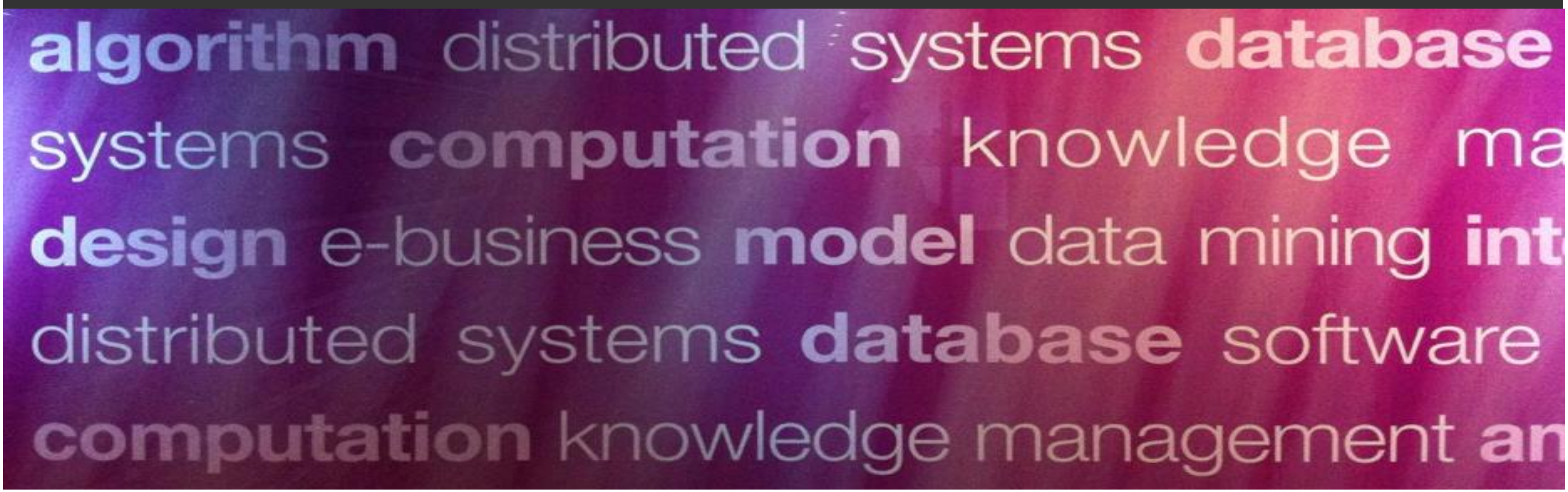
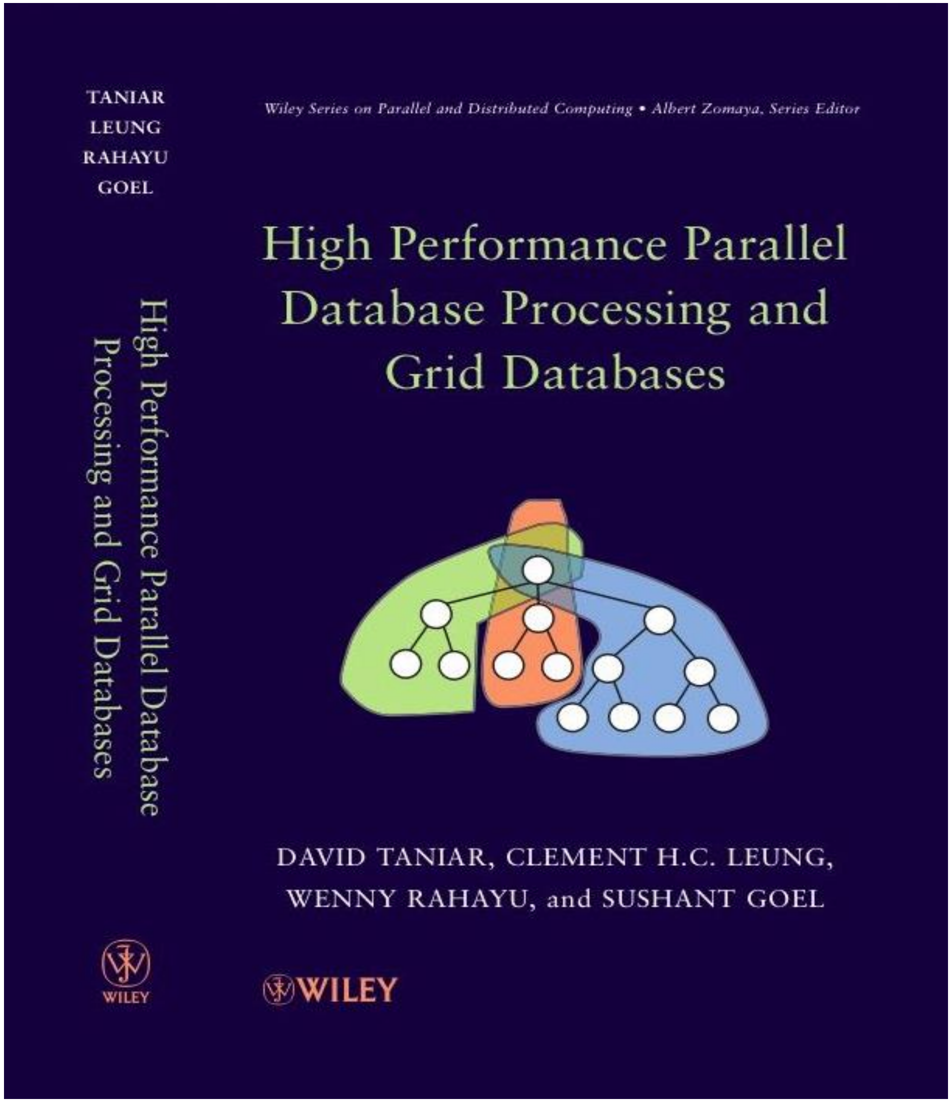
MONASH University

Information Technology

FIT5202 (Volume IV – Sort and Group By)

Week 4b – Parallel Group By





Chapter 4 Parallel Sort and GroupBy

4.1. GroupBy, and Serial GroupBy

```
Select Suburb, Count(*)
From Student
Group By Suburb;
```

Student	Suburb
Adam	Clayton
Ben	Hawthorn
Chris	Doncaster
Daniel	Caulfield
Eric	Kew
Fred	Richmond
Garry	Hawthorn
Harold	Elwood
Irene	Clayton
Jessica	Caulfield
Katie	Malvern
Leonard	Balwyn
Mary	Hawthorn

GroupBy
- Combine rows/records into groups to get some summary
- Records in the same group share the same key
- often used with aggregation (e.g., count, sum, average)

4.1. Serial GroupBy Processing (cont'd)

Select Suburb, Count(*)
From Student
Group By Suburb;

Processing Steps:

1. Read the first student record, and hash the suburb to the hash table

Student	Suburb
Adam	Clayton
Ben	Hawthorn
Chris	Doncaster
Daniel	Caulfield
Eric	Kew
Fred	Richmond
Garry	Hawthorn
Harold	Elwood
Irene	Clayton
Jessica	Caulfield
Katie	Malvern
Leonard	Balwyn
Mary	Hawthorn

Hash the record using a certain hash function

Hash Table

1		
2		
3		
4		
5		
6		
7		
8	Clayton	1
9		

Use Hash table to do grouping

4.1. Serial GroupBy Processing (cont'd)

Select Suburb, Count(*)
From Student
Group By Suburb;

Processing Steps:

1. Read the first student record, and hash the suburb to the hash table
2. Read the second record and hash it

Student	Suburb
Adam	Clayton
Ben	Hawthorn
Chris	Doncaster
Daniel	Caulfield
Eric	Kew
Fred	Richmond
Garry	Hawthorn
Harold	Elwood
Irene	Clayton
Jessica	Caulfield
Katie	Malvern
Leonard	Balwyn
Mary	Hawthorn

Hash Table

1	Hawthorn	1
2		
3		
4		
5		
6		
7		
8	Clayton	1
9		

4.1. Serial GroupBy Processing (cont'd)

Select Suburb, Count(*)
From Student
Group By Suburb;

Processing Steps:

1. Read the first student record, and hash the suburb to the hash table
2. Read the second record and hash it
3. Read the subsequent records one-by-one and hash them

Student	Suburb
Adam	Clayton
Ben	Hawthorn
Chris	Doncaster
Daniel	Caulfield
Eric	Kew
Fred	Richmond
Garry	Hawthorn
Harold	Elwood
Irene	Clayton
Jessica	Caulfield
Katie	Malvern
Leonard	Balwyn
Mary	Hawthorn

Hash Table

1	Hawthorn	3
2	Caulfield	2
3	Malvern	1
4	Balwyn	1
5	Kew	1
6	Richmond	1
7	Elwood	1
8	Clayton	2
9	Doncaster	1

4.1. Serial GroupBy Processing (cont'd)

Select Suburb, Count(*)
From Student
Group By Suburb;

Processing Steps:

1. Read the first student record, and hash the suburb to the hash table
2. Read the second record and hash it
3. Read the subsequent records one-by-one and hash them
4. Read the Hash Table, and store this in disk as the query results

Query Results in Disk

Hawthorn	3
Caulfield	2
Malvern	1
Balwyn	1
Kew	1
Richmond	1
Elwood	1
Clayton	2
Doncaster	1

Hash Table in Main-Memory

1	Hawthorn	3
2	Caulfield	2
3	Malvern	1
4	Balwyn	1
5	Kew	1
6	Richmond	1
7	Elwood	1
8	Clayton	2
9	Doncaster	1

4.1. Serial GroupBy Processing (cont'd)

Select Suburb, Count(*)
From Student
Group By Suburb;

Student	Suburb
Adam	Clayton
Ben	Hawthorn
Leonard	Balwyn
Mary	Hawthorn

This will work, if we assume that the main-memory can hold the entire Hash Table.

How about if the Hash Table is so big that it cannot fit into the main-memory.

For example, how about if the main-memory can only hold 4 hash records at a time? How does the Group By processing work?

Hash Table

1	Hawthorn	3
2	Caulfield	2
3	Malvern	1
4	Balwyn	1
5	Kew	1
6	Richmond	1
7	Elwood	1
8	Clayton	2
9	Doncaster	1

4.1. Serial GroupBy Processing (cont'd)

Student	Suburb
Adam	Clayton
Ben	Hawthorn
Chris	Doncaster
Daniel	Caulfield
Eric	Kew
Fred	Richmond
Garry	Hawthorn
Harold	Elwood
Irene	Clayton
Jessica	Caulfield
Katie	Malvern
Leonard	Balwyn
Mary	Hawthorn

Hash Table

Assume that the main-memory can hold 4 records in the hash table.

It needs a bigger hash table, but it doesn't have.

Student	Suburb
Adam	Clayton
Ben	Hawthorn
Chris	Doncaster
Daniel	Caulfield
Eric	Kew
Fred	Richmond
Garry	Hawthorn
Harold	Elwood
Irene	Clayton
Jessica	Caulfield
Katie	Malvern
Leonard	Balwyn
Mary	Hawthorn

Hash Data Partitioning based on the **Suburb**

Hash values

1	Hawthorn
2	Caulfield
3	Malvern
4	Balwyn
5	Kew
6	Richmond
7	Elwood
8	Clayton
9	Doncaster

In Main-Memory

Ben	Hawthorn
Daniel	Caulfield
Garry	Hawthorn
Jessica	Caulfield
Mary	Hawthorn

In Disk

Eric	Kew
Fred	Richmond
Katie	Malvern
Leonard	Balwyn

Adam	Clayton
Chris	Doncaster
Harold	Elwood
Irene	Clayton

Hash Processing

Ben	Hawthorn
Daniel	Caulfield
Garry	Hawthorn
Jessica	Caulfield
Mary	Hawthorn

Load to Main-Memory

Hash Table in Main-Memory

Hawthorn	3
Caulfield	2

Hash records in each partition into hash table, and perform grouping

Eric	Kew
Fred	Richmond
Katie	Malvern
Leonard	Balwyn

Still in Disk

Adam	Clayton
Chris	Doncaster
Harold	Elwood
Irene	Clayton

One partition will be grouped at a time

Query Results in Disk

Hawthorn	3
Caulfield	2

Hash Processing

Hash Table in Main-Memory

Kew	1
Malvern	1
Richmond	1
Balwyn	1

Flush to Disk

Ben	Hawthorn
Daniel	Caulfield
Garry	Hawthorn
Jessica	Caulfield
Mary	Hawthorn

Load to Main-Memory

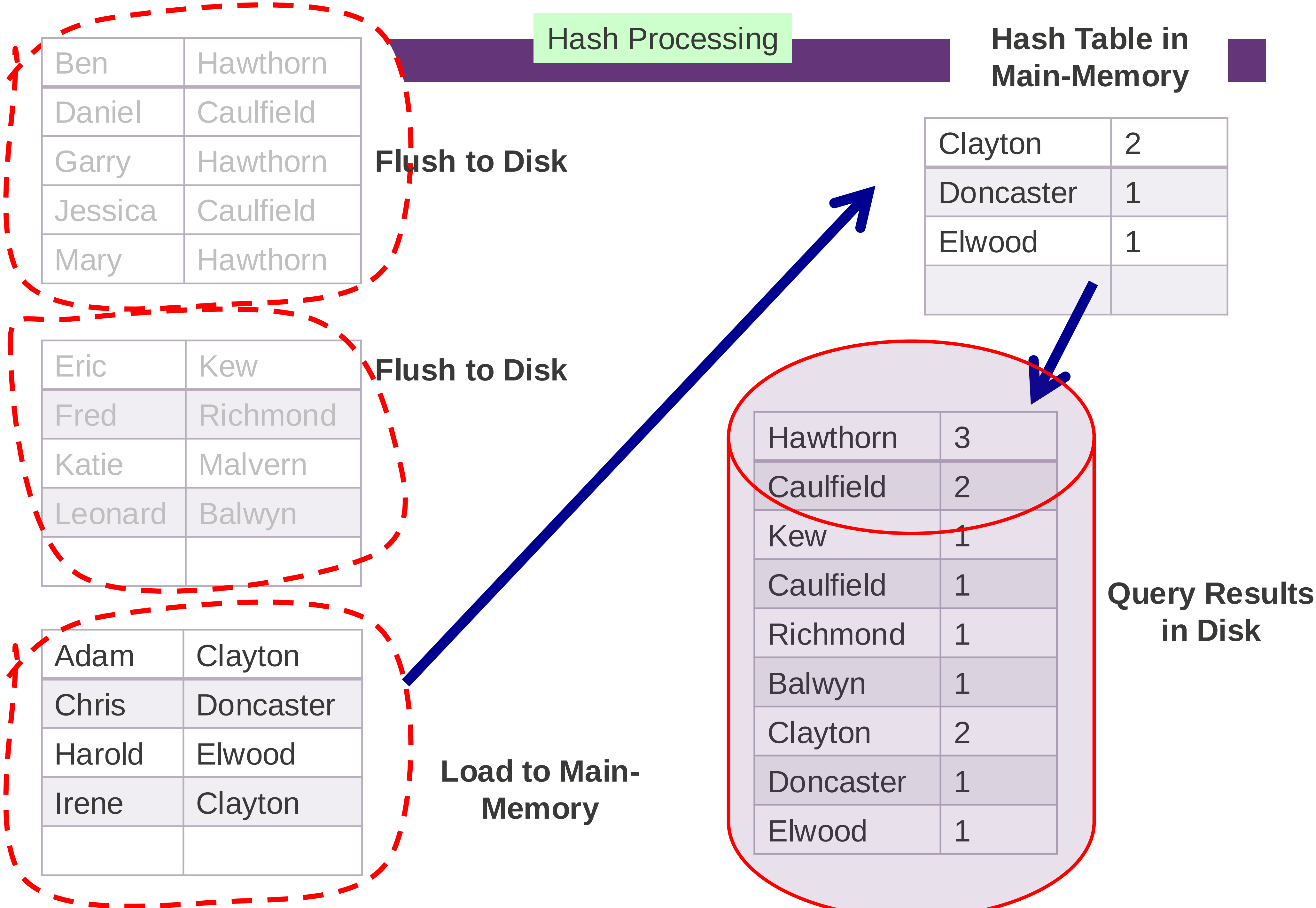
Eric	Kew
Fred	Richmond
Katie	Malvern
Leonard	Balwyn

Still in Disk

Adam	Clayton
Chris	Doncaster
Harold	Elwood
Irene	Clayton

Query Results in Disk

Hawthorn	3
Caulfield	2
Kew	1
Caulfield	1
Richmond	1
Balwyn	1


Hash Processing
Hash Table in Main-Memory
Ben Hawthorn
Daniel Caulfield
Garry Hawthorn
Jessica Caulfield
Mary Hawthorn
Flush to Disk
Eric Kew
Fred Richmond
Katie Malvern
Leonard Balwyn
Flush to Disk
Adam Clayton
Chris Doncaster
Harold Elwood
Irene Clayton
Load to Main-Memory
Clayton 2
Doncaster 1
Elwood 1
Hawthorn 3
Caulfield 2
Kew 1
Caulfield 1
Richmond 1
Balwyn 1
Clayton 2
Doncaster 1
Elwood 1
Query Results in Disk

4.4. Parallel GroupBy

- Traditional methods (Merge-All and Hierarchical Merging)
- Two-phase method
- Redistribution method

Without data redistribution

With data redistribution

4.4. Parallel GroupBy (cont'd)

Step 1: Data partitioning: Each processor is assigned a partition of data
Step 2: **Local groupby/aggregation**

- **Traditional Methods**
  - Step 1: local aggregate in each processor
  - Step 2: global aggregation
    - May use a Merge-All or Hierarchical method
  - Need to pay a special attention to some aggregate functions (AVG) when performing a local aggregate process (e.g. averaging – need to keep sub-total)

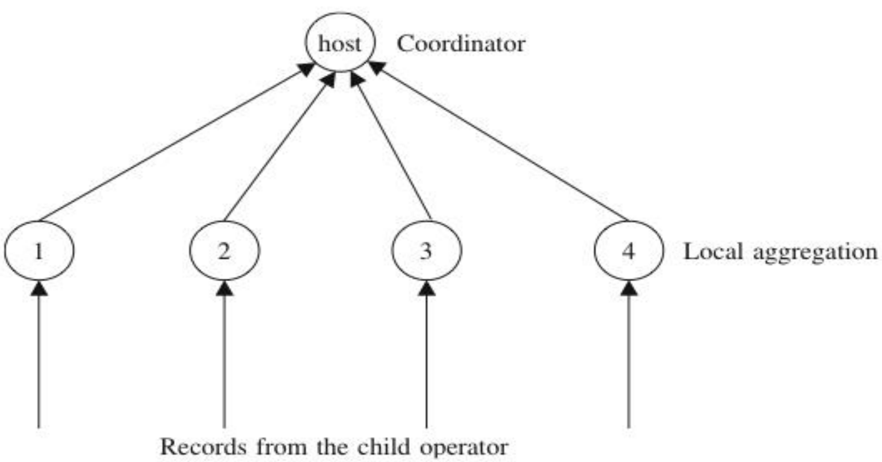


Figure 4.10 Traditional method

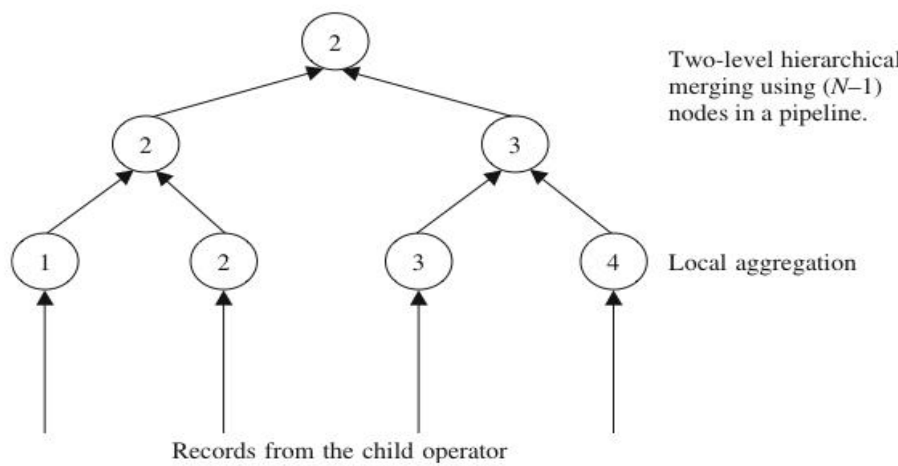
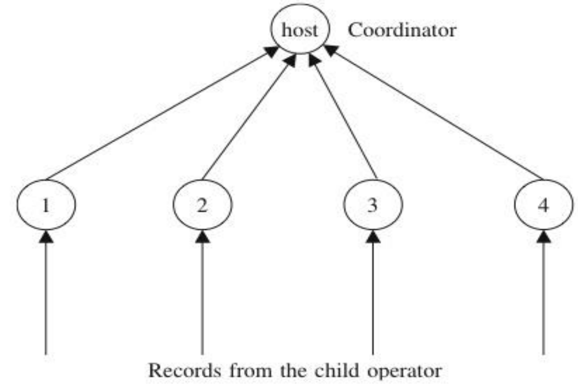


Figure 4.11 Hierarchical merging method

4.4. Parallel GroupBy (cont'd)

- **Traditional Method: Merge All**



Initial Data Placement

Processor 1

Adam	Clayton
Ben	Clayton
Chris	Caulfield
Dennis	Malvern
Eric	Vermont

Processor 2

Fred	Hawthorn
George	Richmond
Harold	Elwood
Irene	Malvern
Jessica	Kew

Processor 3

Kelly	Balwyn
Lesley	Hawthorn
Megan	Kew
Naomi	Richmond
Oscar	Vermont

Processor 4

Peter	Elwood
Quin	Kew
Roger	Balwyn
Sarah	Malvern
Tracy	Clayton

4.4. Parallel GroupBy (cont'd)

- **Traditional Method: Merge All**

Clayton	2
Caulfield	1
Malvern	1
Vermont	1

Hawthorn	1
Richmond	1
Elwood	1
Malvern	1
Kew	1

Balwyn	1
Hawthorn	1
Kew	1
Richmond	1
Vermont	1

Elwood	1
Kew	1
Balwyn	1
Malvern	1
Clayton	1

Local Aggregation Phase

Processor 1

Adam	Clayton
Ben	Clayton
Chris	Caulfield
Dennis	Malvern
Eric	Vermont

Processor 2

Fred	Hawthorn
George	Richmond
Harold	Elwood
Irene	Malvern
Jessica	Kew

Processor 3

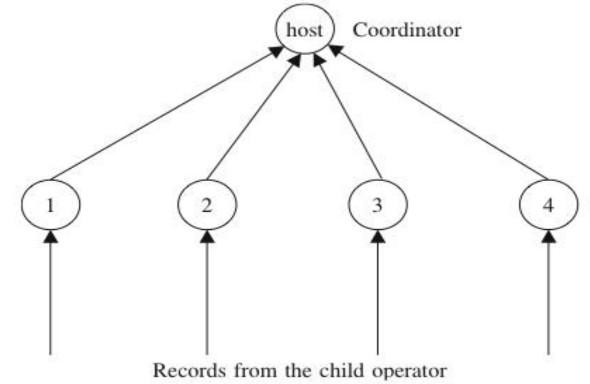
Kelly	Balwyn
Lesley	Hawthorn
Megan	Kew
Naomi	Richmond
Oscar	Vermont

Processor 4

Peter	Elwood
Quin	Kew
Roger	Balwyn
Sarah	Malvern
Tracy	Clayton

4.4. Parallel GroupBy (cont'd)

- **Traditional Method: Merge All**



One processor combines the locally grouped results

Clayton	2
Hawthorn	1
Balwyn	1
Elwood	1
…	…
…	…
Vermont	1
Clayton	1

Global Aggregation Phase

Clayton	2
Caulfield	1
Malvern	1
Vermont	1

Processor 1

Hawthorn	1
Richmond	1
Elwood	1
Malvern	1
Kew	1

Processor 2

Balwyn	1
Hawthorn	1
Kew	1
Richmond	1
Vermont	1

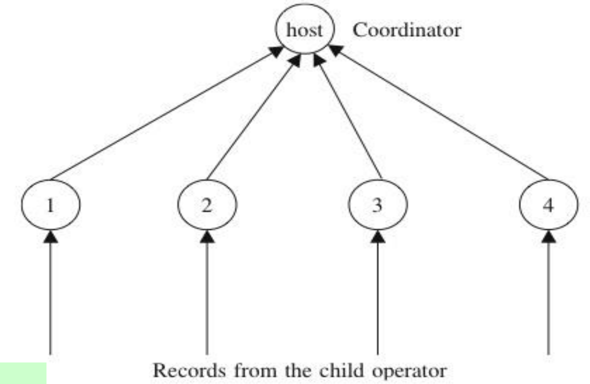
Processor 3

Elwood	1
Kew	1
Balwyn	1
Malvern	1
Clayton	1

Processor 4

4.4. Parallel GroupBy (cont'd)

- **Traditional Method: Merge All**



Clayton	3
Hawthorn	2
Balwyn	2
…	…
…	…
Vermont	2

Final results

Clayton	2
Hawthorn	1
Balwyn	1
Elwood	1
…	…
…	…
Vermont	1
Clayton	1

Global Aggregation Phase

4.4. Parallel GroupBy (cont'd)

- **Two-Phase Method**
  - Step 1: local aggregate in each processor. Each processor groups local records according to the groupby attribute
  - Step 2: global aggregation where all temp results from each processor are redistributed and then final aggregate is performed in each processor

All processors are used for global aggregation

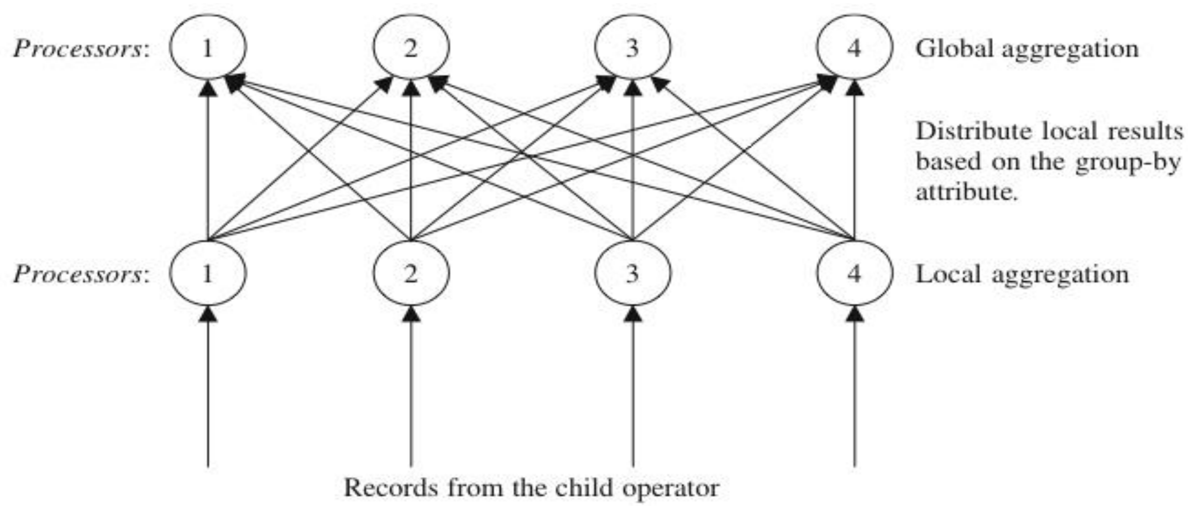
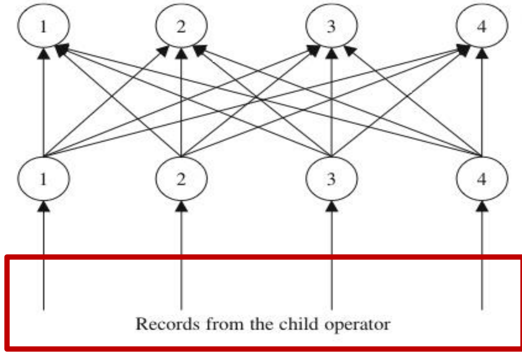


Figure 4.12 Two-phase method

4.4. Parallel GroupBy (cont'd)

- **Two-Phase Method**



Initial Data Placement

Processor 1

Adam	Clayton
Ben	Clayton
Chris	Caulfield
Dennis	Malvern
Eric	Vermont

Processor 2

Fred	Hawthorn
George	Richmond
Harold	Elwood
Irene	Malvern
Jessica	Kew

Processor 3

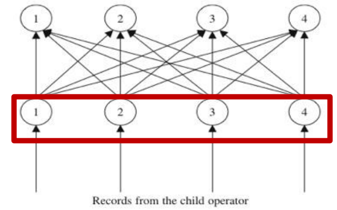
Kelly	Balwyn
Lesley	Hawthorn
Megan	Kew
Naomi	Richmond
Oscar	Vermont

Processor 4

Peter	Elwood
Quin	Kew
Roger	Balwyn
Sarah	Malvern
Tracy	Clayton

4.4. Parallel GroupBy (cont'd)

- **Two-Phase Method**



Clayton	2
Caulfield	1
Malvern	1
Vermont	1

Hawthorn	1
Richmond	1
Elwood	1
Malvern	1
Kew	1

Balwyn	1
Hawthorn	1
Kew	1
Richmond	1
Vermont	1

Elwood	1
Kew	1
Balwyn	1
Malvern	1
Clayton	1

Local Aggregation Phase

Processor 1

Adam	Clayton
Ben	Clayton
Chris	Caulfield
Dennis	Malvern
Eric	Vermont

Processor 2

Fred	Hawthorn
George	Richmond
Harold	Elwood
Irene	Malvern
Jessica	Kew

Processor 3

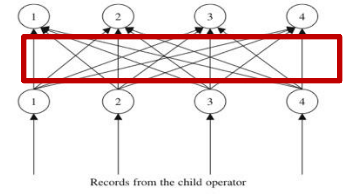
Kelly	Balwyn
Lesley	Hawthorn
Megan	Kew
Naomi	Richmond
Oscar	Vermont

Processor 4

Peter	Elwood
Quin	Kew
Roger	Balwyn
Sarah	Malvern
Tracy	Clayton

4.4. Parallel GroupBy (cont'd)

- **Two-Phase Method**



Clayton	2
Balwyn	1
Elwood	1
Caulfield	1
Elwood	1
Balwyn	1
Clayton	1

A-G

Hawthorn	1
Hawthorn	1
Kew	1
Kew	1
Kew	1

H-L

Malvern	1
Malvern	1
Malvern	1

M-Q

Richmond	1
Vermont	1
Richmond	1
Vermont	1

R-Z

Distribute Local Aggregation Results Phase

Assume range-based re-distribution

Processor 1

Clayton	2
Caulfield	1
Malvern	1
Vermont	1

Processor 2

Hawthorn	1
Richmond	1
Elwood	1
Malvern	1
Kew	1

Processor 3

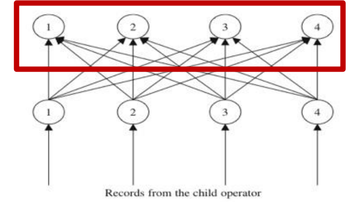
Balwyn	1
Hawthorn	1
Kew	1
Richmond	1
Vermont	1

Processor 4

Elwood	1
Kew	1
Balwyn	1
Malvern	1
Clayton	1

4.4. Parallel GroupBy (cont'd)

- **Two-Phase Method**



Final results

Clayton	3
Balwyn	2
Elwood	2
Caulfield	1

Hawthorn	2
Kew	3

Malvern	3

Richmond	2
Vermont	2

Global Aggregation Phase

Clayton	2
Balwyn	1
Elwood	1
Caulfield	1
Elwood	1
Balwyn	1
Clayton	1

Processor 1

Hawthorn	1
Hawthorn	1
Kew	1
Kew	1
Kew	1

Processor 2

Malvern	1
Malvern	1
Malvern	1

Processor 3

Richmond	1
Vermont	1
Richmond	1
Vermont	1

Processor 4

4.4. Parallel GroupBy (cont'd)

- **Redistribution Method**
  - Step 1 (Partitioning phase): redistribute raw records to all processors
  - Step 2 (Aggregation phase): each processor performs a local aggregation

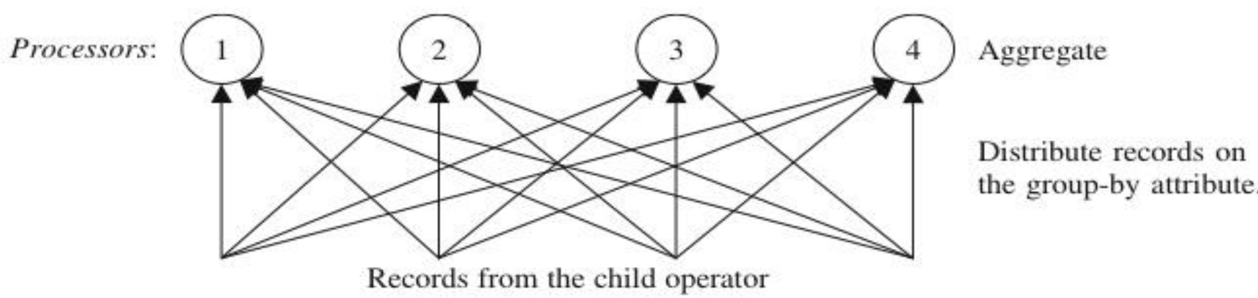
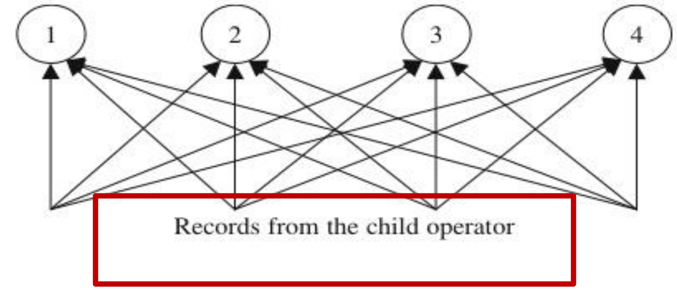


Figure 4.13 Redistribution method

4.4. Parallel GroupBy (cont'd)

- **Redistribution Method**



Initial Data Placement

Processor 1

Adam	Clayton
Ben	Clayton
Chris	Caulfield
Dennis	Malvern
Eric	Vermont

Processor 2

Fred	Hawthorn
George	Richmond
Harold	Elwood
Irene	Malvern
Jessica	Kew

Processor 3

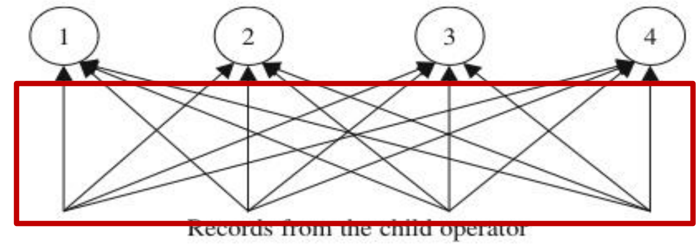
Kelly	Balwyn
Lesley	Hawthorn
Megan	Kew
Naomi	Richmond
Oscar	Vermont

Processor 4

Peter	Elwood
Quin	Kew
Roger	Balwyn
Sarah	Malvern
Tracy	Clayton

4.4. Parallel GroupBy (cont'd)

- **Redistribution Method**



Partitioning Phase

Assume range-based re-distribution based on grouping attribute

Adam	Clayton
Kelly	Balwyn
Peter	Elwood
Ben	Clayton
Chris	Caulfield
Harold	Elwood
Roger	Balwyn
Tracy	Clayton

Fred	Hawthorn
Lesley	Hawthorn
Quin	Kew
Megan	Kew
Jessica	Kew

Dennis	Malvern
Irene	Malvern
Sarah	Malvern

George	Richmond
Naomi	Richmond
Eric	Vermont
Oscar	Vermont

Processor 1 **Processor 2** **Processor 3** **Processor 4**

Adam	Clayton
Ben	Clayton
Chris	Caulfield
Dennis	Malvern
Eric	Vermont

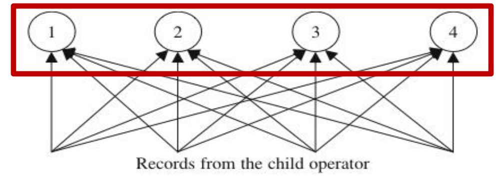
Fred	Hawthorn
George	Richmond
Harold	Elwood
Irene	Malvern
Jessica	Kew

Kelly	Balwyn
Lesley	Hawthorn
Megan	Kew
Naomi	Richmond
Oscar	Vermont

Peter	Elwood
Quin	Kew
Roger	Balwyn
Sarah	Malvern
Tracy	Clayton

4.4. Parallel GroupBy (cont'd)

▪ Redistribution Method



Final results

Clayton	3
Balwyn	2
Elwood	2
Caulfield	1

Hawthorn	2
Kew	3

Malvern	3

Richmond	2
Vermont	2

Aggregation Phase

Adam	Clayton
Kelly	Balwyn
Peter	Elwood
Ben	Clayton
Chris	Caulfield
Harold	Elwood
Roger	Balwyn
Tracy	Clayton

Processor 1

Fred	Hawthorn
Lesley	Hawthorn
Quin	Kew
Megan	Kew
Jessica	Kew

Processor 2

Dennis	Malvern
Irene	Malvern
Sarah	Malvern

Processor 3

George	Richmond
Naomi	Richmond
Eric	Vermont
Oscar	Vermont

Processor 4

4.4. Parallel GroupBy (cont'd)

- **Redistribution Method**

Clayton	3
Balwyn	2
Elwood	2
Caulfield	1

Hawthorn	2
Kew	3

Malvern	3

Richmond	2
Vermont	2

- Skewness: uneven workload after re-distribution

What is the problem here?

Adam	Clayton
Kelly	Balwyn
Peter	Elwood
Ben	Clayton
Chris	Caulfield
Harold	Elwood
Roger	Balwyn
Tracy	Clayton

Processor 1

Fred	Hawthorn
Lesley	Hawthorn
Quin	Kew
Megan	Kew
Jessica	Kew

Processor 2

Dennis	Malvern
Irene	Malvern
Sarah	Malvern

Processor 3

George	Richmond
Naomi	Richmond
Eric	Vermont
Oscar	Vermont

Processor 4

4.4. Parallel GroupBy (cont'd)

Redistribution Method (Task Stealing)

Create 5 buckets, instead of 4

Solution:
To achieve load balancing, create more partitions than the number of processors, then rearrange them to processors

Adam	Clayton
Kelly	Balwyn
Ben	Clayton
Chris	Caulfield
Roger	Balwyn
Tracy	Clayton

Peter	Elwood
Harold	Elwood

Fred	Hawthorn
Lesley	Hawthorn
Quin	Kew
Megan	Kew
Jessica	Kew

Dennis	Malvern
Irene	Malvern
Sarah	Malvern

George	Richmond
Naomi	Richmond
Eric	Vermont
Oscar	Vermont

Processor 1

Adam	Clayton
Ben	Clayton
Chris	Caulfield
Dennis	Malvern
Eric	Vermont

Processor 2

Fred	Hawthorn
George	Richmond
Harold	Elwood
Irene	Malvern
Jessica	Kew

Processor 3

Kelly	Balwyn
Lesley	Hawthorn
Megan	Kew
Naomi	Richmond
Oscar	Vermont

Processor 4

Peter	Elwood
Quin	Kew
Roger	Balwyn
Sarah	Malvern
Tracy	Clayton

4.4. Parallel GroupBy (cont'd)

▪ Redistribution Method (Task Stealing)

P3 will likely to finish groupby processing earlier, can process additional chunk from P1

Processor 1

Adam	Clayton
Kelly	Balwyn
Ben	Clayton
Chris	Caulfield
Roger	Balwyn
Tracy	Clayton

Peter	Elwood
Harold	Elwood

Processor 2

Fred	Hawthorn
Lesley	Hawthorn
Quin	Kew
Megan	Kew
Jessica	Kew

Processor 3

Dennis	Malvern
Irene	Malvern
Sarah	Malvern

Task stealing

Peter	Elwood
Harold	Elwood

Processor 4

George	Richmond
Naomi	Richmond
Eric	Vermont
Oscar	Vermont

4.4. Parallel GroupBy (cont'd)

. Redistribution Method (Task Stealing)

Clayton	3
Balwyn	2
Caulfield	1

Hawthorn	2
Kew	3

Malvern	3
Elwood	2

Richmond	2
Vermont	2

Adam	Clayton
Kelly	Balwyn
Ben	Clayton
Chris	Caulfield
Roger	Balwyn
Tracy	Clayton

Fred	Hawthorn
Lesley	Hawthorn
Quin	Kew
Megan	Kew
Jessica	Kew

Dennis	Malvern
Irene	Malvern
Sarah	Malvern

Peter	Elwood
Harold	Elwood

George	Richmond
Naomi	Richmond
Eric	Vermont
Oscar	Vermont

Processor 1 **Processor 2** **Processor 3** **Processor 4**

4.7. Summary

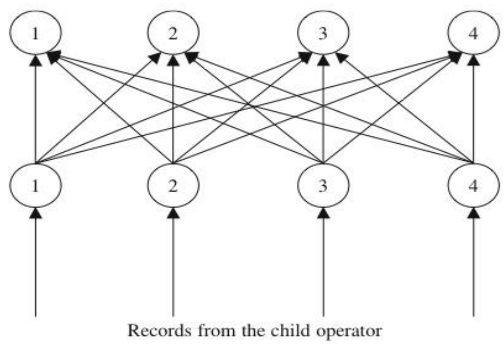
- Parallel groupby algorithms
  - Traditional methods (merge-all and hierarchical methods)
  - Two-phase method – Local aggregation before data redistribution
  - Redistribution method - Local aggregation after data redistribution

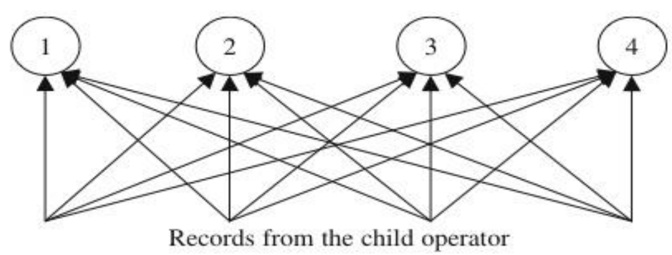
- **Two-phase** and **Redistribution** methods perform better than the traditional and hierarchical merging methods

- **Two-phase method** works well when the number of groups is small, whereas the **Redistribution method** works well when the number of groups is large

Ambuj and Naughton. "Adaptive parallel aggregation algorithms." (1995):

Why??





For two-phase, if the number of groups is large, many duplicates of aggregation in local and global phases