Parallel Databases

Dr. Geetha Mary A
Associate Professor ,
SCOPE, Vellore Institute of Technology, Vellore

Source:
Mc Graw Hill, Inc. 2017, RamaKrishnan and Gehrke, Database Management Systems, Third Edition

Parallel database

1. Introduction
2. Architecture for Parallel databases.
3. Parallel query Evaluation
4. Parallelizing Individual operations.

Introduction

- What is a Centralized Database ?

-all the data is maintained at a single site and assumed that the processing of individual transaction is essentially sequential.

PARALLEL DBMSs

WHY DO WE NEED THEM?

- More and More Data!

  We have databases that hold a high amount of data, in the order of 10^{12} bytes:

  10,000,000,000,000 bytes!

- Faster and Faster Access!

  We have data applications that need to process data at very high speeds:

  10,000s transactions per second!

SINGLE-PROCESSOR DBMS AREN'T UP TO THE JOB!

Why Parallel Access To Data?

At 10 MB/s
1.2 days to scan

1,000 x parallel
1.5 minute to scan.

1 Terabyte
10 MB/s
Bandwidth
1 Terabyte

Parallelism:
divide a big problem
into many smaller ones
to be solved in parallel.

Parallel DB

- Parallel database system seeks to improve performance through parallelization of various operations such as loading data ,building indexes, and evaluating queries by using multiple CPUs and Disks in Parallel.

- Motivation for Parallel DB
- Parallel machines are becoming quite common and affordable
  - Prices of microprocessors, memory and disks have dropped sharply
- Databases are growing increasingly large
  - large volumes of transaction data are collected and stored for later analysis.
  - multimedia objects like images are increasingly stored in databases

PARALLEL DBMSs

BENEFITS OF A PARALLEL DBMS

- ☺ Improves Response Time.

INTERQUERY PARALLELISM

It is possible to process a number of transactions in parallel with each other.

- ☺ Improves Throughput.

INTRAQUERY PARALLELISM

It is possible to process 'sub-tasks' of a transaction in parallel with each other.

PARALLEL DBMSs

HOW TO MEASURE THE BENEFITS

- ❖ Speed-Up
  - – Adding more resources results in proportionally less running time for a fixed amount of data.

  10 seconds to scan a DB of 10,000 records using 1 CPU
  1 second to scan a DB of 10,000 records using 10 CPUs

- ❖ Scale-Up
- ❖ If resources are increased in proportion to an increase in data/problem size, the overall time should remain constant

– 1 second to scan a DB of 1,000 records using 1 CPU
 1 second to scan a DB of 10,000 records using 10 CPUs

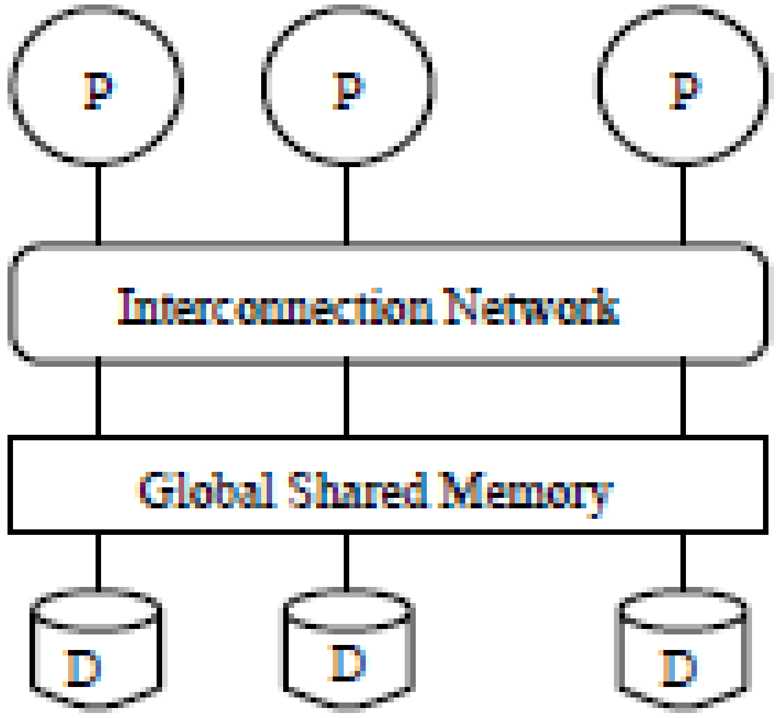
Architectures for Parallel Databases

- The basic idea behind Parallel DB is to carry out evaluation steps in parallel whenever is possible.

- There are many opportunities for parallelism in RDBMS.

- 3 main architectures have been proposed for building parallel DBMSs.
  1. **Shared Memory**
  2. **Shared Disk**
  3. **Shared Nothing**

Shared Memory



SHARED MEMORY

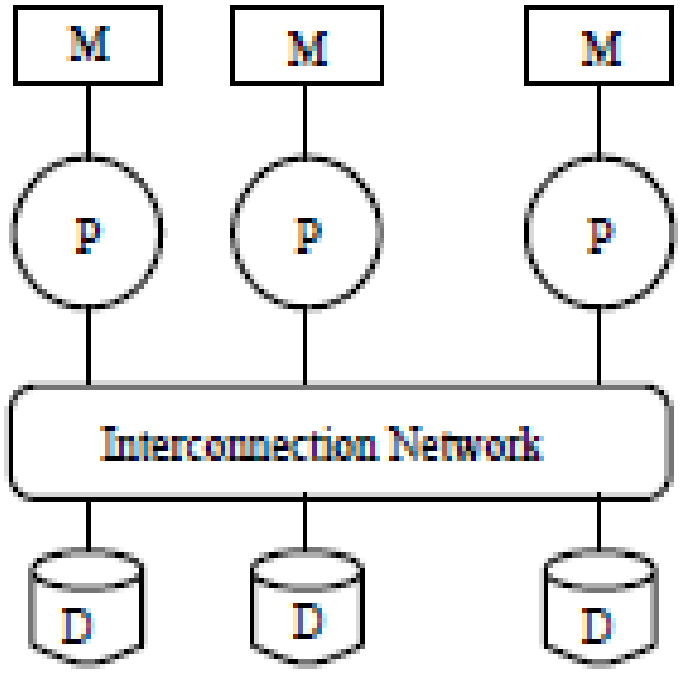
- **Advantages:**

1. It is closer to conventional machine.
2. OS services.
3. overhead is low leveraged to utilize the additional CPUs.

- **Disadvantage:**

1. It leads to bottleneck problem
2. Expensive to build
3. It is less sensitive to partitioning

Shared Disk



SHARED DISK

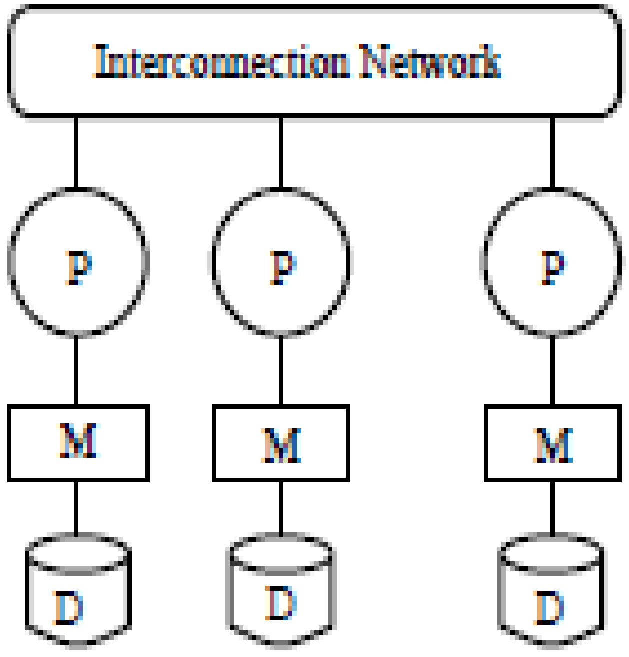
- **Advantages:**

1. Almost same

- **Disadvantages:**

1. More interference
2. Increases N/W band width
3. Shared disk less sensitive to partitioning

Shared Nothing



SHARED NOTHING

- **Advantages:**

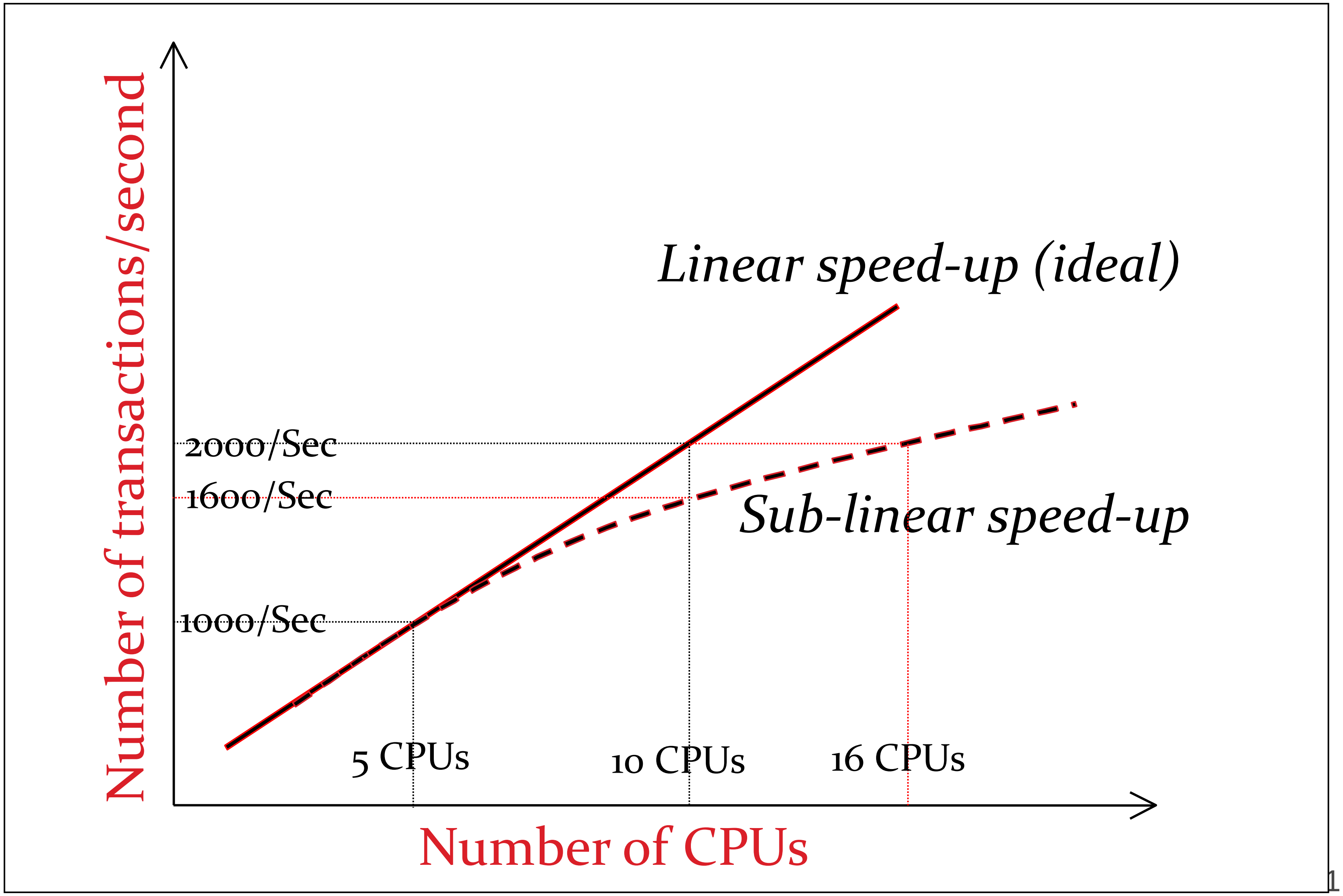
1. It provides linear scale up &linear speed up
2. Shared nothing benefits from "good" partitioning
3. Cheap to build

- **Disadvantage**

1. Hard to program
2. Addition of new nodes requires reorganizing

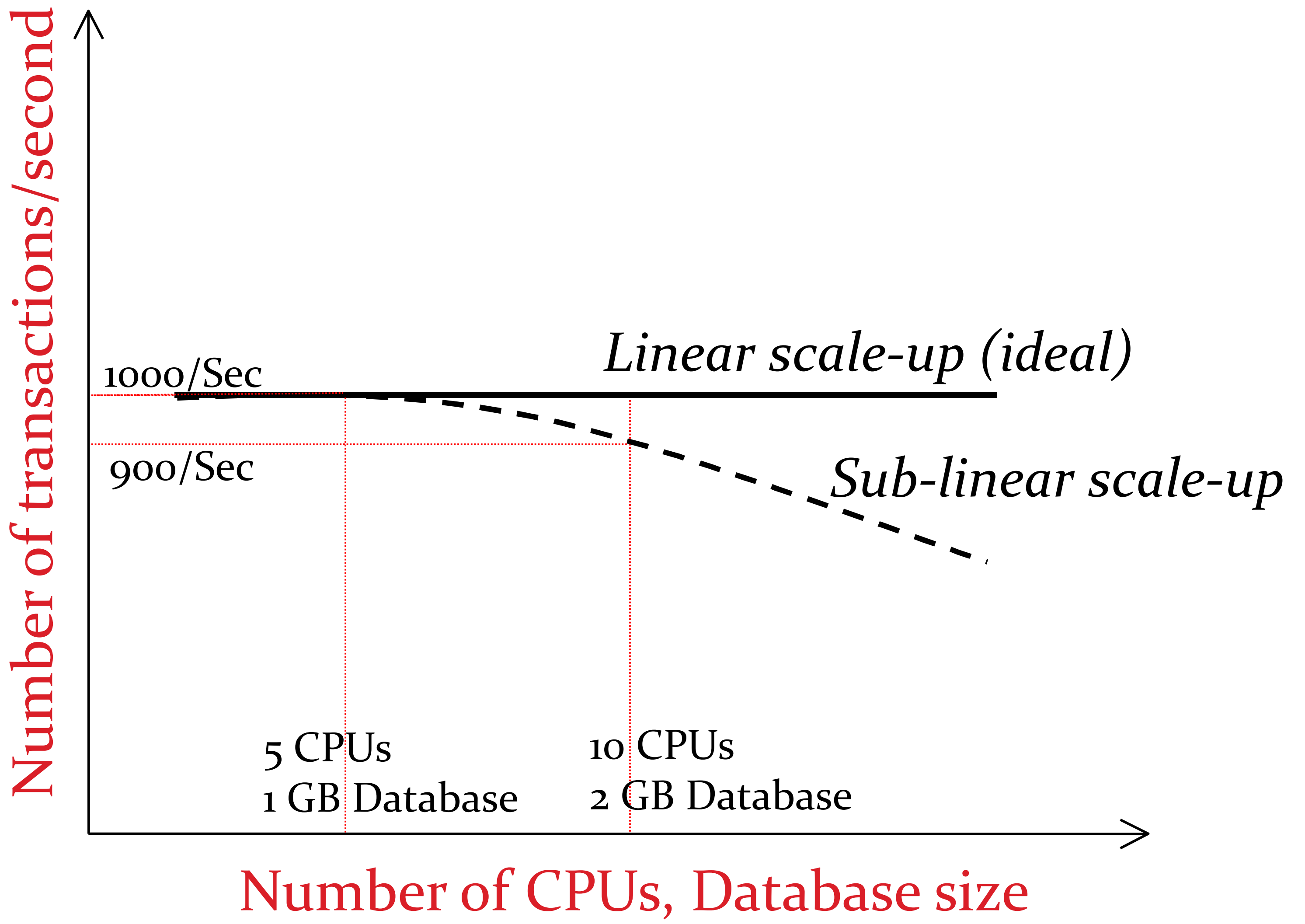
PARALLEL DBMSs

SPEED-UP



PARALLEL DBMSs

SCALE-UP

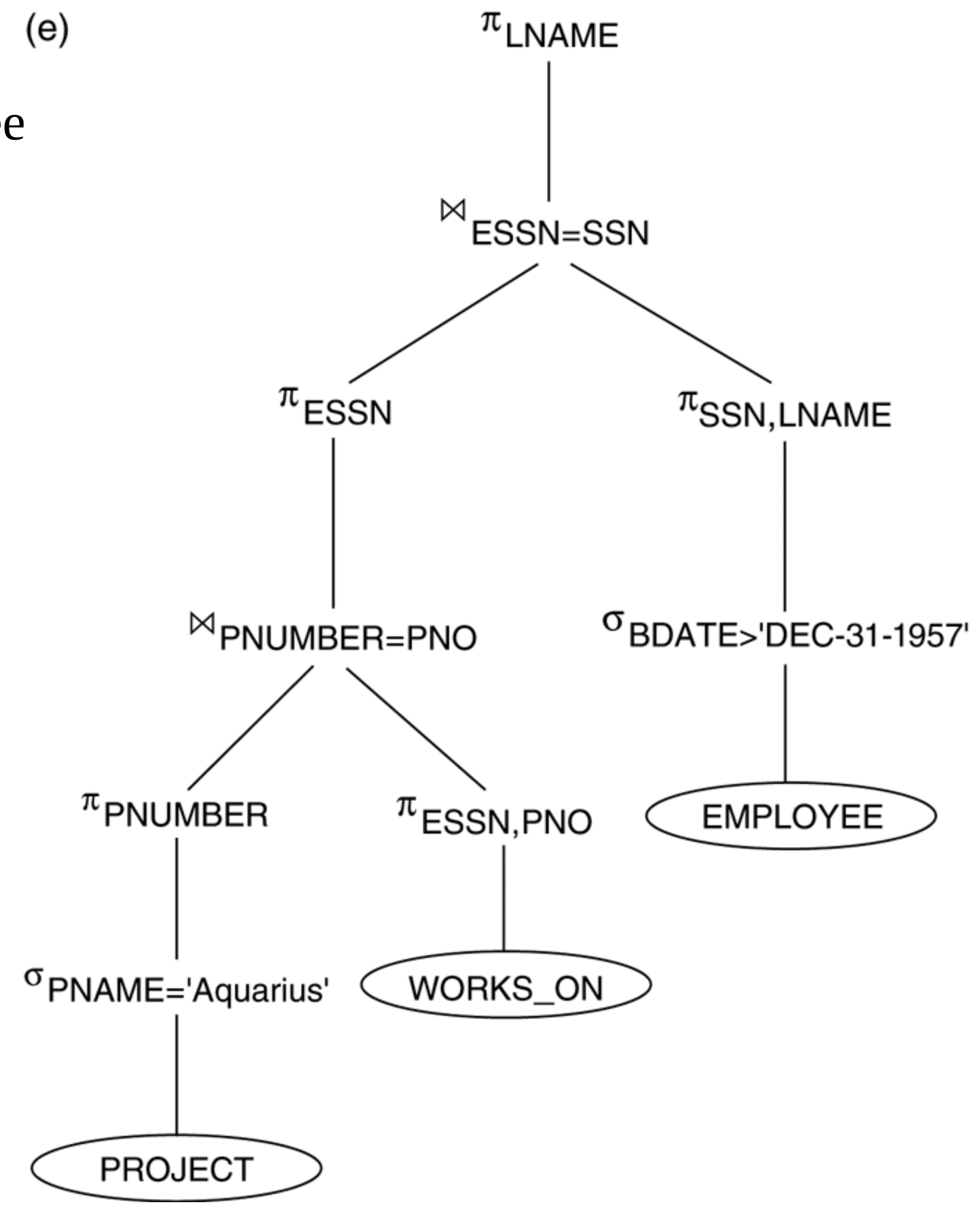


PARALLEL QUERY EVALUATION

A relational query execution plan is graph/tree of relational algebra operators (based on this operators can execute in parallel)

Sample Query Tree

(e)



Different Types of DBMS ||-ism

- Parallel evaluation of a relational query in DBMS With shared –nothing architecture
- Pipelined Parallelism (limited by sorting or aggregation)
- Data-partitioned parallel evaluation.

1. **Inter-query parallelism**
   - Multiple queries run on different sites
2. **Intra-query parallelism**
   - Parallel execution of single query run on different sites.

   a) **Intra-operator parallelism**
      a) get all machines working together to compute a given operation (scan, sort, join).

   b) **Inter-operator parallelism**
   - each operator may run concurrently on a different site (exploits pipelining).
   - In order to evaluate different operators in parallel, we need to evaluate each operator in query plan in Parallel.

Data Partitioning

- **Types of Partitioning**

1. **Horizontal Partitioning**: tuple of a relation are divided among many disks such that each tuple resides on one disk.
   - *It enables to exploit the I/O band width of disks by reading & writing them in parallel.*
   - *Reduce the time required to retrieve relations from disk by partitioning the relations on multiple disks.*
     1. **Range Partitioning**
     2. **Hash Partitioning**
     3. **Round Robin Partitioning**

2. **Vertical Partitioning**

1.Range Partitioning

- Tuples are sorted (conceptually), and n ranges are chosen for the sort key values so that each range contains roughly the same number of tuples;
- tuples in range i are assigned to processor i.
- Eg:
  - sailor _id 1-10 assigned to disk 1
  - sailor _id 10-20 assigned to disk 2
  - sailor _id 20-30 assigned to disk 3

- *range partitioning can lead to* ***data skew;*** *that is, partitions with widely varying number of tuples across*

2.Hash Partitioning

- **A hash function** is applied to selected fields of a tuple to determine its processor.

- Hash partitioning has the additional virtue that it keeps data evenly distributed even if the data grows and shrinks over time.

3.Round Robin Partitioning

- If there are *n processors, the i th tuple is assigned to processor* ***i mod n*** in **round-robin partitioning.**

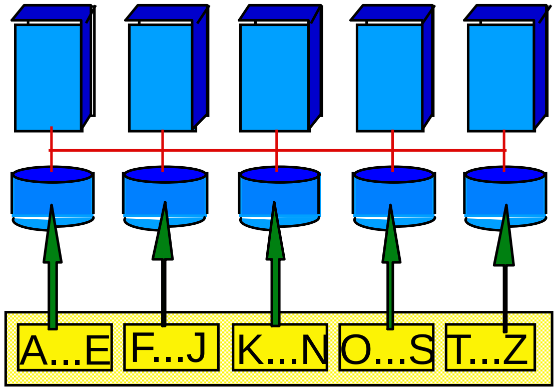
- Round-robin partitioning is suitable for efficiently evaluating queries that access the entire relation.

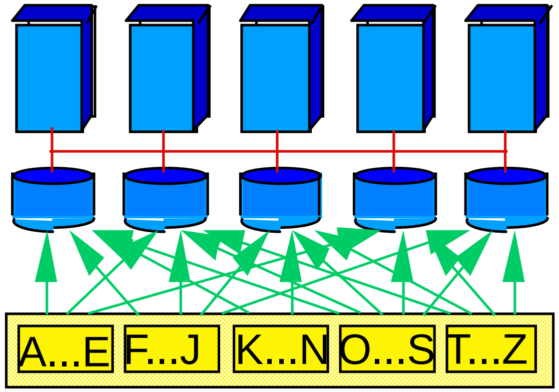
  - *If only a subset of the tuples (e.g., those that satisfy the selection condition age = 20) is required, hash partitioning and range partitioning are better than round-robin partitioning*

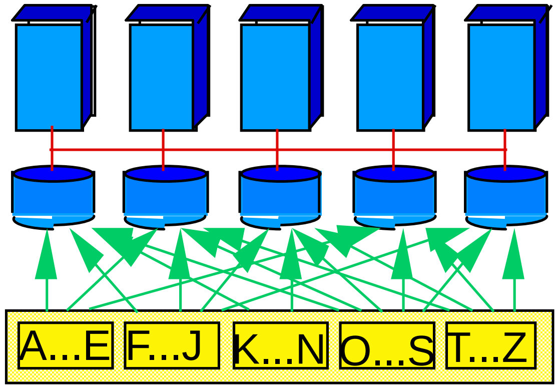
Range

Hash

Round Robin







Good for equijoins, exact-match queries, and range queries

Good for equijoins, exact match queries

Good to spread load

Parallelizing Sequential Operator Evaluation Code

1. An elegant software architecture for parallel DBMSs enables us to readily parallelize existing code for sequentially evaluating a relational operator.

2. The basic idea is to use parallel data streams.

3. Streams are merged as needed to provide the inputs for a relational operator.

4. The output of an operator is split as needed to parallelize subsequent processing.

5. A parallel evaluation plan consists of a dataflow network of ***relational, merge, and split operators.***

PARALLELIZING INDIVIDUAL OPERATIONS

- How various operations can be implemented in parallel in a shared-nothing architecture?

- Techniques
  1. Bulk loading& scanning
  2. Sorting
  3. Joins

1.Bulk Loading and scanning

- ***scanning a relation:*** *Pages* can be read in parallel while scanning a relation, and the retrieved tuples can then be merged, if the relation is partitioned across several disks.

- ***bulk loading*:** if a relation has associated indexes, any sorting of data entries required for building the indexes during bulk loading can also be done in parallel.

2.Parallel Sorting :

- **Parallel sorting steps:**
  1. First redistribute all tuples in the relation using range partitioning.
  2. Each processor then sorts the tuples assigned to it
  3. The entire sorted relation can be retrieved by visiting the processors in an order corresponding to the ranges assigned to them.

- Problem: ***Data skew***

- Solution: “sample” the data at the outset to determine good range partition points.

A particularly important application of parallel sorting is sorting the data entries in tree-structured indexes.

3.Parallel Join

1. The basic idea for joining A and B in parallel is to decompose the join into a collection of k smaller joins by using partition.

2. By using the same partitioning function for both A and B, we ensure that the union of the k smaller joins computes the join of A and B.

- **Hash-Join**
- **Sort-merge-join**

Sort-merge-join

- partition A and B by dividing the range of the join attribute into k disjoint subranges and placing A and B tuples into partitions according to the subrange to which their values belong.

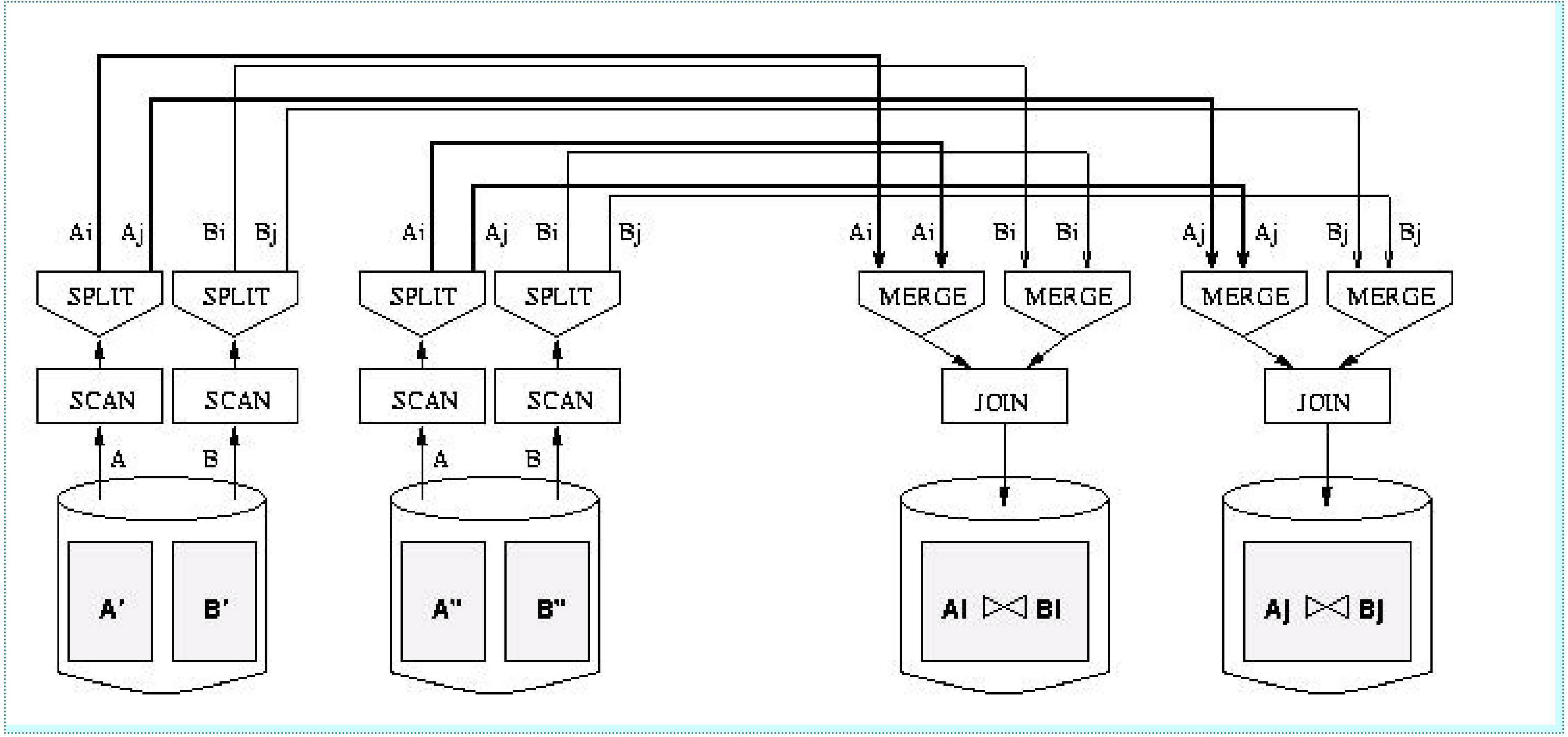
- Each processor carry out a local join.

- In this case the number of partitions k is chosen to be equal to the number of processors n .

- The result of the join of A and B, the output of the join process may be split into several data streams.

The advantage that the output is available in sorted order

Dataflow Network of Operators for Parallel Join



Good use of split/merge makes it easier to build parallel versions of sequential join code

Parallel Query Optimization

- Different operations in a query in parallel and execute multiple queries in parallel.
- Interoperation parallelism in a query:
  - The result of one operator can be pipelined in to another. (left deep plan)
  - Multiple independent operations can be executed concurrently.