CSE544
Data Management

Lectures 15
Parallel Query Processing

Announcements

- Poster presentations:
  - Friday: 10am – ?? In the atrium
  - No access to the CS printer? → Walter!
  - Please bring a laptop to give a demo

- Review of the Snowflake paper was due today

- Homework 5 will be posted on Wednesday

Outline

- MapReduce
- Snowflake
- Optimal parallel algorithm

References

- Jeffrey Dean and Sanjay Ghemawat, MapReduce: Simplified Data Processing on Large Clusters. OSDI'04
- D. DeWitt and M. Stonebraker. Mapreduce – a major step backward. In Database Column (Blog), 2008.

Distributed File System (DFS)

- For very large files: TBs, PBs

- Each file partitioned into *chunks* (64MB)

- Each chunk replicated (≥3 times) – why?

- Implementations:
  - Google's DFS: GFS, proprietary
  - Hadoop's DFS: HDFS, open source

MapReduce

- Google:
  - Started around 2000
  - Paper published 2004
  - Discontinued September 2019
- Free variant: Hadoop

- MapReduce = high-level programming model and implementation for large-scale parallel data processing

Data Model

Files!

A file = a bag of **(key, value)** pairs

A MapReduce program:

- Input: a bag of **(inputkey, value)** pairs
- Output: a bag of **(outputkey, value)** pairs

Step 1: the MAP Phase

User provides the MAP-function:

- Input: **(input key, value)**
- Ouput: bag of **(intermediate key, value)**

System applies the map function in parallel to all **(input key, value)** pairs in input file

Step 2: the REDUCE Phase

User provides the REDUCE function:

- Input: **(intermediate key, bag of values)**
- Output: bag of output **(values)**

System groups all pairs with the same intermediate key, and passes the bag of values to the REDUCE function

Example

- Counting the number of occurrences of each word in a large collection of documents
- Each Document
  - The key = document id (did)
  - The value = set of words (word)

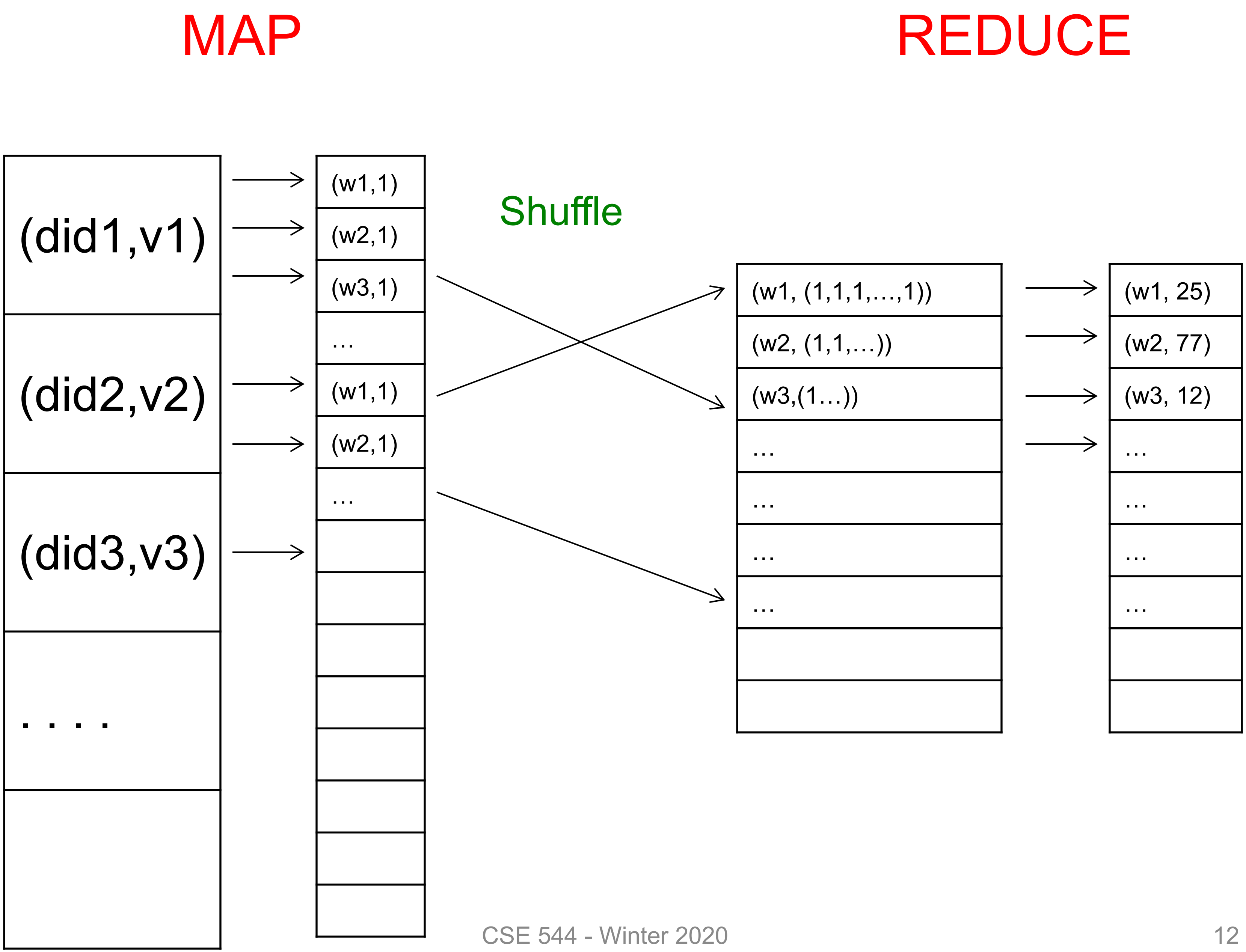
```
map(String key, String value):
   // key: document name
   // value: document contents
   for each word w in value:
     EmitIntermediate(w, "1");
```

```
reduce(String key, Iterator values):
   // key: a word
   // values: a list of counts
   int result = 0;
   for each v in values:
     result += ParseInt(v);
   Emit(AsString(result));
```

MapReduce = GroupBy-Aggregate

Occurrence(docID, word)

```
select   word, count(*)
from     Occurrence
group by word
```


MAP
REDUCE
(did1,v1)
(did2,v2)
(did3,v3)
. . . .
(w1,1)
(w2,1)
(w3,1)
…
(w1,1)
(w2,1)
…
Shuffle
(w1, (1,1,1,…,1))
(w2, (1,1,…))
(w3,(1…))
…
…
…
…
(w1, 25)
(w2, 77)
(w3, 12)
…
…
…
…

Jobs v.s. Tasks

- A MapReduce Job
  - One simple "query", e.g. count words in docs
  - Complex queries may require many jobs
- A Map Task, or a Reduce Task
  - A group of instantiations of the map-, or reduce-function, to be scheduled on a single worker

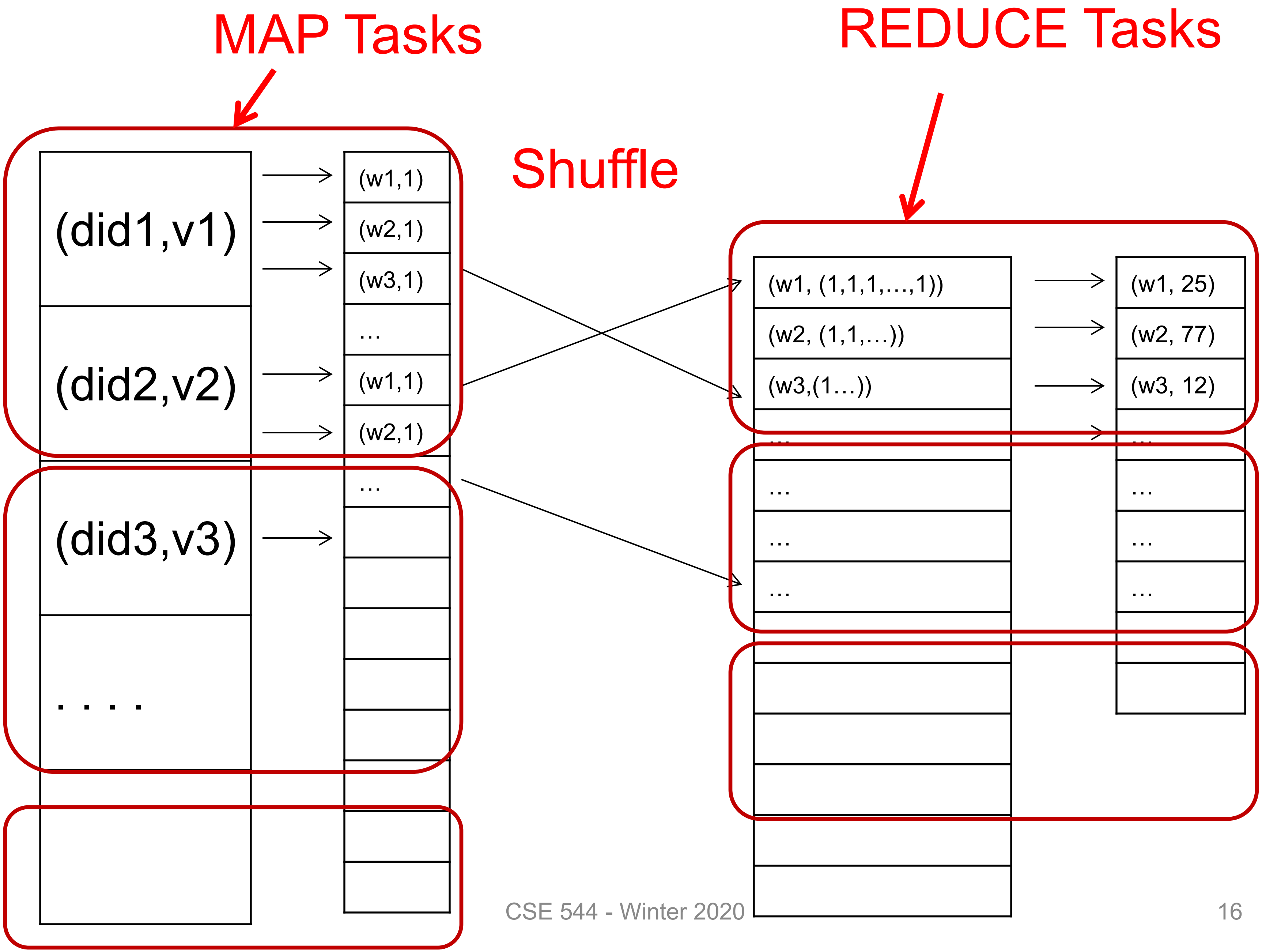
Workers

- A worker is a process that executes one task at a time
- Typically there is one worker per processor, hence 4 or 8 per node

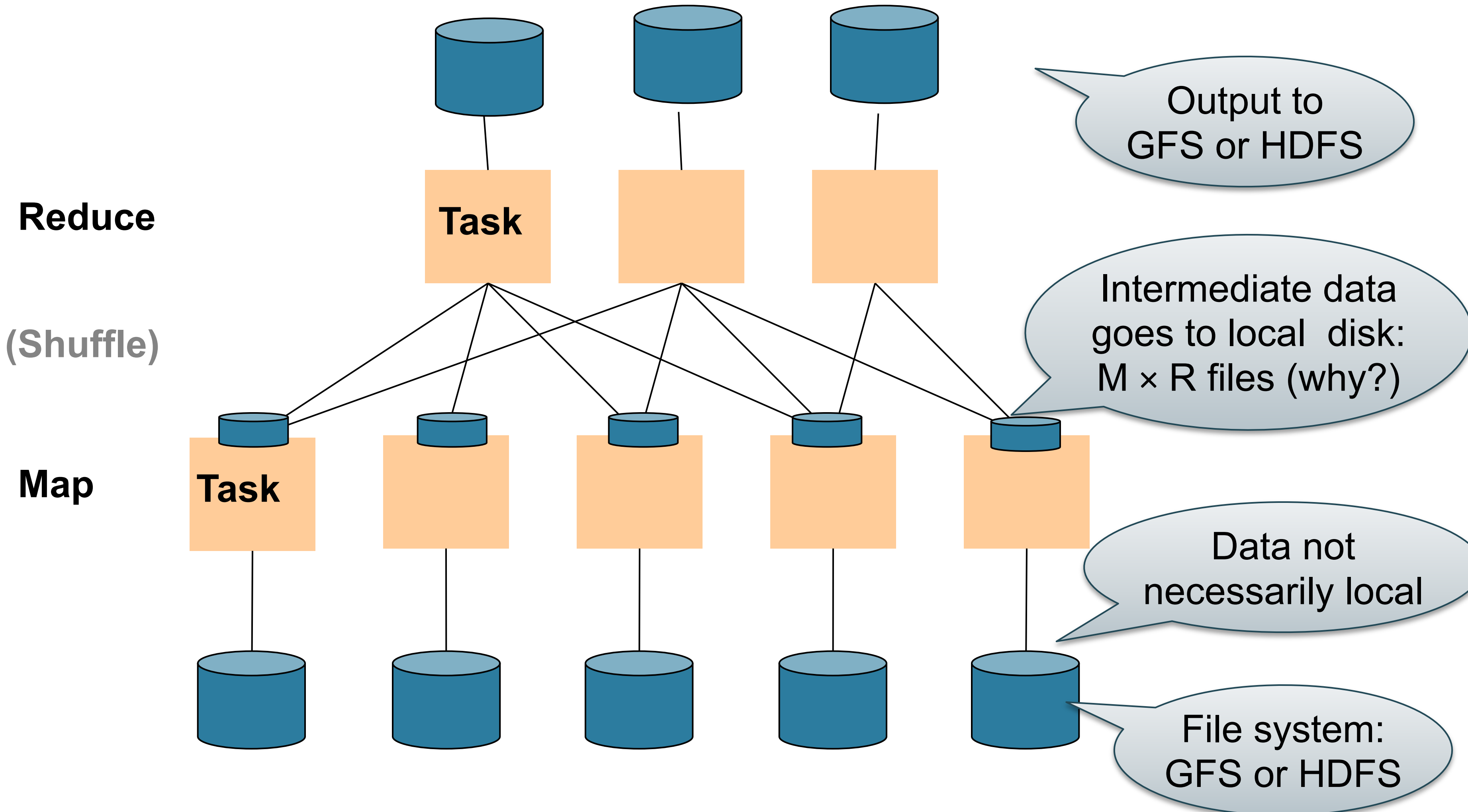
Fault Tolerance

- If one server fails once every year…
  ... then a job with 10,000 servers will fail in less than one hour

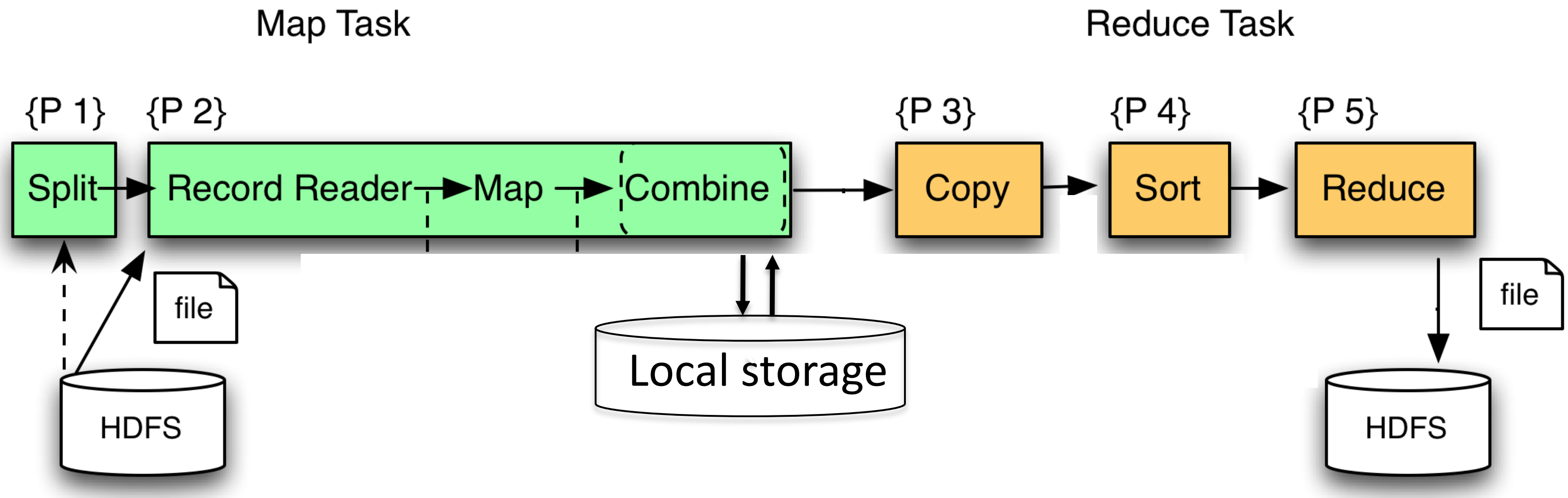
- MapReduce handles fault tolerance by writing intermediate files to disk:
  – Mappers write file to local disk
  – Reducers read the files (=reshuffling); if the server fails, the reduce task is restarted on another server


MAP Tasks
REDUCE Tasks
Shuffle
(did1,v1)
(did2,v2)
(did3,v3)
. . . .
(w1,1)
(w2,1)
(w3,1)
…
(w1,1)
(w2,1)
…
(w1, (1,1,1,…,1))
(w2, (1,1,…))
(w3,(1…))
…
…
…
(w1, 25)
(w2, 77)
(w3, 12)
…
…
…

MapReduce Execution Details



MapReduce Phases



Implementation

- There is one master node
- Master partitions input file into *M splits*, by key
- Master assigns *workers* (=servers) to the *M map tasks*, keeps track of their progress
- Workers write their output to local disk, partition into *R regions*
- Master assigns workers to the *R reduce tasks*
- Reduce workers read regions from the map workers' local disks

Interesting Implementation Details

Worker failure:

- Master pings workers periodically,
- If down then reassigns the task to another worker

Interesting Implementation Details

Backup tasks:

- *Straggler* = a machine that takes unusually long time to complete one of the last tasks.
  - Bad disk forces frequent correctable errors (30MB/s → 1MB/s)
  - The cluster scheduler has scheduled other tasks on that machine
- Stragglers are a main reason for slowdown
- Solution*: pre-emptive backup execution of the last few remaining in-progress tasks*

MapReduce v.s. Databases

Blog by DeWitt and Stonebraker

- “Schemas are good”
- “Indexes”
- “Skew” (MR mitigates it somewhat, how?)
- The M * R problem – what is it?
- “Parallel databases uses push (to sockets) instead of pull” – what’s the point?

Snowflake – Discussion

- ”The Snowflake Elastic Data Warehouse”, Dageville et al., SIGMOD’2016

Snowflake

- It is an SaaS – what is this? Give other examples of types of cloud services…

Snowflake

- It is an SaaS – what is this? Give other examples of types of cloud services…
- SaaS = software as a service
- Other examples:
  - Platform as a service (PaaS): e.g. Amazon's EC
  - Infrastructure as a service (virtual machines)
  - Software as a Service
  - Function as a Service: Amazon's Lambda

Snowflake

- Describe Snowflake’s Data Storage

Snowflake

- Describe Snowflake's Data Storage

In class:

- S3:PUT/GET/DELETE
- Table → horizontal partition in *files*
- Blobs+PAX
- Temp storage→S3

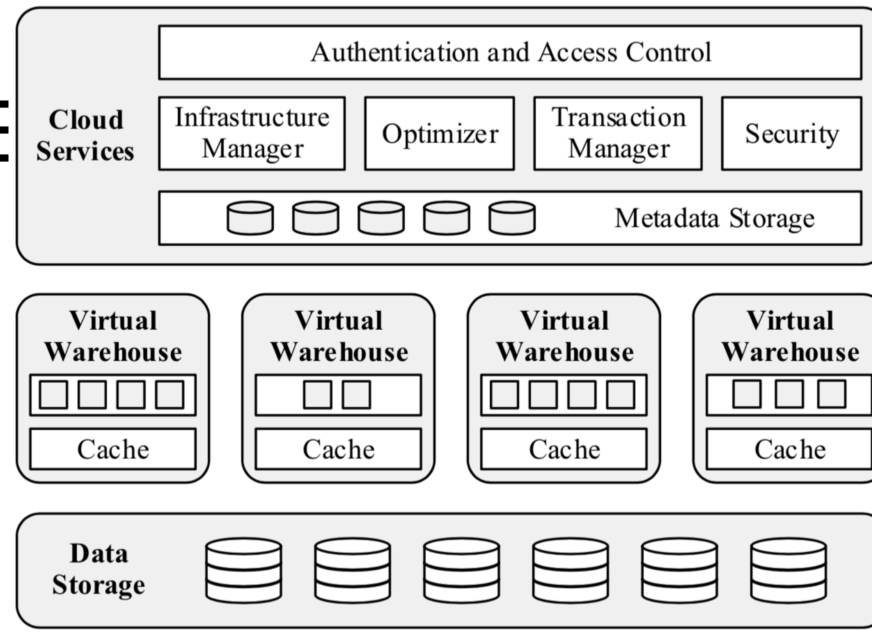


Figure 1: Multi-Cluster, Shared Data Architecture

Snowflake

- Describe Elasticity in Snowflake

- Describe failure handling in Snowflake

Snowflake

- Describe Elasticity in Snowflake
  - Virtual Warehouse (VW) serves one user
  - T-Shirt sizes: X-Small … XX-Large
  - Small query may run on subset of VW
- Describe failure handling in Snowflake

Snowflake

- Describe Elasticity in Snowflake
  – Virtual Warehouse (VW) serves one user
  – T-Shirt sizes: X-Small … XX-Large
  – Small query may run on subset of VW
- Describe failure handling in Snowflake
  – Restart the query
  – No partial retries (like MapReduce or Spark)

Snowflake

- Describe its execution engine

Snowflake

- Describe its execution engine
- Column-oriented (in class)
- Vectorized (“tuple batches” – in class)
- Push-based (in class)

Snowflake

- What does Snowflake use instead of indexes?

Snowflake

- What does Snowflake use instead of indexes?

- “Pruning”: for each file (recall: this is a horizontal partition of a table) and each attribute, it stores the min/max values in that column in that file; may skip files when not needed.

Parallel Processing of Complex Queries

Communication v.s. Rounds

- Multi-join Query: R ⋈ S ⋈ T ⋈ K
- Solution 1: use multiple rounds:
  - Round 1: R ⋈ S
  - Round 2: (R ⋈ S) ⋈ T
  - Round 3: ((R ⋈ S) ⋈ T) ⋈ K
  - …
- Solution 2: use a single round, with more communication

Outline

- Basics
- Unequal Inputs
- Skew
- Multiple rounds

$|R| = |S| = |T| = m$ tuples

The Triangles Query

$Q(x,y,z) = R(x,y) \wedge S(y,z) \wedge T(z,x)$

Round 1: $Temp(x,y,z) = R(x,y) \wedge S(y,z)$

Round 2: $Q(x,y,z) = Temp(x,y,z) \wedge T(z,x)$

Problem: $|Temp| \gg m$

$|R| = |S| = |T| = m$ tuples

The Triangles Query

$Q(x,y,z) = R(x,y) \wedge S(y,z) \wedge T(z,x)$

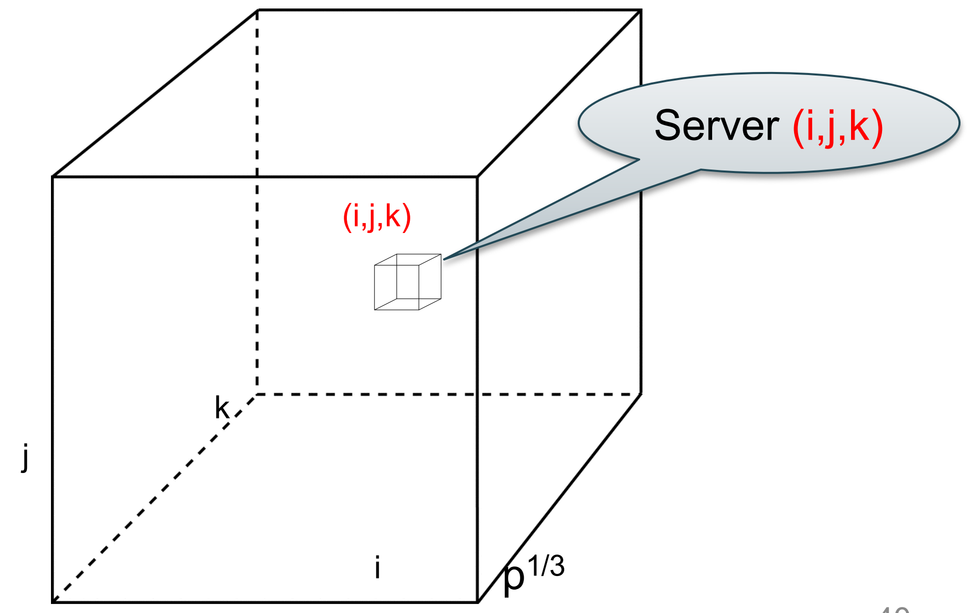
Algorithm in one round!

- [Afrati'10] Shares Algo (MapReduce)
- [Beame'13,'14] HyperCube Algo (MPC)

Triangles in One Round

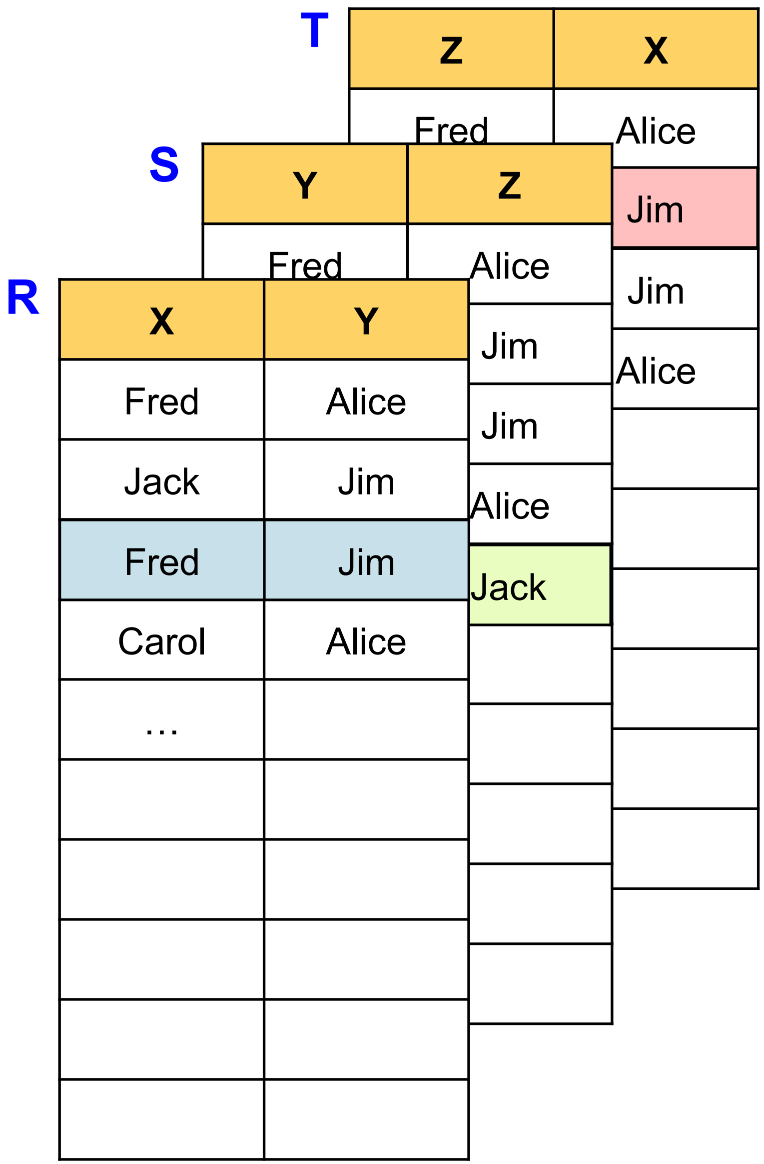
- Place servers in a cube $p = p^{1/3} \times p^{1/3} \times p^{1/3}$
- Each server identified by (i,j,k)

- Choose 3 random, independent hash functions:
  h_1 : Dom → $[p^{1/3}]$
  h_2 : Dom → $[p^{1/3}]$
  h_3 : Dom → $[p^{1/3}]$



$Q(x,y,z) = R(x,y) \wedge S(y,z) \wedge T(z,x)$ $|R| = |S| = |T| = m$ tuples

Triangles in One Round



Round 1:

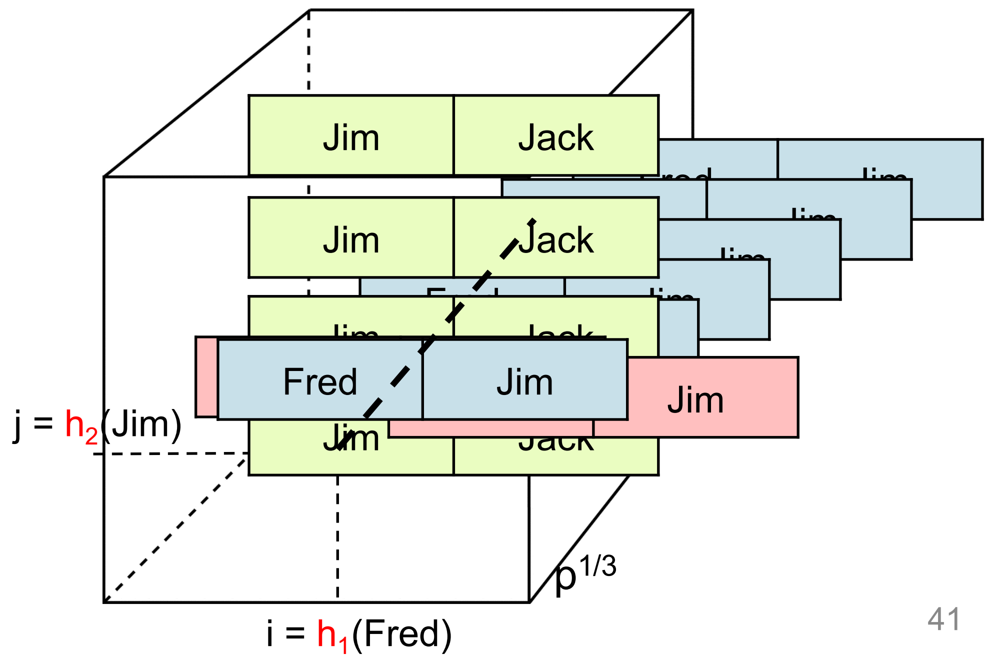
Send $R(x,y)$ to all servers $(h_1(x), h_2(y), *)$

Send $S(y,z)$ to all servers $(*, h_2(y), h_3(z))$

Send $T(z,x)$ to all servers $(h_1(x), *, h_3(z))$

Output:

compute locally $R(x,y) \wedge S(y,z) \wedge T(z,x)$



Communication Cost

Theorem HyperCube has load $L = O(m/p^{2/3})$ w.h.p., on any input database without skew.

Skew threshold: $m/p^{1/3}$ or lower

This load is optimal, even for data without skew

Recap

- So far we discussed:
  - Join $L = m/p$
  - Triangles $L = m/p^{2/3}$

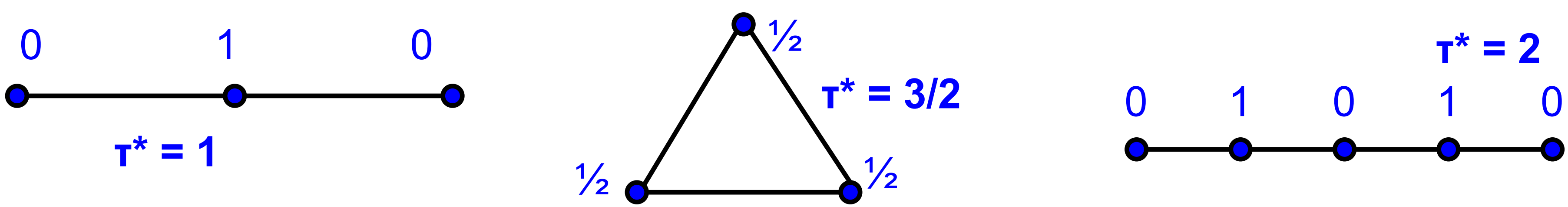
- How do we compute a full CQ?

$$\mathbf{Q}(\mathbf{x_1}, \ldots, \mathbf{x_k}) = \mathbf{S_1}(\bar{\mathbf{x}}_1) \wedge \mathbf{S_2}(\bar{\mathbf{x}}_2) \wedge \cdots \wedge \mathbf{S}_\ell(\bar{\mathbf{x}}_\ell)$$

- Hypercube: $p = p_1 * p_2 * \ldots * p_k$
- Optimize shares $p_1, p_2, \ldots, p_k$ to minimize L

Review

Definition. A *fractional vertex cover* of a hypergraph are weights $v_1, \ldots, v_k$ s.t. for each hyperedge, the sum of its weights is ≥ 1

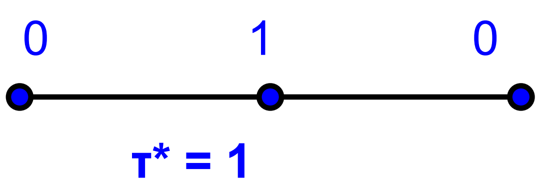


$\mathbf{Q}(\mathbf{x_1}, \ldots, \mathbf{x_k}) = \mathbf{S_1}(\bar{\mathbf{x}}_1) \wedge \mathbf{S_2}(\bar{\mathbf{x}}_2) \wedge \cdots \wedge \mathbf{S}_\ell(\bar{\mathbf{x}}_\ell)$ $|S_1| = |S_2| = \ldots = m$

Optimal Shares

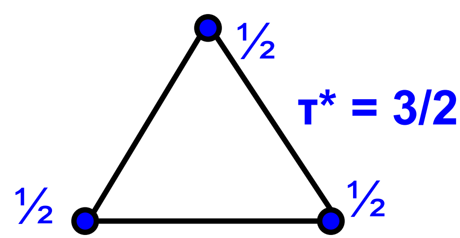
Theorem. The optimal shares are $p_i = p^{v_i^* / \tau^*}$.
The optimal load is $L = O(m/p^{1/\tau^*})$ on databases without skew.

R(x,y)∧S(y,z)



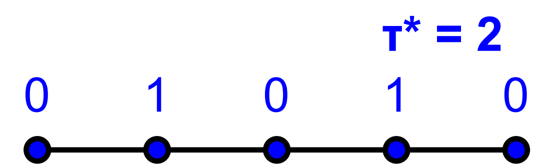
$L = m / p$

R(x,y)∧S(y,z)∧T(z,x)

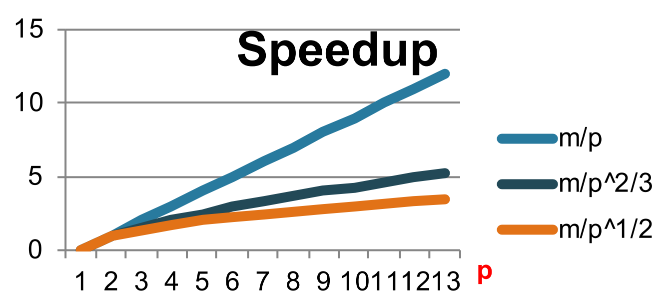


$L = m / p^{2/3}$

R(x,y)∧S(y,z)∧T(z,u)∧K(u,v)



$L = m / p^{1/2}$



Discussion

- Hypercube algorithm: communication only
  - We do not discuss the local computation
- Optimal algorithm = optimal vertex cover
- Load $m/p^{1/\tau^*}$ depends on input, not output!

Many restrictions…

Next: remove restrictions

Unequal Inputs

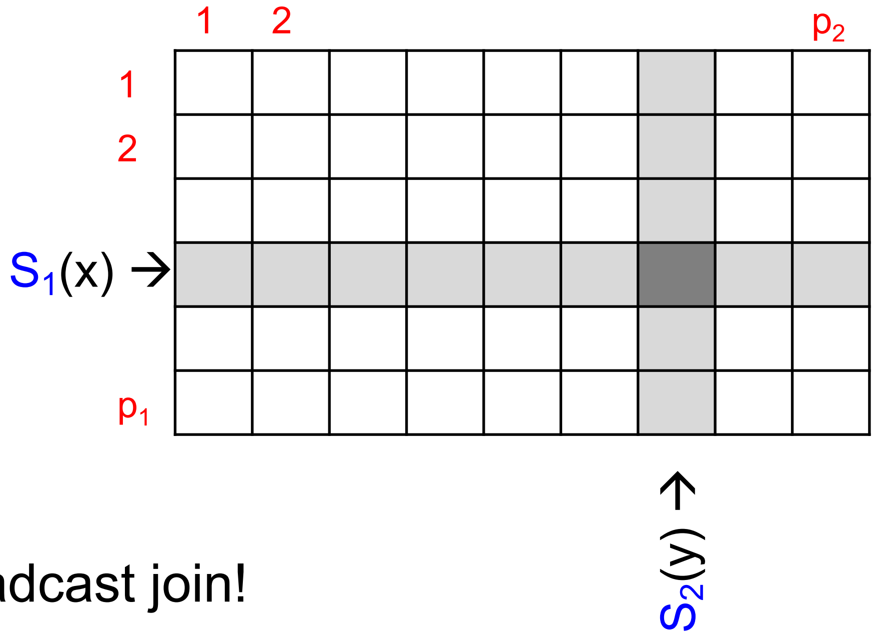
Motivation

- Cardinalities m_1, m_2, … are the simplest kind of data statistics
- State of the art: even the simplest optimizers today use cardinalities
- E.g.: $R \bowtie S$: if $R >> S$, then broadcast S

- What is the optimal load $L = f(m_1, m_2, \ldots)$?

Warm up: Cartesian Product

$Q(x,y) = S_1(x) \wedge S_2(y)$, $|S_1| = m_1$ $|S_2| = m_2$



Fact Optimal load

$$L_{opt} = 2\left(\frac{m_1 m_2}{p}\right)^{1/2}$$

Proof optimal when

$m_1 / p_1 = m_2 / p_2 = (m_1 m_2 / p)^{1/2}$

If $m_1 \ll m_2$ then it becomes broadcast join!

$Q(x_1,\ldots,x_c) = S_1(x_1) \wedge \cdots \wedge S_c(x_c)$

Fact. Optimal load

$$L_{opt} = c\left(\frac{m_1 \cdots m_c}{p}\right)^{1/c}$$

$\mathbf{Q}(\mathbf{x_1}, \ldots, \mathbf{x_k}) = \mathbf{S_1}(\bar{\mathbf{x}}_\mathbf{1}) \wedge \mathbf{S_2}(\bar{\mathbf{x}}_\mathbf{2}) \wedge \cdots \wedge \mathbf{S}_\ell(\bar{\mathbf{x}}_\ell)$ Relations sizes= m_1, m_2, …

A Simple Lower Bound on L

Definition. A *edge packing* is a set of atoms $\mathbf{S_{j_1}}, \cdots \mathbf{S_{j_c}}$ that do not share any variables.

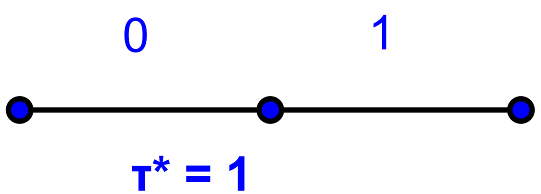
Fact. For any packing, $\mathbf{S_{j_1}}, \cdots \mathbf{S_{j_c}}$ any 1-round algorithm computing Q has load $\mathbf{L} \geq \mathbf{c} \left(\frac{\mathbf{m_{j_1}} \cdots \mathbf{m_{j_c}}}{\mathbf{p}} \right)^{\mathbf{1/c}}$

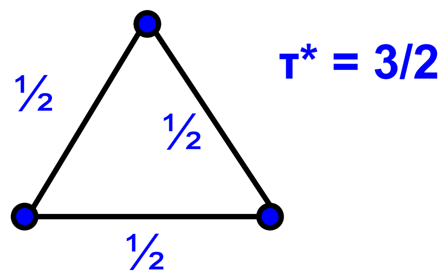
Proof To compute Q, the algorithm must also compute $\mathbf{S_{j_1}} \times \cdots \times \mathbf{S_{j_c}}$

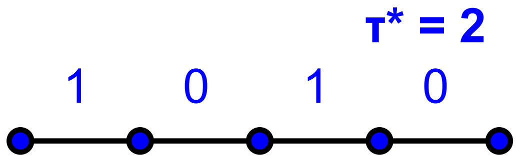
$\mathbf{Q}(\mathbf{x_1}, \ldots, \mathbf{x_k}) = \mathbf{S_1}(\bar{\mathbf{x}}_1) \wedge \mathbf{S_2}(\bar{\mathbf{x}}_2) \wedge \cdots \wedge \mathbf{S}_\ell(\bar{\mathbf{x}}_\ell)$ Relations sizes= m_1, m_2, …

Background

Definition. A *fractional edge packing* of a hypergraph are weights u_1 …, u_l s.t. for each node, the sum of its weights is ≤ 1







$\mathbf{Q}(\mathbf{x_1}, \ldots, \mathbf{x_k}) = \mathbf{S_1}(\bar{\mathbf{x}}_1) \wedge \mathbf{S_2}(\bar{\mathbf{x}}_2) \wedge \cdots \wedge \mathbf{S}_\ell(\bar{\mathbf{x}}_\ell)$ Relations sizes= m_1, m_2, …

Optimal Load L

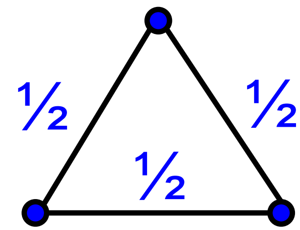
Define $\mathbf{L}(\mathbf{u}) \stackrel{\text{def}}{=} \left(\frac{\mathbf{m_1}^{\mathbf{u_1}} \cdot \mathbf{m_2}^{\mathbf{u_2}} \cdots \mathbf{m}_\ell^{\mathbf{u}_\ell}}{\mathbf{p}} \right)^{\frac{1}{\mathbf{u_1} + \mathbf{u_2} + \cdots + \mathbf{u}_\ell}}$

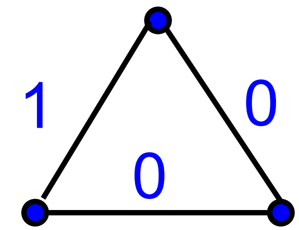
For any fractional edge packing **u**

Theorem Optimal 1-round load is L = $\max_{\mathbf{u}}$ L(**u**)

Example

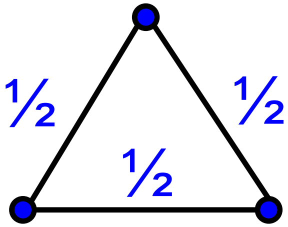
$Q(x,y,z) = R(x,y) \wedge S(y,z) \wedge T(z,x)$

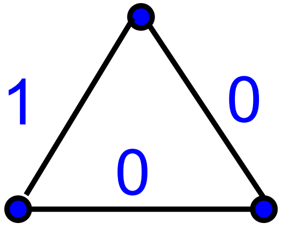




Example

$Q(x,y,z) = R(x,y) \wedge S(y,z) \wedge T(z,x)$

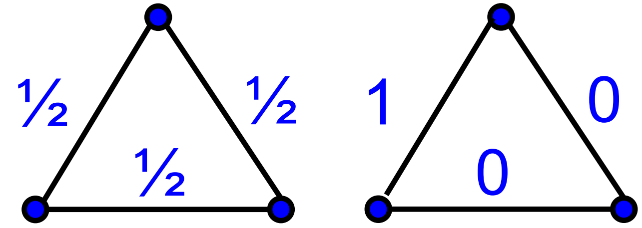




$$\mathbf{L}(\mathbf{u}) =$$

$$\left(\frac{\mathbf{m_R}^{\mathbf{u_R}} \cdot \mathbf{m_S}^{\mathbf{u_S}} \cdot \mathbf{m_T}^{\mathbf{u_T}}}{\mathbf{p}}\right)^{\frac{1}{\mathbf{u_R}+\mathbf{u_S}+\mathbf{u_T}}}$$

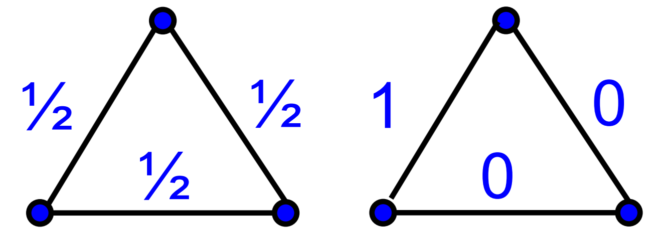
Example



$Q(x,y,z) = R(x,y) \wedge S(y,z) \wedge T(z,x)$

Vertex of edge packing polytope u_R, u_S, u_T	$L(u) = \left(\frac{m_R^{u_R} \cdot m_S^{u_S} \cdot m_T^{u_T}}{p}\right)^{\frac{1}{u_R+u_S+u_T}}$
1/2, 1/2, 1/2	

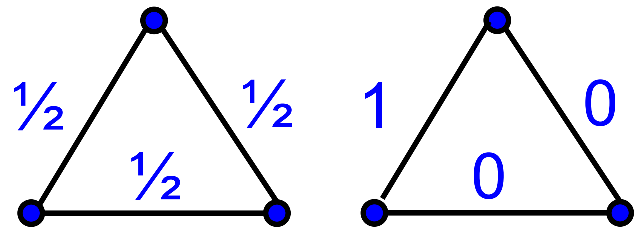
Example



$Q(x,y,z) = R(x,y) \wedge S(y,z) \wedge T(z,x)$

Vertex of edge packing polytope u_R, u_S, u_T	$L(u) = \left(\frac{m_R^{u_R} \cdot m_S^{u_S} \cdot m_T^{u_T}}{p}\right)^{\frac{1}{u_R+u_S+u_T}}$
1/2, 1/2, 1/2	
1, 0, 0	
0, 1, 0	
0, 0, 1	
0, 0, 0	

Example

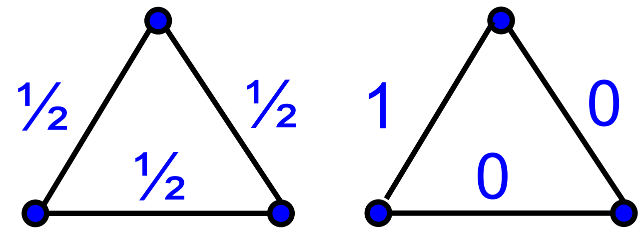


$Q(x,y,z) = R(x,y) \wedge S(y,z) \wedge T(z,x)$

Vertex of edge packing polytope u_R, u_S, u_T	$L(u) = \left(\frac{m_R^{u_R} \cdot m_S^{u_S} \cdot m_T^{u_T}}{p}\right)^{\frac{1}{u_R+u_S+u_T}}$
1/2, 1/2, 1/2	$(m_R\, m_S\, m_T)^{1/3} / p^{2/3}$
1, 0, 0	m_R / p
0, 1, 0	m_S / p
0, 0, 1	m_T / p
0, 0, 0	0

L = max of these five values

Example

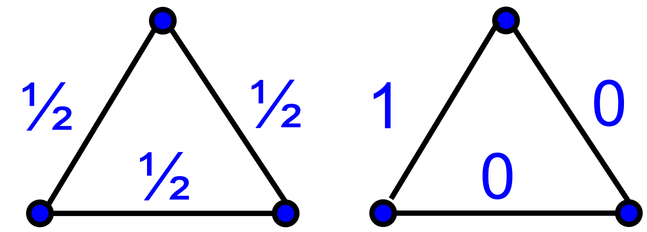


Q(x,y,z) = R(x,y) ∧ S(y,z) ∧ T(z,x)

Vertex of edge packing polytope u_R, u_S, u_T	$\mathbf{L}(\mathbf{u}) = \left(\frac{\mathbf{m_R}^{\mathbf{u_R}} \cdot \mathbf{m_S}^{\mathbf{u_S}} \cdot \mathbf{m_T}^{\mathbf{u_T}}}{\mathbf{p}}\right)^{\frac{1}{\mathbf{u_R}+\mathbf{u_S}+\mathbf{u_T}}}$	Max when …	HC Algorithm $\mathbf{p}^{\mathbf{e_x}} \cdot \mathbf{p}^{\mathbf{e_y}} \cdot \mathbf{p}^{\mathbf{e_z}}$
1/2, 1/2, 1/2	$(m_R\ m_S\ m_T)^{1/3} / p^{2/3}$		
1, 0, 0	m_R / p		
0, 1, 0	m_S / p		
0, 0, 1	m_T / p		
0, 0, 0	0		

L = max of these five values

Example

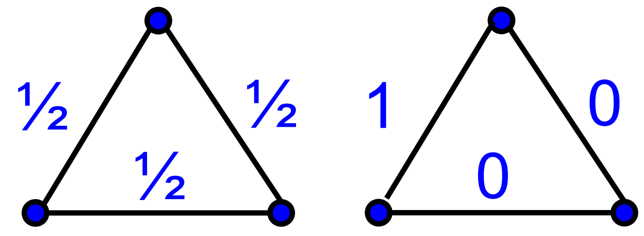


$Q(x,y,z) = R(x,y) \wedge S(y,z) \wedge T(z,x)$

Vertex of edge packing polytope u_R, u_S, u_T	$L(\mathbf{u}) = \left(\frac{m_R^{u_R} \cdot m_S^{u_S} \cdot m_T^{u_T}}{p}\right)^{\frac{1}{u_R+u_S+u_T}}$	Max when ...	HC Algorithm $p^{e_x} \cdot p^{e_y} \cdot p^{e_z}$
1/2, 1/2, 1/2	$(m_R\ m_S\ m_T)^{1/3} / p^{2/3}$		
1, 0, 0	m_R / p		
0, 1, 0	m_S / p		
0, 0, 1	m_T / p		
0, 0, 0	0	never	

L = max of these five values

Example



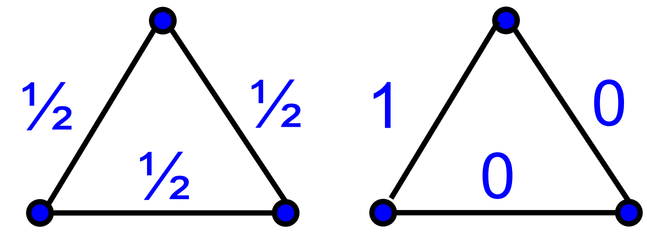
$Q(x,y,z) = R(x,y) \wedge S(y,z) \wedge T(z,x)$

Vertex of edge packing polytope u_R, u_S, u_T	$\mathbf{L}(\mathbf{u}) = \left(\frac{\mathbf{m_R}^{\mathbf{u_R}} \cdot \mathbf{m_S}^{\mathbf{u_S}} \cdot \mathbf{m_T}^{\mathbf{u_T}}}{\mathbf{p}}\right)^{\frac{1}{\mathbf{u_R}+\mathbf{u_S}+\mathbf{u_T}}}$	Max when ...	HC Algorithm $\mathbf{p}^{\mathbf{e_x}} \cdot \mathbf{p}^{\mathbf{e_y}} \cdot \mathbf{p}^{\mathbf{e_z}}$
1/2, 1/2, 1/2	$(m_R\ m_S\ m_T)^{1/3} / p^{2/3}$	$m_R \approx m_S \approx m_T$	$e_x, e_y, e_z > 0$
1, 0, 0	m_R / p		
0, 1, 0	m_S / p		
0, 0, 1	m_T / p		
0, 0, 0	0	never	

L = max of these five values

Example

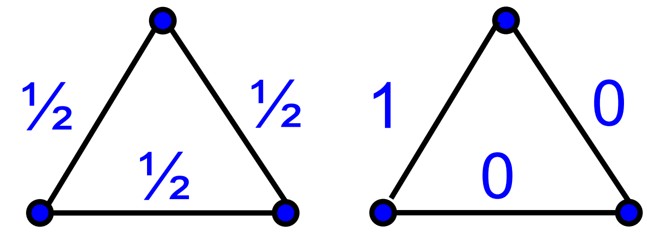
$Q(x,y,z) = R(x,y) \wedge S(y,z) \wedge T(z,x)$



Vertex of edge packing polytope u_R, u_S, u_T	$L(\mathbf{u}) = \left(\frac{m_R^{u_R} \cdot m_S^{u_S} \cdot m_T^{u_T}}{p}\right)^{\frac{1}{u_R+u_S+u_T}}$	Max when ...	HC Algorithm $p^{e_x} \cdot p^{e_y} \cdot p^{e_z}$
1/2, 1/2, 1/2	$(m_R\, m_S\, m_T)^{1/3} / p^{2/3}$	$m_R \approx m_S \approx m_T$	$e_x, e_y, e_z > 0$
1, 0, 0	m_R / p	$\frac{m_R}{p} \geq \sqrt{\frac{m_S m_T}{p}}$	$e_z = 0$ join R with product S×T
0, 1, 0	m_S / p	. . .	. . .
0, 0, 1	m_T / p	. . .	. . .
0, 0, 0	0	never	

L = max of these five values

Example



$Q(x,y,z) = R(x,y) \wedge S(y,z) \wedge T(z,x)$

Vertex of edge packing polytope u_R, u_S, u_T	$L(\mathbf{u}) = \left(\frac{m_R^{u_R} \cdot m_S^{u_S} \cdot m_T^{u_T}}{p}\right)^{\frac{1}{u_R+u_S+u_T}}$	Max when ...	HC Algorithm $p^{e_x} \cdot p^{e_y} \cdot p^{e_z}$
1/2, 1/2, 1/2	$(m_R\ m_S\ m_T)^{1/3} / p^{2/3}$	$m_R \approx m_S \approx m_T$	$e_x, e_y, e_z > 0$
1, 0, 0	m_R / p	$\frac{m_R}{p} \geq \sqrt{\frac{m_S m_T}{p}}$	$e_z = 0$ join R with product S×T
0, 1, 0	m_S / p	. . .	. . .
0, 0, 1	m_T / p	. . .	. . .
0, 0, 0	0	never	

Better speedup

L = max of these five values

Discussion (1/2)

- Closed formula for *optimal load* L

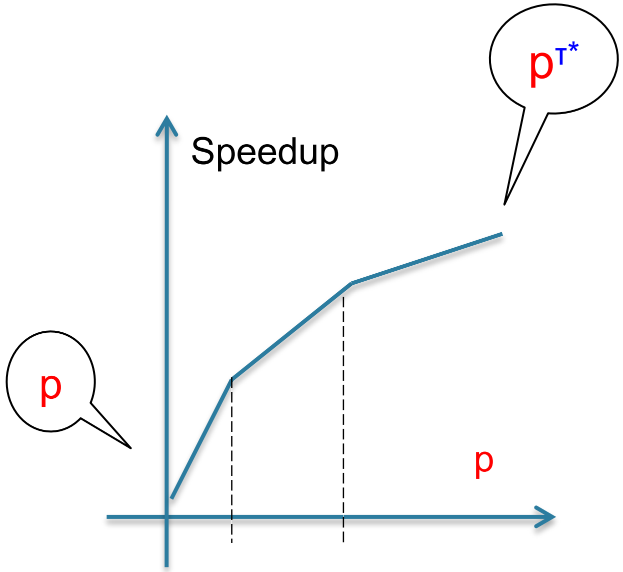
- No closed formula for shares $p_1, \ldots, p_k$, but can compute numerically

- Optimal Plan: broadcast smaller relations, hash-partition larger relations

Discussion (2/2)

Optimal Plan Depends on p

- When p is small:
  - Broadcast the "small" relation(s)
  - Linear speedup ~ p
- When p is large
  - All relations look "big"
  - Sub-linear speedup ~ p^{τ^*}



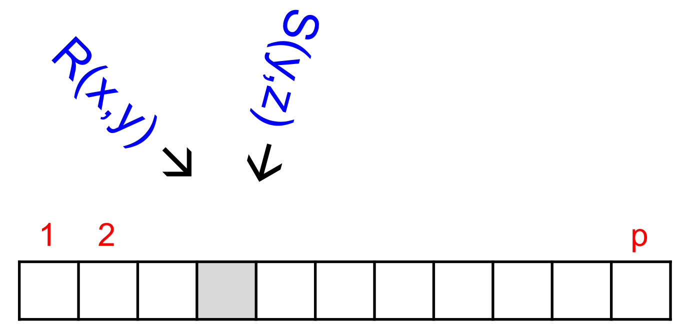
Skew

Skew Matters

- Skewed data significantly degrades the performance in distributed query processing; skewed values must be treated specially
- State of the art in large scale distributed system: DIY ☹

Skewed Values → Residual Query

$Q(x,y,z) = R(x,y) \wedge S(y,z)$



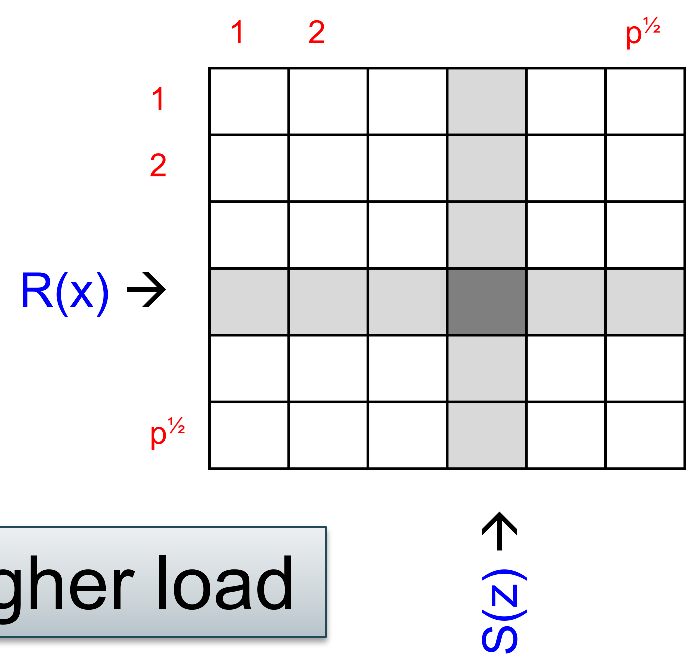
No-skew:

$\tau^* = 1, \quad L = m/p$

Skewed: $Q(x,z) = R(x) \wedge S(z)$

$\tau^* = 2, \quad L = m/p^{1/2}$

Residual query



Skew necessarily leads to higher load

Residual-Query Algorithm

Def. A value is a *heavy hitter* if it occurs > m/p times

Def. Fix **x** ⊆{x_1, …, x_k}. The *residual query* $Q_{\mathbf{x}}$ is obtained from Q by removing the variables **x** and the empty atoms.

Algorithm: In parallel, for every combination of heavy/light, compute the residual query for that combination

Theorem. The algorithm is optimal for 1 round.

Example

$Q(x,y,z) = R(x,y) \wedge S(y,z) \wedge T(z,x)$

Heavy hitter = a value that occurs at least m/p times
Each attribute has at most p heavy hitters

x	y	z	Residual query	τ^*	L	$p_1 \times p_2 \times p_3$
light	light	light	$R(x,y) \wedge S(y,z) \wedge T(z,x)$	3/2	$m/p^{2/3}$	$p^{1/3} \times p^{1/3} \times p^{1/3}$
. . . .			. . .			

Example

$Q(x,y,z) = R(x,y) \wedge S(y,z) \wedge T(z,x)$

Heavy hitter = a value that occurs at least m/p times

Each attribute has at most p heavy hitters

x	y	z	Residual query	τ^*	L	$p_1 \times p_2 \times p_3$
light	light	light	$R(x,y) \wedge S(y,z) \wedge T(z,x)$	3/2	$m/p^{2/3}$	$p^{1/3} \times p^{1/3} \times p^{1/3}$
light	light	heavy	$R(x,y) \wedge S(y) \wedge T(x)$	2	$m/p^{1/2}$	$p^{1/2} \times p^{1/2} \times 1$
. . . .			. . .			

Example

$Q(x,y,z) = R(x,y) \wedge S(y,z) \wedge T(z,x)$

Heavy hitter = a value that occurs at least m/p times

Each attribute has at most p heavy hitters

x	y	z	Residual query	τ^*	L	$p_1 \times p_2 \times p_3$
light	light	light	$R(x,y) \wedge S(y,z) \wedge T(z,x)$	3/2	$m/p^{2/3}$	$p^{1/3} \times p^{1/3} \times p^{1/3}$
light	light	heavy	$R(x,y) \wedge S(y) \wedge T(x)$	2	$m/p^{1/2}$	$p^{1/2} \times p^{1/2} \times 1$
light	heavy	heavy	$R(x) \wedge T(x)$	1	m/p	$p \times 1 \times 1$
. . . .			. . .			

Broadcast $S(y,z)$
OK because $|S| \leq p^2$

Discussion

- General principle for skew: ignore heavy hitter, compute residual query
- When data is skewed, load necessarily increases

Next: use ***multiple rounds*** to avoid increase

Multiple Rounds

Multiple Rounds

State of the art:

- Each operator level in the query plan is a separate round

Theoretical results are limited

- No skew: difficult theoretical analysis
- Skewed data: optimality for some queries

A Lower Bound

ρ^* = optimal edge covering number of Q

AGM bound

Theorem Suppose each S_j has size ≤ m. Then $|Q(DB)| \leq m^{\rho^*}$.

Corollary Any r rounds algorithm has load $L \geq m/p^{1/\rho^*} \times 1/r$

Proof Let DB be a "worst" instance $|Q(DB)| = m^{\rho^*}$

A server receives in total at most r × L tuples from each S_j

A server can output at most $(r \times L)^{\rho^*}$ answers from Q(DB)

All p servers output $p \times (r \times L)^{\rho^*} \geq m^{\rho^*}$ answers.

Discussion

- Multi-rounds help mitigate skew penalty
- Optimal load known to be m^{ρ^*} but only in special cases; open in general
- Vertex cover τ^* versus edge cover ρ^*
  - 1-round, no-skew v.s. multi-rounds, skew
  - For graphs: $\tau^* \leq \rho^*$
  - For hypergraphs: no relationship τ^*, ρ^*

Conclusions

Communication cost

- Critical parameter in distributed computing
- Full CQ only: for aggregates, see FAQ
- Shared nothing: but also shared memory