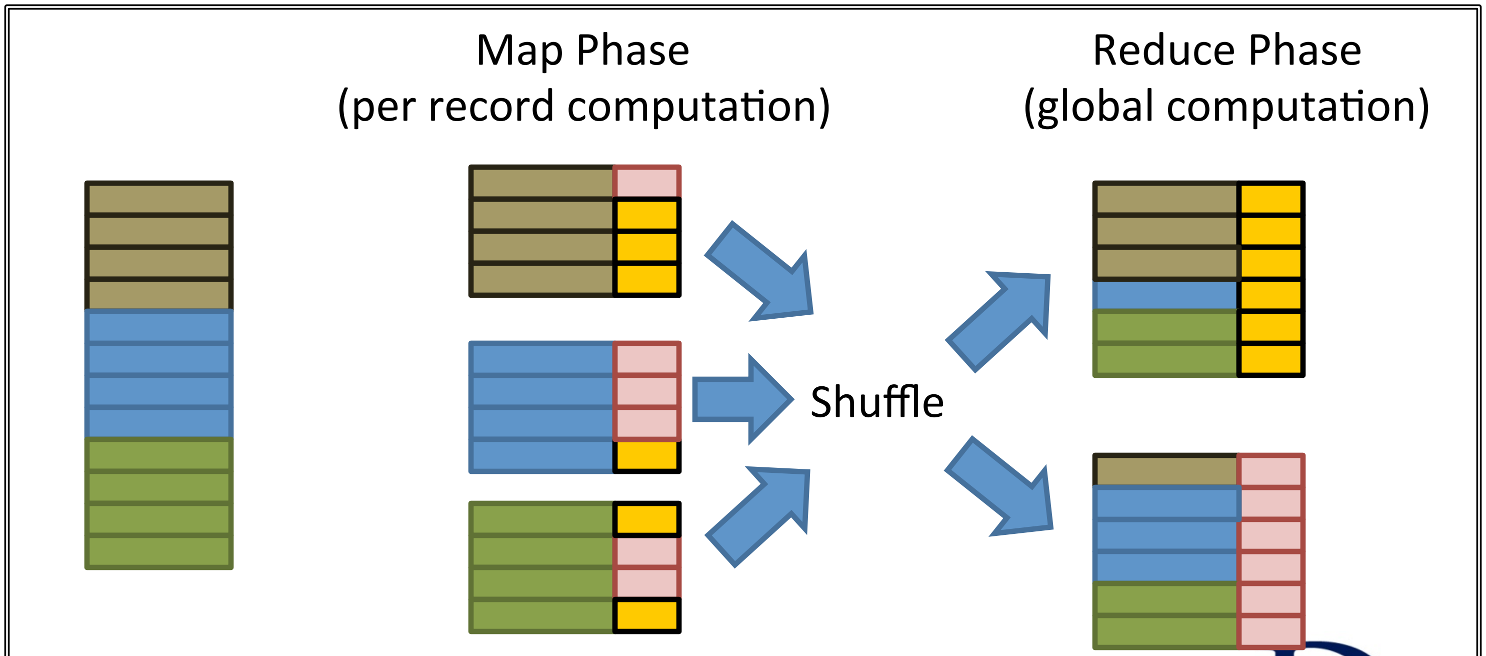
Graph Algorithms & Iteration on Map-Reduce

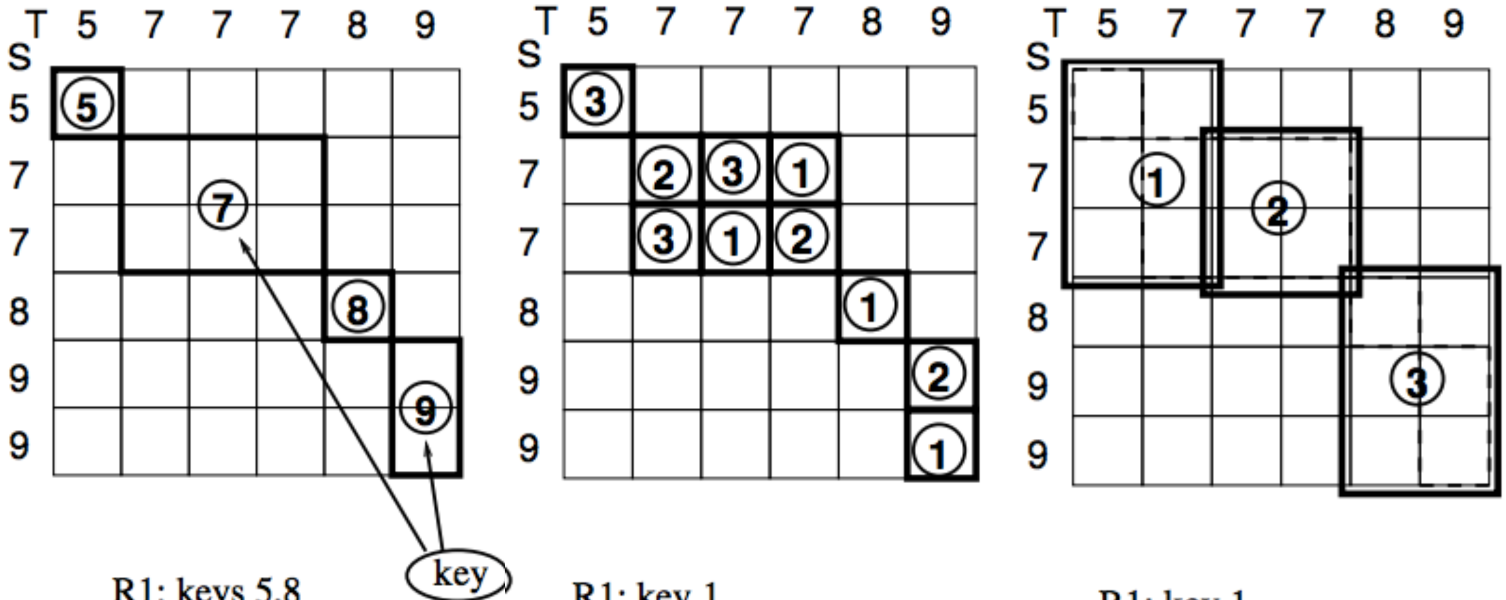
CompSci 590.03
Instructor: Ashwin Machanavajjhala

Recap: Map-Reduce

```
map     (k1,v1)        → list(k2,v2);
reduce  (k2,list(v2))  → list(k3,v3).
```



Recap: Optimizing Joins



R1: keys 5,8
Input: S1,S4
T1,T5
Output: 2 tuples

R2: key 7
Input: S2,S3
T2,T3,T4
Output: 6 tuples

R3: key 9
Input: S5,S6
T6
Output: 2 tuples

max–reducer–input = 5
max–reducer–output = 6

R1: key 1
Input: S2,S3,S4,S6
T3,T4,T5,T6
Output: 4 tuples

R2: key 2
Input: S2,S3,S5
T2,T4,T6
Output: 3 tuples

R3: key 3
Input: S1,S2,S3
T1,T2,T3
Output: 3 tuples

max–reducer–input = 8
max–reducer–output = 4

R1: key 1
Input: S1,S2,S3
T1,T2
Output: 3 tuples

R2: key 2
Input: S2,S3
T3,T4
Output: 4 tuples

R3: key 3
Input: S4,S5,S6
T5,T6
Output: 3 tuples

max–reducer–input = 5
max–reducer–output = 4

This Class

- Graph Processing on Map Reduce

Duke
UNIVERSITY

GRAPH PROCESSING

Graph Algorithms

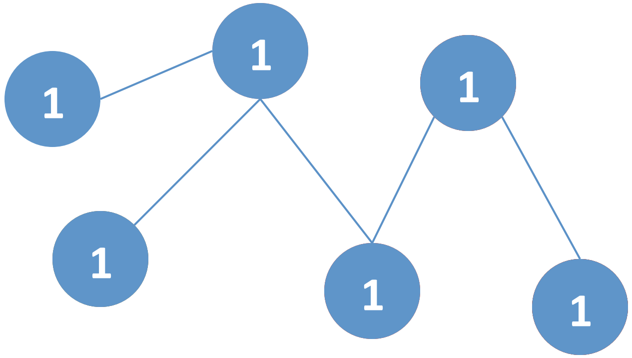
- Diameter Estimation
  - Length of the longest shortest path in the graph
- Connected Components
  - Undirected s-t connectivity (USTCON): check whether two nodes are connected.
- PageRank
  - Calculate importance of nodes in a graph
- Random Walks with Restarts
  - Similarity function that encodes proximity of nodes in a graph

Connected Components

- What is an efficient algorithm for computing the connected components in a graph?

HCC [Kang et al ICDM '09]

- Each node's label l(v) is initialized to itself
- In each iteration
  l(v) = min {l(v), min $_{y\ \varepsilon\ neigh(v)}$ l(y)}



- O(d) iterations (d = diameter of the graph)
  O(|V| + |E|) communication per iteration

Duke UNIVERSITY

GIM-V

- Generalized Iterative Matrix-Vector Multiplication

Connected Components

- Let c^h denote the component-id of a vertex in iteration h

- $c^{h+1} = M \times_G c^h$
  - $c^{h+1}[i] = \min(c^{h+1}[i], c^{new}[i])$
  - $c^{new}[i] = \min_j(m[i,j] \times c^h[j])$

Step 1: Generate m[j,j] x c[j]
Step 2: Aggregate to find the min for each node

- Keep iterating till $c^{h+1} = c^h$.

Duke UNIVERSITY

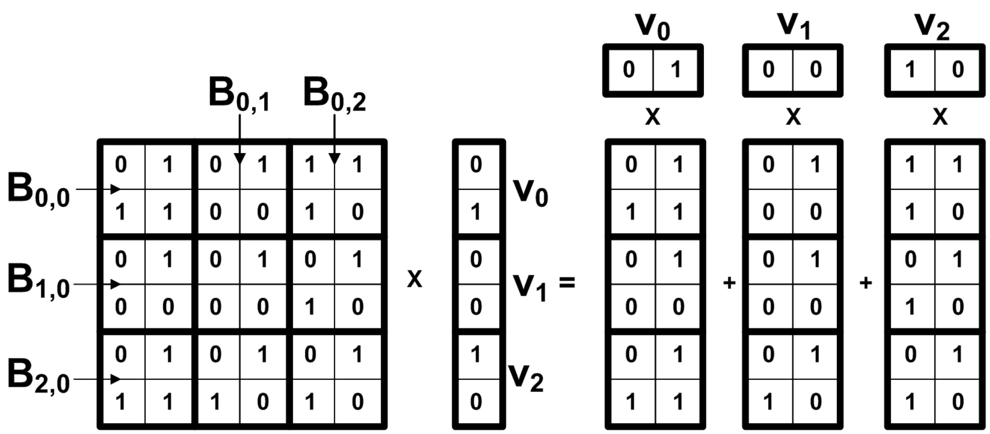
GIM-V and Page Rank

$$p = (cE^T + (1-c)U)p$$

- $p^{next} = M \times_G p^{cur}$
- $p^{next}[i] = (1\text{-}c)/n + sum_j (c \times m[i,j] \times p^{cur}[j])$

GIM-V BL

- We assumed each edge in the graph is represented using a different row.
- Can speed up processing if each row represents a bxb sub matrix



Duke UNIVERSITY

Connected Components

- Iterative Matrix Vector products need O(d) map reduce steps to find the connected components in a graph.

- Diameter of a graph can be large.
  - > 20 for many real world graphs.

- Each map reduce step requires writing data to disk + remotely reading data from disk (I/O + communication)

- Can we find connected components using a smaller number of iterations?

Hash-to-all

- Maintain a cluster at each node
  - Current estimate of connected component
- Initialize cluster(v) = Neighbors(v) U {v}
- Each node sends its cluster to all nodes in the cluster
  - Map: (v, C(v)) → {(u, C(v))} for all u in C(v)
- Union all the clusters sent to a node v
  - Reduce: (u, {C1, C2, …, Ck}) → (u, C1 U C2 U … U Ck)

Hash-to-all

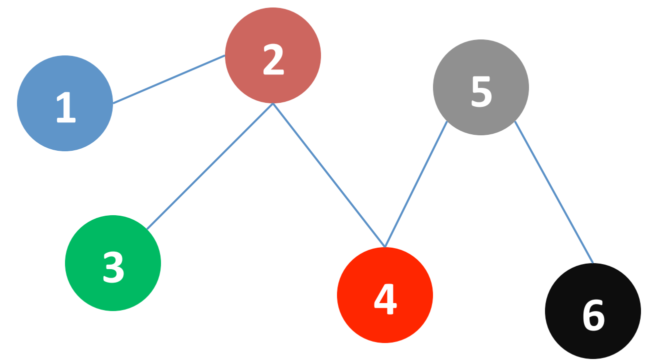
- Number of rounds = log d
  - Proof?

- Communication per round = $O(n|V| + |E|)$
  - Each node is replicated at most n times, where n is the maximum size of a connected component.

Duke UNIVERSITY

Hash-to-Min

- Each node *v* maintains a cluster C(*v*) which is initialized to
  {*v*} U Neighbors(*v*)



- In each iteration

  Map:

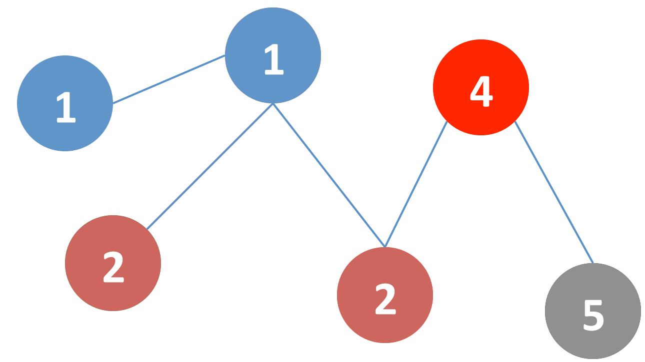
  v_{min} = min {C(v)}
  Send C(v) to v_{min}
  Send v_{min} to nodes in C(v)

  Reduce:

  C(v) is the union of all incoming clusters

Hash-to-Min

- Each node *v* maintains a cluster C(*v*) which is initialized to
  {*v*} U Neighbors(*v*)



- In each iteration

  Map:

  v_{min} = min {C(v)}
  Send C(v) to v_{min}
  Send v_{min} to nodes in C(v)

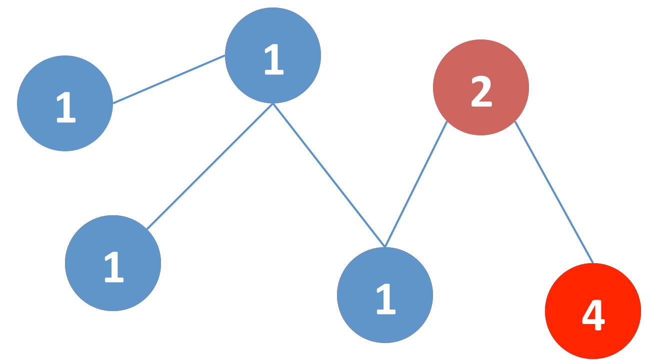
  Reduce:

  C(v) is the union of all incoming clusters

v	C(v)
1	1,2
2	1,2,3,4
3	2,3
4	2,4,5
5	4,5,6
6	5,6

Hash-to-Min

- Each node *v* maintains a cluster C(*v*) which is initialized to
  {*v*} U Neighbors(*v*)



- In each iteration

  Map:

  v_{min} = min {C(v)}

  Send C(v) to v_{min}

  Send v_{min} to nodes in C(v)

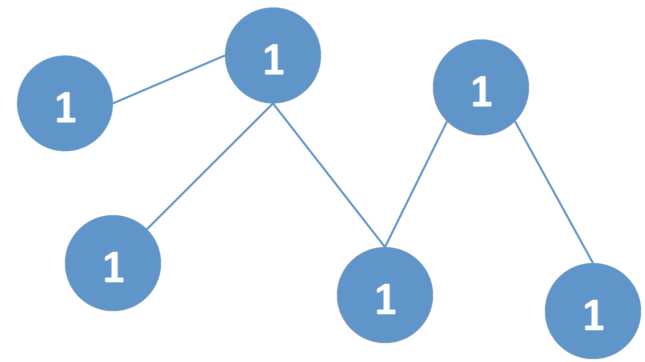
  Reduce:

  C(v) is the union of all incoming clusters

v	C(v)
1	1,2,3,4
2	1,2,3,4,5
3	1
4	1,4,5,6
5	2
6	4

Hash-to-Min

- Each node *v* maintains a cluster C(*v*) which is initialized to
  {*v*} U Neighbors(*v*)



- In each iteration

  Map:

  v_{min} = min {C(v)}
  Send C(v) to v_{min}
  Send v_{min} to nodes in C(v)

  Reduce:

  C(v) is the union of all incoming clusters

v	C(v)
1	1,2,3,4,5,6
2	1
3	1
4	1
5	1
6	1

Hash-to-Min

- In the end, cluster of vertex with minimum id contains the entire connected component.
  Cluster of other vertices in the component is a singleton having the minimum vertex.

- Communication cost: Assuming a random assignment of ids to vertices, expected communication cost is O(k(|V| + |E|)) in iteration k

- Number of iterations: ???
  - On a path graph: 4 log n
  - In a general graph: 4 log d (conjecture)

Leader Algorithm

- Let π be an arbitrary total order over the vertices.
- Begin with $l(v) = v$, and all nodes active

In each iteration:

- Let $C(v)$ be the connected component containing v
- Let $\Gamma(v)$ be the neighbors of $C(v)$ that are not in $C(v)$
- Call each active node a leader with probability ½.
- For each active non-leader w, find $w^* = \min(\Gamma(w))$
- If w^* is not empty and $l(w^*)$ is a leader,
  then mark w as passive, and
  relabel each node with label w by $l(w^*)$

Correctness

- If at any point of time two nodes s and t have the same label, then they are connected in G.

- Consider an iteration, when l(s) ≠ l(t) before the iteration, but l(s) = l(t) after.
- This means, l(s) = w (non-leader node), l(t) = w*
- By induction, s is connected to all nodes in Γ(w),
  t is connected to all nodes in Γ(w*), and
  w is connected to w*.
- Therefore, s and t are connected.

Duke UNIVERSITY

Number of Iterations

- Every connected component has a unique label after O(log N) rounds with high probability

- Suppose there is some connected component with two active labels.

- An active label w survives an iteration if:
  1. w is marked a leader
  2. w is not marked a leader and l(w*) is not marked a leader

- Hence, in every iteration, the expected number of active labels reduces by ¼.

Summary

- No native support for iteration in Map-Reduce
  – Each iteration writes/reads data from disk leading to overheads

- Many graph algorithms need iterative computation
  – Need to design algorithms that can minimize number of iterations

Hash-Greater-to-Min

- Each vertex v maintains:
  v_{min} : minimum node
  C(v) : cluster

- Run Hcc 2 times …
  Map: send v_{min} to neighbors
  Reduce: Compute new v_{min} and add it to C(v)

- Run Greater to min step once …
  Map: Let $C_{>=v}$ be all nodes in C(v) that have id >= v
  Send $v_m$in to all nodes in $C_{>=v}$
  Send $C_{>=v}$ to v_{min}
  Reduce: Union the incoming clusters.

Hash-Greater-to-Min

- Theorem: The algorithm completes in expectation 3 log n steps (over random node orderings), where n is the size of the largest component.

- Lemma: Let GT(v) be the set of nodes where v is the minimum node (after a greater-to-min step). Then GT(v) = set of nodes in C(v) that have ids >= v.

Duke UNIVERSITY

Proof of Theorem

- After 3K rounds, let Mk be the nodes that appear as minimum on some nodes.
- GTk(m) = set of nodes where m is the minimum
- GTk(m) is disjoint from GTk(m') for all m and m'.
- Construct a graph G_{Mk}, with vertices from Mk, and (m,m') is an edge if there exist v in GTk(m) and v' in GTk(m') such that (v,v') is an edge in the original graph.

Duke UNIVERSITY

Proof of Theorem

- Consider a connected component in GMk (and let |Mk| = s)

- If m < m’ are connected in GMk, then m’ will no longer be a minimum node after 3 rounds:
  - There exist v in GTk(m) and v’ in GTk(m’) that are neighbors in G
  - In one step of Hcc, v send m to v’
  - In second step of Hcc, v’ sends m’ to m