## Session 1

## parallel processing

**Reason**

- Parallel processing divides a large task into smaller subtasks
- Database processing works well with parallelism (coarse-grained parallelism)
- Lesser complexity but need to work with a large volume of data

The primary objective of parallel database processing is to gain performance improvement

**Measure**

**Throughput**: the number of tasks that can be completed within a given time interval

**Response time**: the amount of time it takes to complete a single task from the time it is submitted

**Speed up:**

- Performance improvement gained because of extra processing elements added
- Running a given task in less time by increasing the degree of parallelism

**Scale up:**

- Handling of larger tasks by increasing the degree of parallelism
- The ability to process larger tasks in the same amount of time by providing more resources.

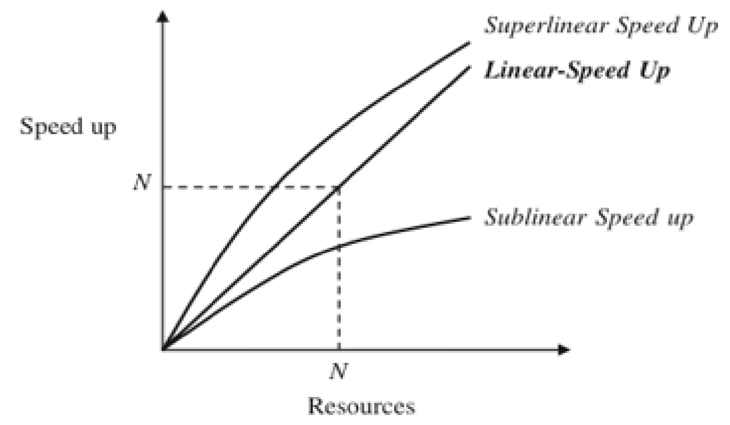


**Figure 1.1** Speed up

### Session 2

## Parallel Obstacles

**Start-up and Consolidation costs**

- Start up: initiation of multiple processes
- Consolidation: the cost for collecting results obtained from each processor by a host processor

**Interference and Communication, and**

- Interference: competing to access shared resources
- Communication: one process communicating with other processes, and often one has to wait for others to be ready for communication (i.e. waiting time).

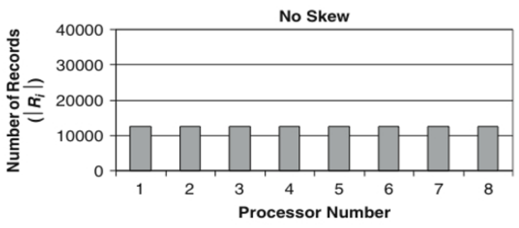
**Skew**

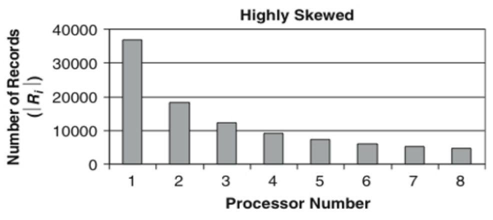
- Unevenness of workload

- **Skew**
  - *Zipf* distribution model to model skew. Measured in terms of different sizes of fragments allocated to the processors

$$|R_i| = \frac{|R|}{i^\theta \times \sum_{j=1}^{N} \frac{1}{j^\theta}} \qquad \text{where } 0 \le \theta \le 1 \qquad (2.1)$$

  - The symbol θ denotes the degree of skewness, where θ = 0 indicates no skew, and **θ = 1 indicates highly skewed**
  - |*R*| is number of records in the table, **|*Ri*| is number of records in processor *i***, and ***N*** is number of processor (*j* is a loop counter, starting from 1 to *N*)
  - Example: |*R*|=100,000 records, *N*=8 processors



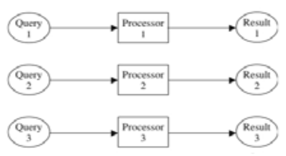


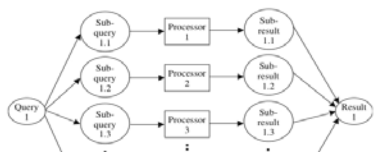
## Forms of Parallelism

➢ **Interquery parallelism**

Speeding up the processing of a query by executing in parallel different operations in a query expression. (e.g. simultaneous sorting or searching)

- Different queries or transactions are executed in parallel with one another
- Main aim: scaling up transaction processing systems





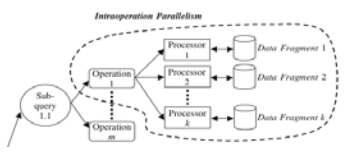
- **Intraquery parallelism**

Speeding up the processing of a query by parallelizing the execution of each individual operation (e.g. parallel sort, parallel search, etc)

  – Execution of a single query in parallel on multiple processors and disks

  – Main aim: speeding up long-running queries

- **Interoperation parallelism**
  – Parallelism created by concurrently executing different operations within the same query or transaction
- **Intraoperation parallelism**



Parallelism due to the data being partitioned

- **Pipeline Parallelism**

  – Multiple operations form some sort of assembly line to manufacture the query results

- **Independent Parallelism**

  – Operations in a query that do not depend on one another are executed in parallel

## Parallel computers

- **Shared-memory architecture**

  – Shared-Memory: all processors share a common main memory and secondary memory

  – Load balancing is relatively easy to achieve, but suffer from memory and bus contention

- **Shared-disk architecture**

  – Shared-Disk: all processors, each of which has its own local main memory, share the disks

- **Shared-nothing architecture**

  – Shared-Disk: all processors, each of which has its own local main memory, share the disks

  – Load balancing becomes difficult

- **Shared-something architecture**

## Basic Data Partitioning

- **Round-robin data partitioning**
  Each record in turn is allocated to a processing element in a clockwise manner
  Data evenly distributed, hence supports load balance
- **Hash data partitioning**
  A hash function is used to partition the data
  Hence, data is grouped semantically, that is data on the same group shared the same hash value

- **Range data partitioning**
  Spreads the records based on a given range of the partitioning attribute
- **Random-unequal data partitioning**
  This is common especially when the operation is actually an operation based on temporary results obtained from the previous operations

### Complex Data Partitioning

- **Hybrid-Range Partitioning Strategy (HRPS)**

Partitions the table into many fragments using range, and the fragments are distributed to all processors using round-robin

- **Multiattribute Grid Declustering (MAGIC)**

Based on multiple attributes - to support search queries based on either of data partitioning attributes
Support range and exact match search on each of the partitioning attributes

## Search Algorithms

- **Processor activation or involvement**

The number of processors to be used by the algorithm
If we know where the data to be sought are stored, then there is no point in activating all other processors in the searching process
Depends on the data partitioning method use

- **Local searching method**

The searching method applied to the processor(s) involved in the searching process
Depends on the data ordering, regarding the type of the search (exact match of range)

- **Key comparison**

Compare the data from the table with the condition specified by the query
When a match is found: continue to find other matches, or terminate
Depends on whether the data in the table is unique or not

### Session 3 - Join

## Parallel Join Algorithms

Parallelism of join queries is achieved through *data parallelism*, whereby the same task is applied to different parts of the data. After data partitioning is completed, each processor will have its own data to work with using any serial join algorithm

***Divide and Broadcast***

Two stages: data partitioning using the divide and broadcast method, and a local join

Divide one table into multiple disjoint partitions(range or hash partitioning), where each partition is allocated a processor, and broadcast the other table to all available processors

- **Serial Join Algorithms**

– Nested loop join algorithm

For each record of table *R*, it goes through all records of table *S*

– Sort-merge join algorithm

Both tables must be pre-sorted based on the join attribute(s). If not, then both tables must be sorted first, then merge the two sorted tables

– Hash-based join algorithm

The records of files *R* and *S* both hashed to the *same hash file*, using the *same hashing function*on the join attributes *A*of *R*and *B*of *S*as hash keys

## ➢ Cost Models for Parallel Join

- *Scan cost* for loading data from local disk in each processor is:

  $$(Si / P) \times IO$$

- *Select cost* for getting record out of data page is:

  $$|Si| \times (tr + tw)$$

- *Join* costs involve reading, hashing, and probing:
  $(|Ri| \times (tr + th) + (|S| \times (tr + th + tj))$

- *Generating result records* cost is: $|Ri| \times \sigma j \times |S| \times tw$

- *Disk* cost for storing the final result is: $(\pi_R \times Ri \times \sigma j \times \pi_S \times S / P) \times IO$

Storing cost = (S/P – Si/P) x (IO)

**|S| = 600 records, each record has the length of 100 bytes, and N=3.**

Solution:

S = 60000 bytes
Si = S/N = 60000/3 = 20000 bytes

|S| = 600 records
|Si| = 600/3 = 200 records

**Divide and Broadcast based parallel join**

**Transfer cost = (Si/P) x (N-1) x (mp+ml)**

**Receiving cost = (S/P – Si/P) x (mp)**

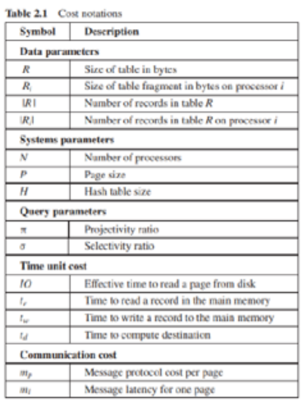


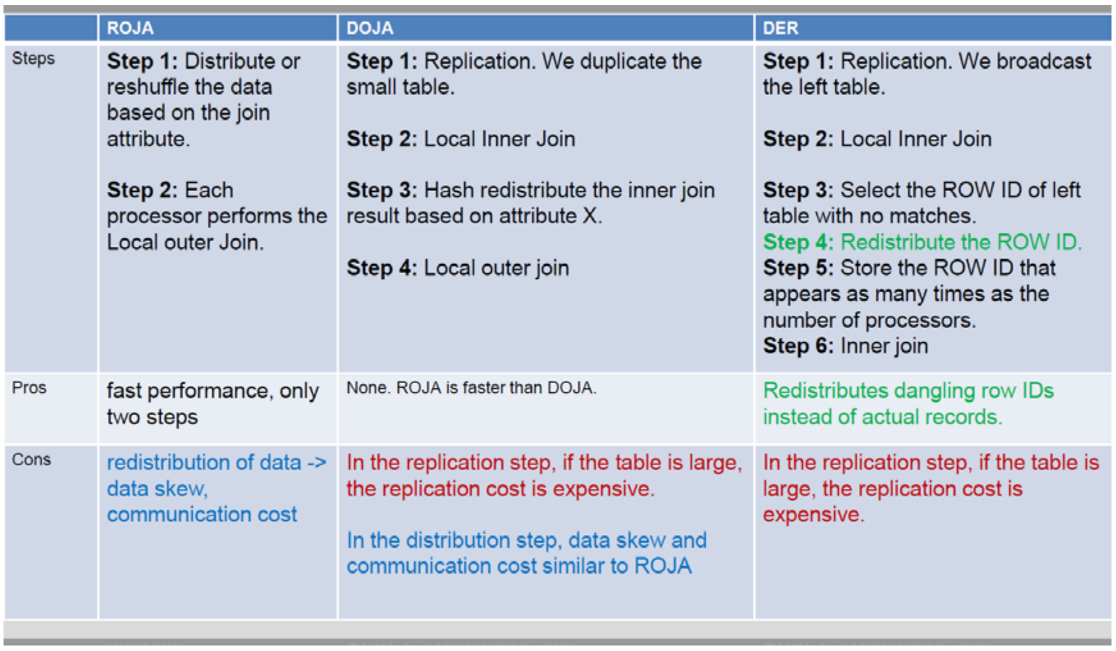
**Table 2.1** Cost notations

| Symbol | Description |
|---|---|
| **Data parameters** | |
| $R$ | Size of table in bytes |
| $R_i$ | Size of table fragment in bytes on processor $i$ |
| $\lvert R \rvert$ | Number of records in table $R$ |
| $\lvert R_i \rvert$ | Number of records in table $R$ on processor $i$ |
| **Systems parameters** | |
| $N$ | Number of processors |
| $P$ | Page size |
| $H$ | Hash table size |
| **Query parameters** | |
| $\pi$ | Projectivity ratio |
| $\sigma$ | Selectivity ratio |
| **Time unit cost** | |
| $IO$ | Effective time to read a page from disk |
| $t_r$ | Time to read a record in the main memory |
| $t_w$ | Time to write a record to the main memory |
| $t_d$ | Time to compute destination |
| **Communication cost** | |
| $m_p$ | Message protocol cost per page |
| $m_l$ | Message latency for one page |

**Table 2.1** Cost notations

| Symbol | Description |
|---|---|
| **Data parameters** | |
| $R$ | Size of table in bytes |
| $R_i$ | Size of table fragment in bytes on processor $i$ |
| $\|R\|$ | Number of records in table $R$ |
| $\|R_i\|$ | Number of records in table $R$ on processor $i$ |
| **Systems parameters** | |
| $N$ | Number of processors |
| $P$ | Page size |
| $H$ | Hash table size |
| **Query parameters** | |
| $\pi$ | Projectivity ratio |
| $\sigma$ | Selectivity ratio |
| **Time unit cost** | |
| $IO$ | Effective time to read a page from disk |
| $t_r$ | Time to read a record in the main memory |
| $t_w$ | Time to write a record to the main memory |
| $t_d$ | Time to compute destination |
| **Communication cost** | |
| $m_p$ | Message protocol cost per page |
| $m_l$ | Message latency for one page |

## ➢ Parallel Outer Join



| | ROJA | DOJA | DER |
|---|---|---|---|
| Steps | **Step 1:** Distribute or reshuffle the data based on the join attribute.<br><br>**Step 2:** Each processor performs the Local outer Join. | **Step 1:** Replication. We duplicate the small table.<br><br>**Step 2:** Local Inner Join<br><br>**Step 3:** Hash redistribute the inner join result based on attribute X.<br><br>**Step 4:** Local outer join | **Step 1:** Replication. We broadcast the left table.<br><br>**Step 2:** Local Inner Join<br><br>**Step 3:** Select the ROW ID of left table with no matches.<br>**Step 4:** Redistribute the ROW ID.<br>**Step 5:** Store the ROW ID that appears as many times as the number of processors.<br>**Step 6:** Inner join |
| Pros | fast performance, only two steps | None. ROJA is faster than DOJA. | Redistributes dangling row IDs instead of actual records. |
| Cons | redistribution of data -> data skew, communication cost | In the replication step, if the table is large, the replication cost is expensive.<br><br>In the distribution step, data skew and communication cost similar to ROJA | In the replication step, if the table is large, the replication cost is expensive. |

**OJSO (Outer Join Skew Optimization)**

Step 1: Redistribute R and S (same as the previous example)
Step 2: (a) Outer Join R and S, but the results are divided into $J_{2redis}$ and $J_{local}$
Step 2: (b) Redistribute $J_{2redis}$ and T; and do an outer join
Step 3: Union the final results in each processor

Session 4

## Parallel External Sort

External sorting assumes that the data does not fit into main memory

### ➢ Parallel Merge-All Sort

Two phases: local sort and final merge
Problems with merging:Heavy load on one processor

### ➢ Parallel Binary-Merge Sort

Local sort similar to traditional method, Merging in pairs only (两个两个merge)
[见下图1]

### ➢ Parallel Redistribution Binary-Merge Sort

Redistribute: 把数据重新按它的range进行重分区（1-5，6-10…）

### ➢ Parallel Redistribution Merge-All Sort

[见下图2]

### ➢ Parallel Partitioned Sort

[见下图3]

## Parallel Redistribution Binary-Merge Sort

Parallelism at all levels in the pipeline hierarchy
Step 1: local sort
Step 2: redistribute the results of local sort
Step 3: merge using the same pool of processors

Benefit: merging becomes lighter than without redistribution
Problem: height of the tree

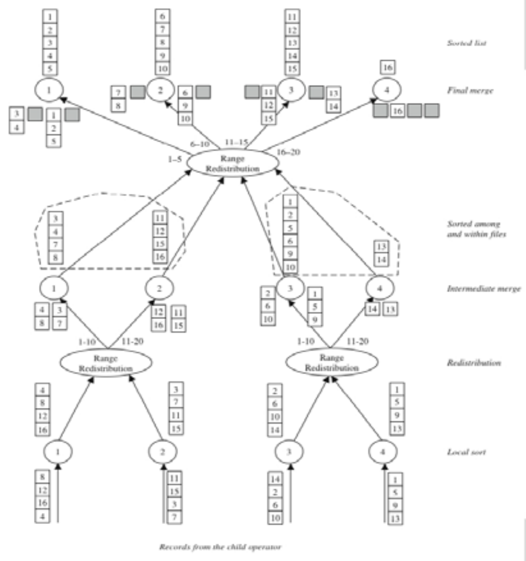


Figure 4.6 Parallel redistribution binary-merge sort

MONASH University

## Parallel Redistribution Merge-All Sort

- Reduce the height of the tree, and still maintain parallelism
- Like parallel merge-all sort, but with redistribution
- The advantage is true parallelism in merging
- Skew problem in the merging

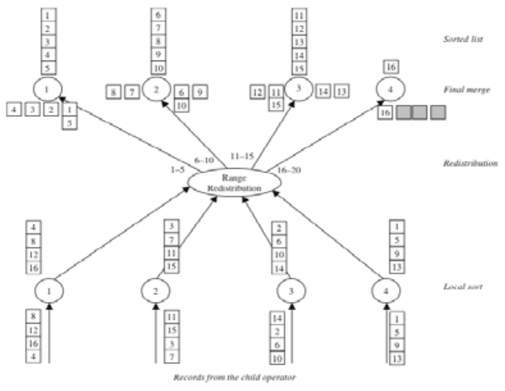
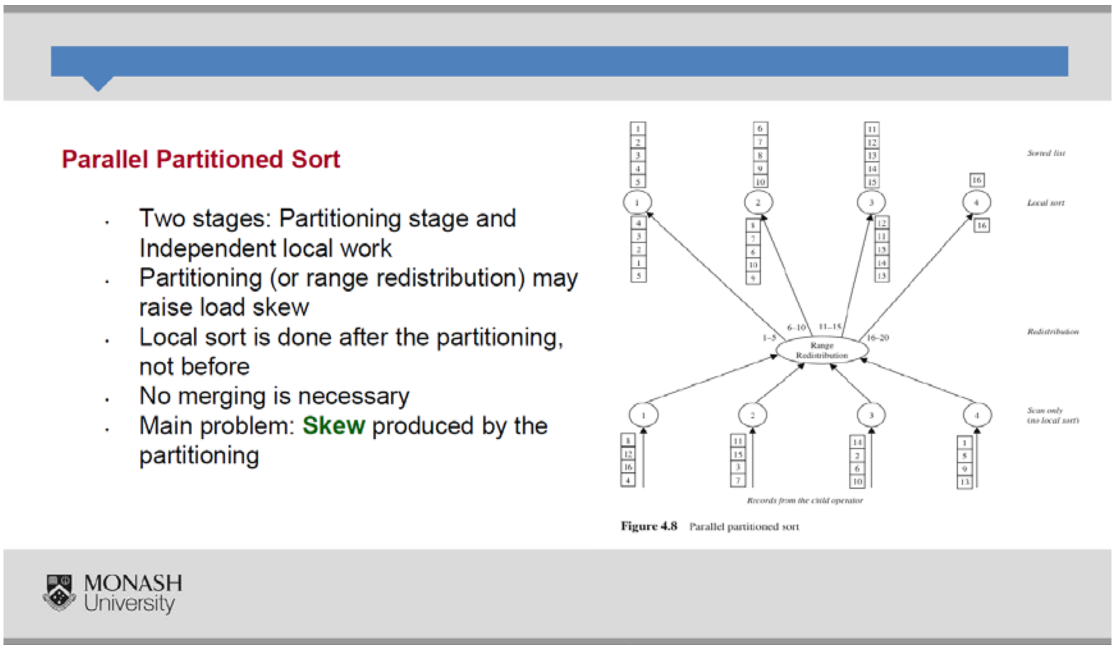


Figure 4.7 Parallel redistribution merge-all sort

MONASH University



**Parallel Partitioned Sort**

- Two stages: Partitioning stage and Independent local work
- Partitioning (or range redistribution) may raise load skew
- Local sort is done after the partitioning, not before
- No merging is necessary
- Main problem: **Skew** produced by the partitioning

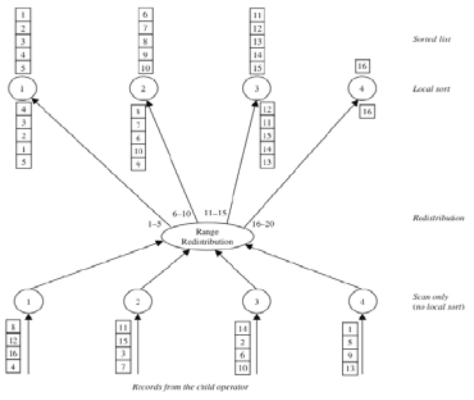


**Figure 4.8** Parallel partitioned sort

MONASH University

## Parallel Group By

➤ **Traditional Methods**

Step 1: local aggregate in each processor
Step 2: global aggregation

➤ **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

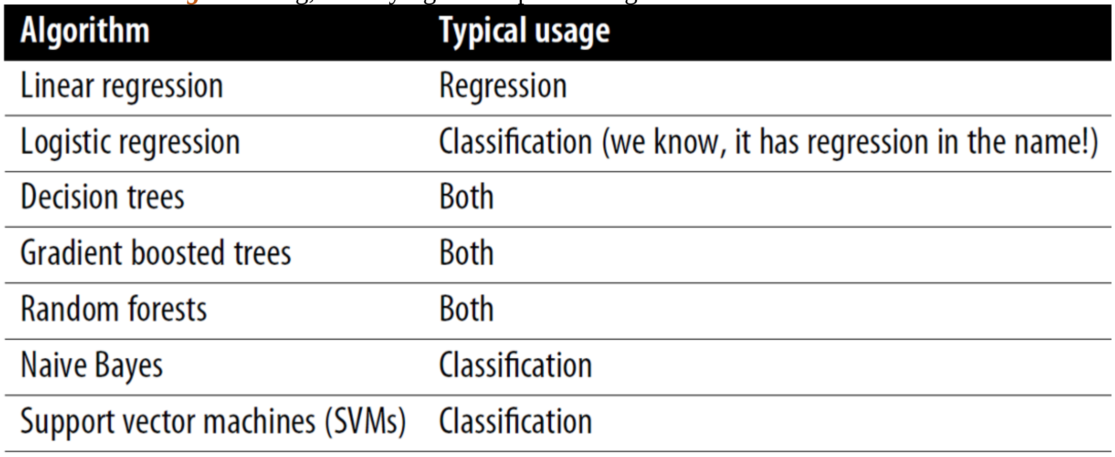
➤ **Redistribution Method**

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

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
数据小的时候local aggregate花费少，大数据的时候local aggregate花费多。

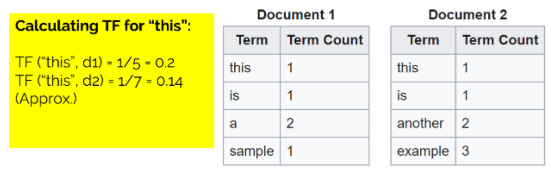
## Session 5 – Machine Learning

***Model***: A specification of a mathematical (or probabilistic) relationship that exist between multiple different variables
***Machine learning***: Creating, Modifying and Implementing models that are learnt from data



| Algorithm | Typical usage |
|---|---|
| Linear regression | Regression |
| Logistic regression | Classification (we know, it has regression in the name!) |
| Decision trees | Both |
| Gradient boosted trees | Both |
| Random forests | Both |
| Naive Bayes | Classification |
| Support vector machines (SVMs) | Classification |

**Unsupervised** : Instead of predicting a label, unsupervised ML helps you to better understand the structure of your data.
**Bias:** is the gap between the value predicted by the model and the actual value of the data
**Variance:** measures the distance of the predicted values in relation to each other.
**Overfitting** (high variance, low bias) is a model that performs well on the training data but generalizes poorly to any new data
**Underfitting** (low variance, high bias) is an overly simple model that does not perform well even on the training data.
**Precision:** *measures the % of the correct classification from the predicted members: true positives / (true positives + false positives)*
**Recall:** *measures the % of the correct classification from the overall members: true positives / (true positives + false negatives)*
**F1:** *measures the balances of precision and recall: 2*((precision*recall) / (precision + recall))*
**String Indexing:** Encoding a string column of labels to a column of label indices
**One Hot Encoding:** Maps a categorical feature represented as a label index to a binary vector. Using this encoding and allowing the model to assume a natural ordering between categories may result in poor performance or unexpected results.
**TF-IDF**
*The term frequency (TF)*, which is the number of times the term occurs in that document, and
*The inverse document frequency (IDF)*, which measures how (in)frequently a term occurs across the whole document corpus.



**Calculating TF for "this":**

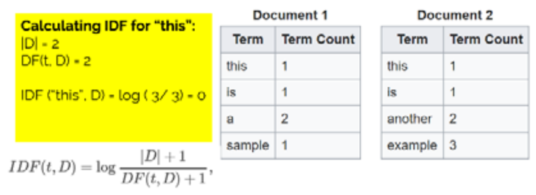
TF ("this", d1) = 1/5 = 0.2
TF ("this", d2) = 1/7 = 0.14
(Approx.)

Document 1

| Term | Term Count |
|---|---|
| this | 1 |
| is | 1 |
| a | 2 |
| sample | 1 |

Document 2

| Term | Term Count |
|---|---|
| this | 1 |
| is | 1 |
| another | 2 |
| example | 3 |



**Calculating IDF for "this":**
|D| = 2
DF(t, D) = 2

IDF ("this", D) = log ( 3/ 3) = 0

$$IDF(t,D) = \log\frac{|D|+1}{DF(t,D)+1},$$

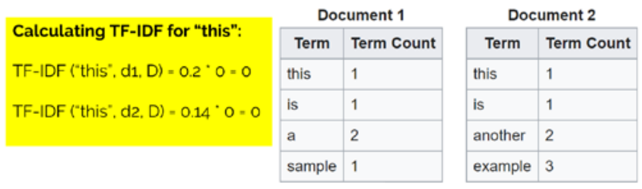
Document 1

| Term | Term Count |
|---|---|
| this | 1 |
| is | 1 |
| a | 2 |
| sample | 1 |

Document 2

| Term | Term Count |
|---|---|
| this | 1 |
| is | 1 |
| another | 2 |
| example | 3 |

先算TF,就是这个词出现的次数除以文件里的总单词数量
|D|就是文档的数量, DF就是这个单词出现在多少个文档里, 这里两个文档都有了就是2
再两个相乘



**Calculating TF-IDF for "this":**

TF-IDF ("this", d1, D) = 0.2 * 0 = 0

TF-IDF ("this", d2, D) = 0.14 * 0 = 0

Document 1

| Term | Term Count |
|---|---|
| this | 1 |
| is | 1 |
| a | 2 |
| sample | 1 |

Document 2

| Term | Term Count |
|---|---|
| this | 1 |
| is | 1 |
| another | 2 |
| example | 3 |

$$TFIDF(t,d,D) = TF(t,d) \cdot IDF(t,D).$$

**Data parallelism:** ***Vertical Partitioning of Training dataset （按列划分）***
**Result parallelism:** ***Horizontal Data Partitioning(横向划分)***

## Classification Algorithms

### ➢ Decision Trees

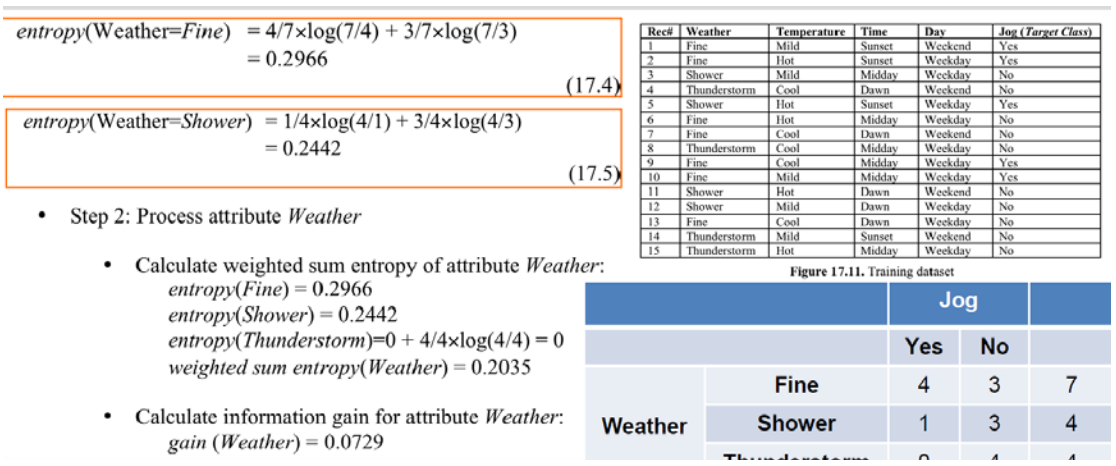
It splits the sample into two or more homogeneous sets (leaves) based on the most significant differentiators in the input variables.

*Advantages*: Easy to generate rules.There are almost null hyper-parameters to be tuned.
*Disadvantages*:Might suffer from overfitting.Does not easily work with non-numerical data.

### ➢ Random Forest

**Random forest** (or **random forests**) is an ensemble classifier that consists of many decision trees and outputs the class that is the mode of the class's output by individual trees.

### ➢ ID3 计算过程



$$entropy(\text{Weather}=Fine) = 4/7\times\log(7/4) + 3/7\times\log(7/3) = 0.2966 \quad (17.4)$$

$$entropy(\text{Weather}=Shower) = 1/4\times\log(4/1) + 3/4\times\log(4/3) = 0.2442 \quad (17.5)$$

- Step 2: Process attribute *Weather*
  - Calculate weighted sum entropy of attribute *Weather*:
    *entropy*(*Fine*) = 0.2966
    *entropy*(*Shower*) = 0.2442
    *entropy*(*Thunderstorm*)=0 + 4/4×log(4/4) = 0
    *weighted sum entropy*(*Weather*) = 0.2035
  - Calculate information gain for attribute *Weather*:
    *gain* (*Weather*) = 0.0729

| Rec# | Weather | Temperature | Time | Day | Jog (*Target Class*) |
|---|---|---|---|---|---|
| 1 | Fine | Mild | Sunset | Weekend | Yes |
| 2 | Fine | Hot | Sunset | Weekday | Yes |
| 3 | Shower | Mild | Midday | Weekday | No |
| 4 | Thunderstorm | Cool | Dawn | Weekend | No |
| 5 | Shower | Hot | Sunset | Weekday | Yes |
| 6 | Fine | Hot | Midday | Weekday | No |
| 7 | Fine | Cool | Dawn | Weekend | No |
| 8 | Thunderstorm | Cool | Midday | Weekday | No |
| 9 | Fine | Cool | Midday | Weekday | Yes |
| 10 | Fine | Mild | Midday | Weekday | Yes |
| 11 | Shower | Hot | Dawn | Weekend | No |
| 12 | Shower | Mild | Dawn | Weekday | No |
| 13 | Fine | Cool | Dawn | Weekday | No |
| 14 | Thunderstorm | Mild | Sunset | Weekend | No |
| 15 | Thunderstorm | Hot | Midday | Weekday | No |

**Figure 17.11.** Training dataset

| | | Jog | | |
|---|---|---|---|---|
| | | Yes | No | |
| Weather | Fine | 4 | 3 | 7 |
| | Shower | 1 | 3 | 4 |

$$\begin{aligned}\text{Weighted sum entropy } (Weather) &= \text{Weighted entropy } (Fine)\\ &\quad + \text{Weighted entropy } (Shower)\\ &\quad + \text{Weighted entropy } (Thunderstorm)\\ &= 7/15 \times 0.2966 + 4/15 \times 0.2442 + 4/15 \times 0\\ &= 0.2035\end{aligned} \quad (17.6)$$

$$\begin{aligned}gain(Weather) &= entropy(\text{training dataset } D) - entropy(\text{attribute } Weather)\\ &= 0.2764 - 0.2035\\ &= 0.0729\end{aligned} \quad (17.7)$$

➢ **Optimisations(最佳化)**

*Bagging*: Bootstrap aggregating is a method that result in low variance
Gradient boosting: selecting best classifiers to improve prediction accuracy with each new tree.

*Advantages*: It is robust to correlated predictors. solve both regression and classification problems. It can handle thousands of input variables without variable selection.

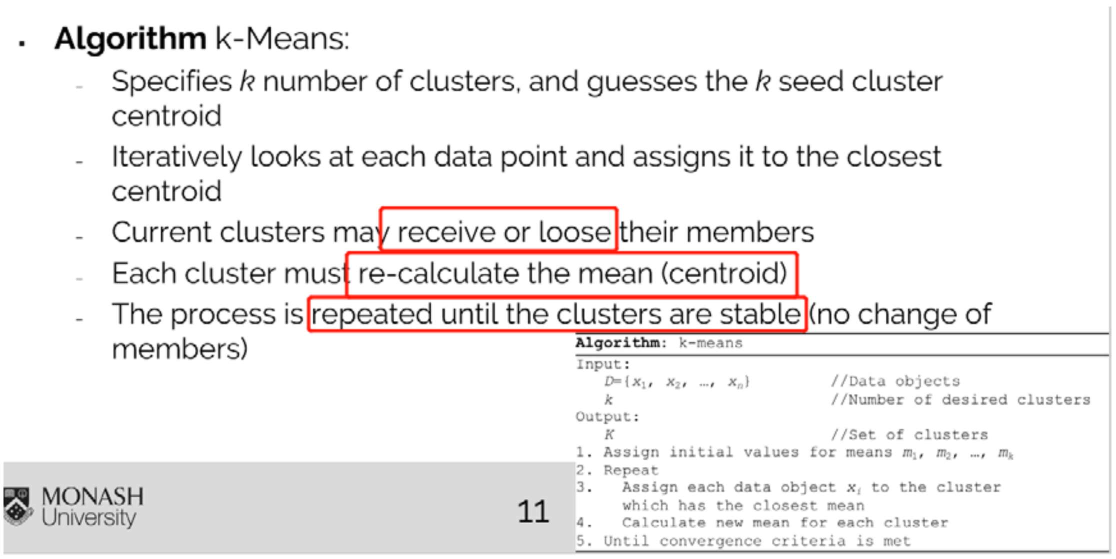
*Disadvantages*: The Random Forest model is difficult to interpret. It tends to return erratic predictions for observations out of range of training data

## Session 6

*Clustering:* A member is closer to another member within the same group than to a member of a different group

## K-Means

K-means is a partitional clustering algorithm
Each cluster has a cluster center, called centroid.



- **Algorithm** k-Means:
  - Specifies *k* number of clusters, and guesses the *k* seed cluster centroid
  - Iteratively looks at each data point and assigns it to the closest centroid
  - Current clusters may receive or loose their members
  - Each cluster must re-calculate the mean (centroid)
  - The process is repeated until the clusters are stable (no change of members)

```
Algorithm: k-means
Input:
    D={x_1, x_2, ..., x_n}        //Data objects
    k                             //Number of desired clusters
Output:
    K                             //Set of clusters
1. Assign initial values for means m_1, m_2, ..., m_k
2. Repeat
3.    Assign each data object x_i to the cluster
      which has the closest mean
4.    Calculate new mean for each cluster
5. Until convergence criteria is met
```

MONASH University 11

*Pros*: Simple and fast for low dimensional data (time complexity of K Means is linear i.e. O(n)), Scales to large data sets Easily adapts to new data points
*Cons*: It will not identify outliers, Restricted to data which has the notion of a centre (centroid)

*Data parallelism of k-means:* *它每一个process里面的数据是不会转移到另一个数据里面*
*Result Parallelism of k-means:* *他计算中心点的过程中，这些数据是可以相互转移的*

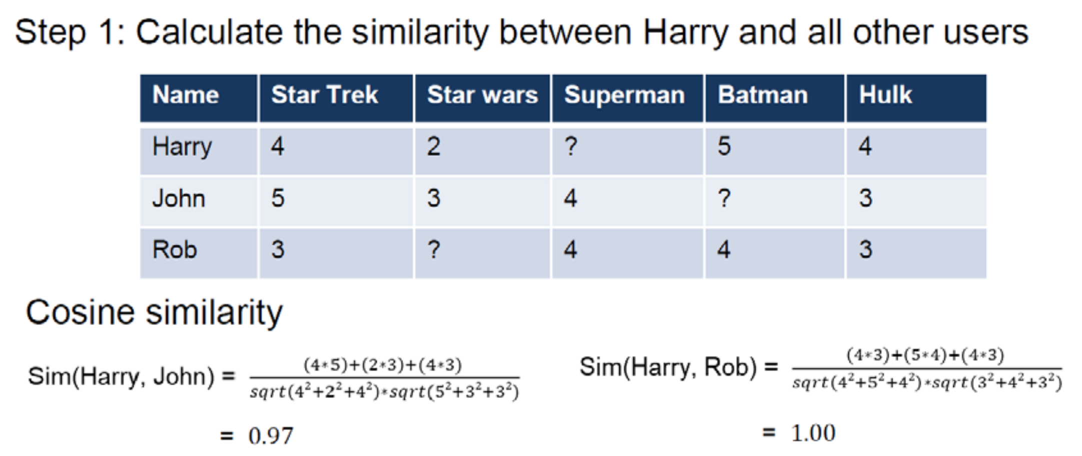
## Session 8

## Collaboration Filtering (Recommender System)

Collaborative filtering: is a mathematical method/formula to find the predictions about how much a user can rate a particular item by comparing that user to all other users.

Explicit(显式): users giving ratings to movies
Implicit(隐式): e.g. views, clicks, purchases, likes, shares etc.).

- **Calculate the similarity**



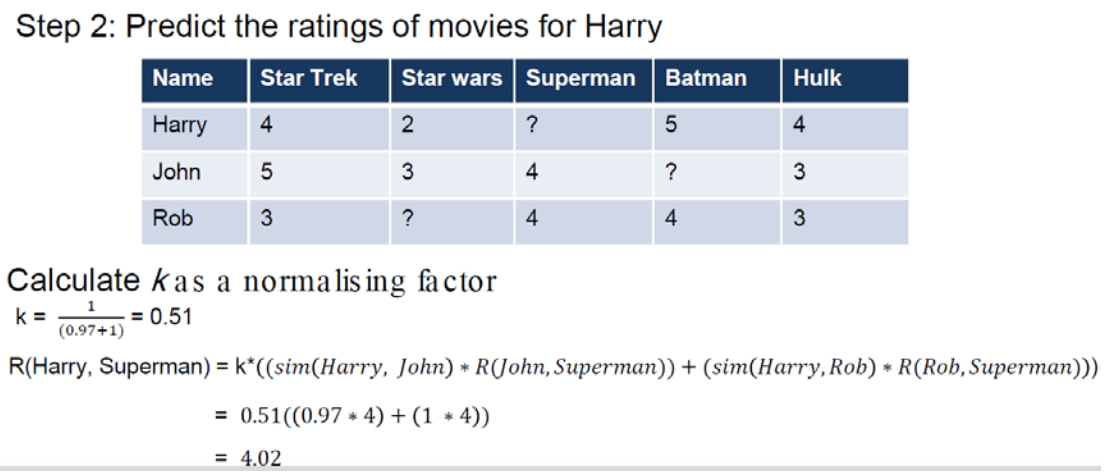
Step 1: Calculate the similarity between Harry and all other users

| Name | Star Trek | Star wars | Superman | Batman | Hulk |
|---|---|---|---|---|---|
| Harry | 4 | 2 | ? | 5 | 4 |
| John | 5 | 3 | 4 | ? | 3 |
| Rob | 3 | ? | 4 | 4 | 3 |

Cosine similarity

$$\text{Sim(Harry, John)} = \frac{(4*5)+(2*3)+(4*3)}{sqrt(4^2+2^2+4^2)*sqrt(5^2+3^2+3^2)} = 0.97$$

$$\text{Sim(Harry, Rob)} = \frac{(4*3)+(5*4)+(4*3)}{sqrt(4^2+5^2+4^2)*sqrt(3^2+4^2+3^2)} = 1.00$$



Step 2: Predict the ratings of movies for Harry

| Name | Star Trek | Star wars | Superman | Batman | Hulk |
|---|---|---|---|---|---|
| Harry | 4 | 2 | ? | 5 | 4 |
| John | 5 | 3 | 4 | ? | 3 |
| Rob | 3 | ? | 4 | 4 | 3 |

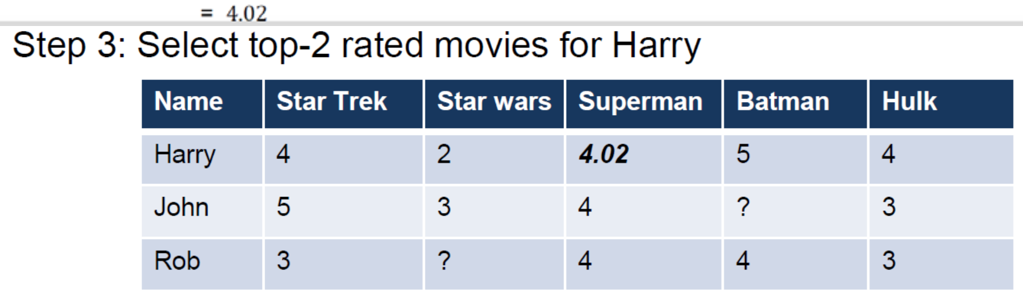
Calculate $k$ as a normalising factor

$$k = \frac{1}{(0.97+1)} = 0.51$$

$$\text{R(Harry, Superman)} = k^*((sim(Harry,\ John) * R(John, Superman)) + (sim(Harry, Rob) * R(Rob, Superman)))$$

$$= 0.51((0.97 * 4) + (1 * 4))$$

$$= 4.02$$



Step 3: Select top-2 rated movies for Harry

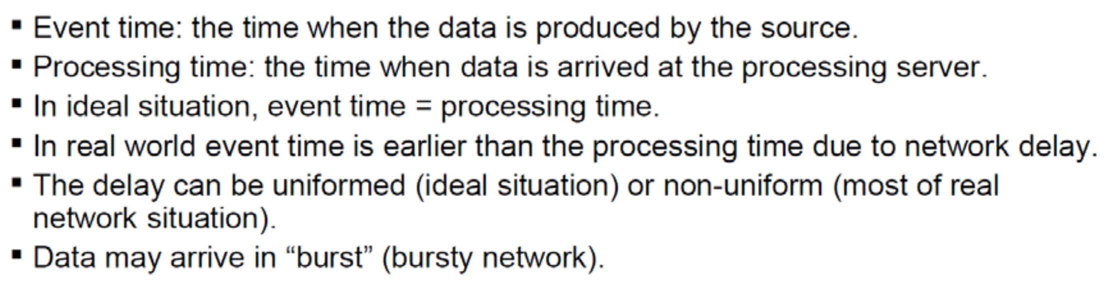
| Name | Star Trek | Star wars | Superman | Batman | Hulk |
|---|---|---|---|---|---|
| Harry | 4 | 2 | ***4.02*** | 5 | 4 |
| John | 5 | 3 | 4 | ? | 3 |
| Rob | 3 | ? | 4 | 4 | 3 |

*Top-2(Harry, movies)*= Batman, Superman

## Session 9

## Data stream

A data stream is a real-time, continuous, ordered (implicitly by arrival time or explicitly by timestamp) sequence of items.



- Event time: the time when the data is produced by the source.
- Processing time: the time when data is arrived at the processing server.
- In ideal situation, event time = processing time.
- In real world event time is earlier than the processing time due to network delay.
- The delay can be uniformed (ideal situation) or non-uniform (most of real network situation).
- Data may arrive in “burst” (bursty network).

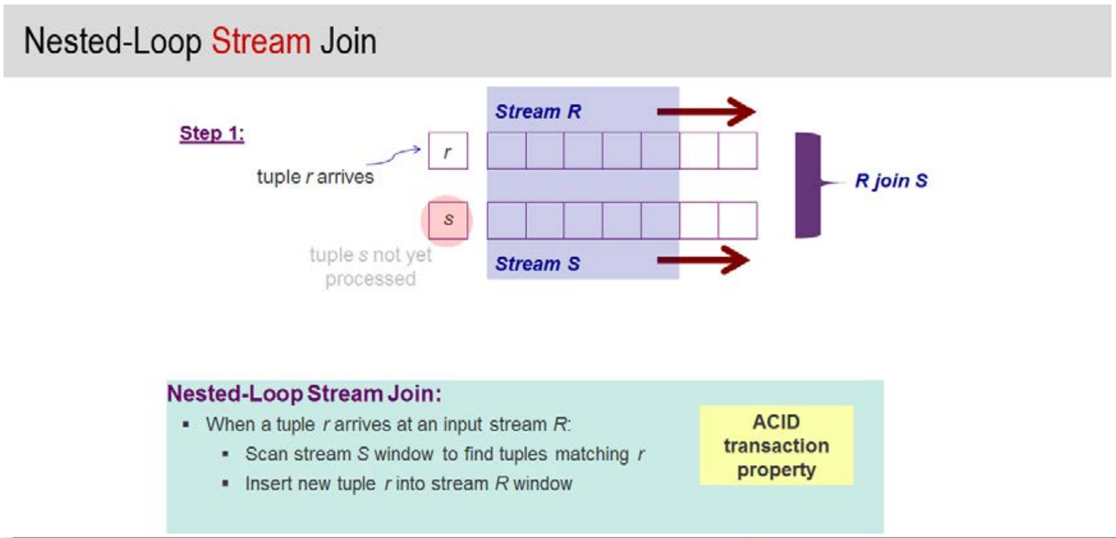
## Apache Kafka

publish-subscribe system. All messages are persisted and replicated to peer brokers for fault tolerance. Messages stay around for a configurable period of time

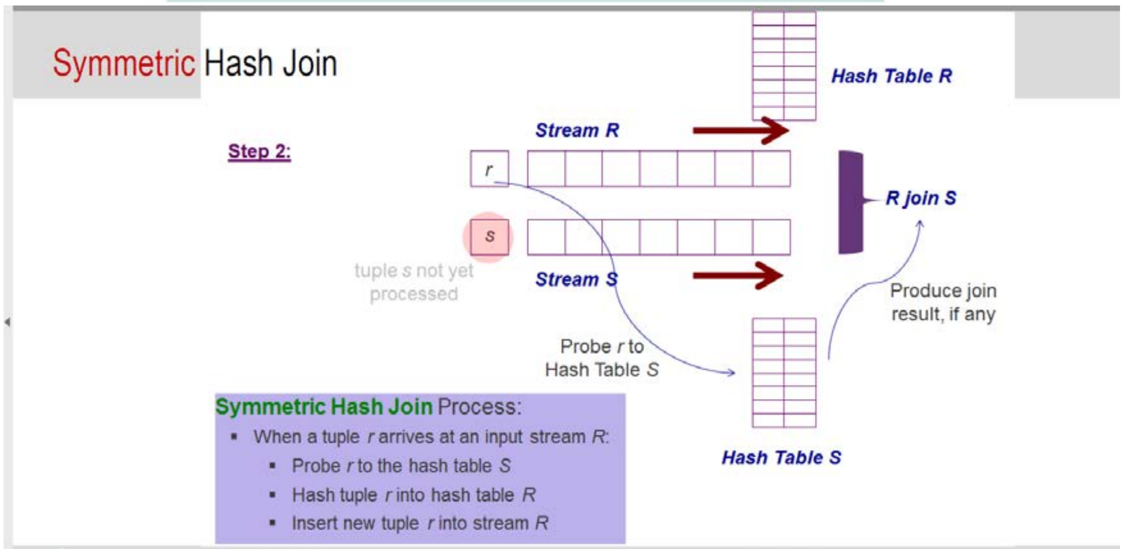
*Advantages:*
Handles large volume of data, reliable, fault tolerant and scalable system.
Handles high-velocity real-time data
Highly durable system as data is persistent.
Handles messages with very low latency

### Session 9

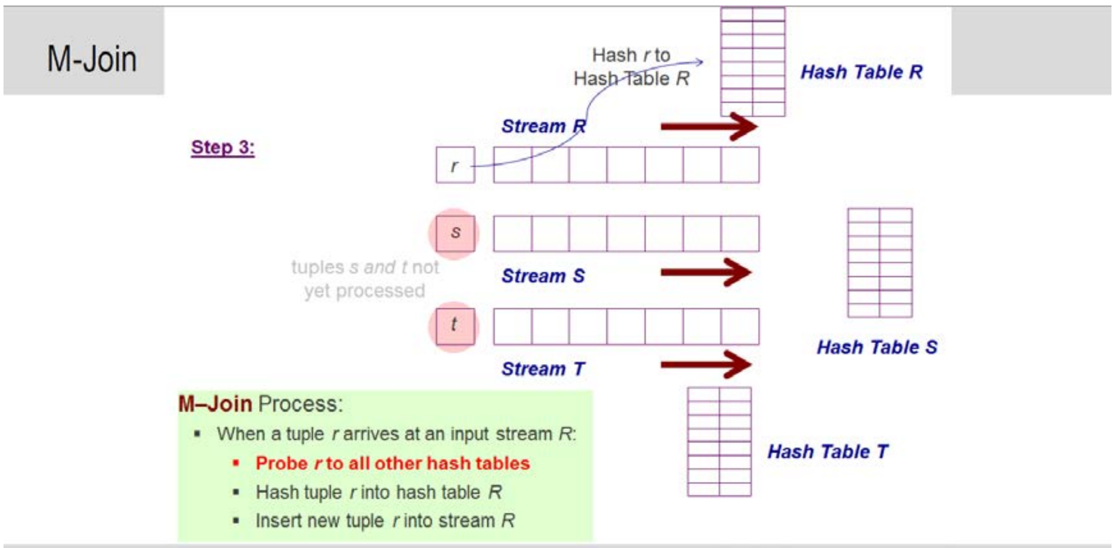
## Stream Join Processing



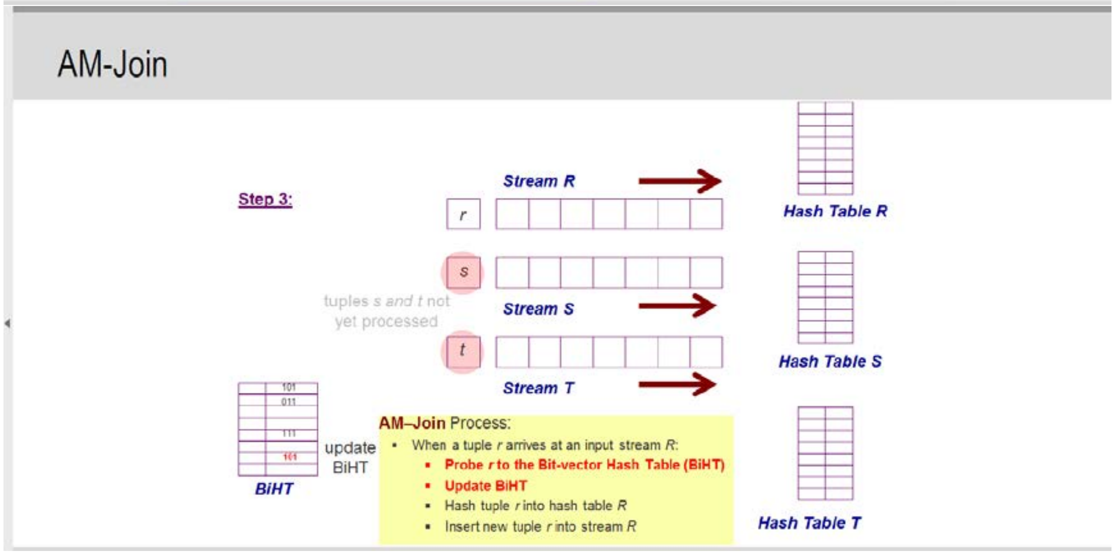
**Nested-Loop Stream Join:**
- When a tuple *r* arrives at an input stream *R*:
  - Scan stream *S* window to find tuples matching *r*
  - Insert new tuple *r* into stream *R* window



**Symmetric Hash Join** Process:
- When a tuple *r* arrives at an input stream *R*:
  - Probe *r* to the hash table *S*
  - Hash tuple *r* into hash table *R*
  - Insert new tuple *r* into stream *R*



M–Join Process:
- When a tuple r arrives at an input stream R:
  - Probe r to all other hash tables
  - Hash tuple r into hash table R
  - Insert new tuple r into stream R



AM–Join Process:
- When a tuple r arrives at an input stream R:
  - Probe r to the Bit-vector Hash Table (BiHT)
  - Update BiHT
  - Hash tuple r into hash table R
  - Insert new tuple r into stream R

## Handshake Join

1. 错开一个
2. 每次hand shake两次，它和它后面的

Session 11

## Granularity

Granularity is the level of detail at which data are stored in a database.
- level-0, the bottom level indicating no aggregation.
- level-1 and level-2 with more aggregation.

Granularity may simplify the complexity of the information.

*Granularity Reduction:* When the time slide is more than one unit of time, there will be a reduction in terms of number of records after aggregation.

## Sensor Arrays

A sensor array is a group of sensors, usually deployed in a certain geometry pattern.
A network of distributed sensors.

They add new dimension to the observation, and hence it helps to estimate more parameters, to have better picture of the environment being observed, and improve accuracy.

- Multiple sensors measuring the same things
  Reduce and then Merge (调换位置)
- Multiple sensors measuring different things
  Reduce, Normalize, and then Merge（Normalize, Merge and then Reduce）

Step 1: Normalize each stream according to the categories
Step 2: Merge the normalized streams.
Step 3: Reduce the granularity of the merged results

Granularity reduction of sensor data is achieved though the windowing schemes.
The drill-down of data streams can be assisted though multi levels of granularity which combined several granularity levels when presenting the data streams
Sensor arrays are multiple sensors that work together in an environment to provide users with more complete picture of the environment.