Goals of Parallel Processing (1)

- To improve performance
  - Speedup
    - Shorten the execution time for the same size of problem
  - Scaleup
    - Increase the size of problem which can be treated during the same time period
      - (Increase the amount of data in databases)
- To improve reliability
  - Redundant configuration for fault tolerant
    - Data/Process Replication (cf. Data Distribution)
  - One of main issues of transaction processing

Goals of Parallel Processing (2)

- What are others?
  - Functionality?
    - Can be simulated (emulated) in a single processor
  - Easy to use? (or to maintenance)
    - in distributed environment
      - It is basically a matter of performance or reliability
      - The other aspects can also be simulated
  - ???

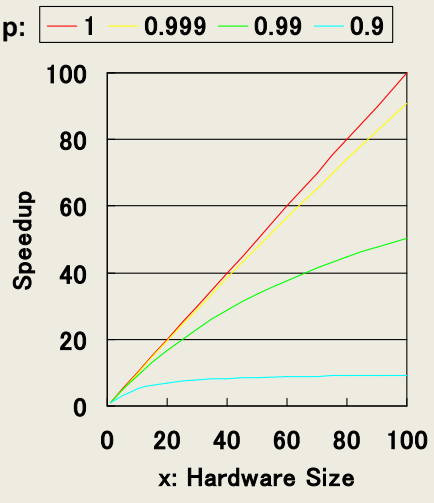
Speedup

- A parallel process includes portions which cannot be executed in parallel but sequential (eg. Initialization)
- We divide execution time into two parts: executed in parallel and sequential
- We denote the ratio of parallel by using p

$$Speedup = \frac{ExecTime_{seq}}{ExecTime_{para}} = \frac{part_{seq} + part_{para}}{part_{seq} + \frac{part_{para}}{x}} = \frac{1}{1 - p + \frac{p}{x}}$$

Linear Speedup

$$Speedup = \frac{ExecTime_{seq}}{ExecTime_{para}} = \frac{part_{seq} + part_{para}}{part_{seq} + \frac{part_{para}}{x}} = \frac{1}{1 - p + \frac{p}{x}}$$



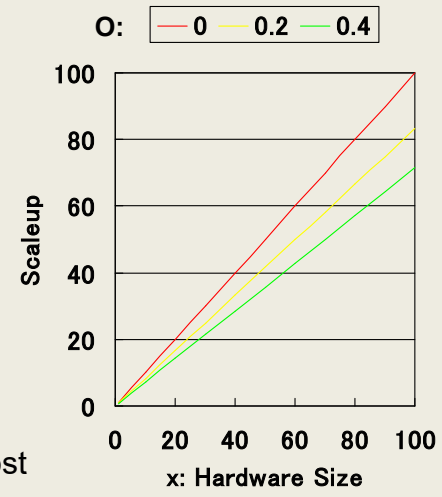
Scaleup

- When each processor executes completely independent process, x times number of processes are completed during a unit time
- The communication overhead decrease the number of completed processes, which is proportional to the size of system, but the scaleup still has linearity

Linear Scaleup

$$Scaleup = \frac{Throughput_{para}}{Throughput_{seq}} = \frac{x}{1+o}$$

x : Hardware Size
o : Parallelization Overhead [O(x)]
e.g. communication cost



Question (8-1)

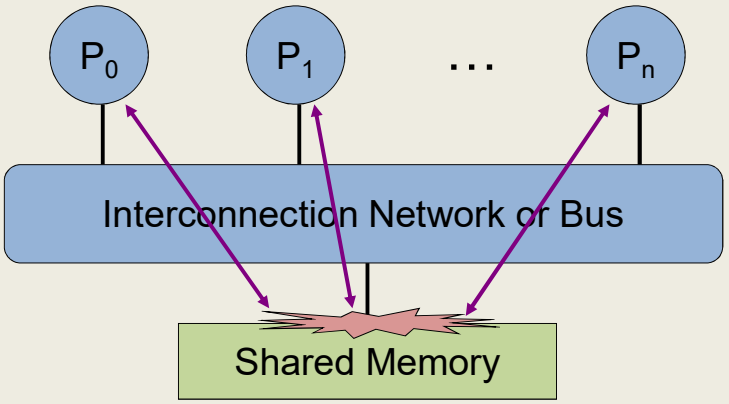
- In the case of parallel processing in data engineering, the realizations of the linear scaleup tends to be easier than those of the linear speedup. Consider the reasons for this matter.

Parallel Hardware Configuration (1)

- Memory Oriented Classification
  - Shared Memory Parallel (SMP) Model
    - Sequent Symmetry, etc
  - Distributed Memory Parallel (DMP) Model [inc. Distributed SM]
    - Thinking Machines CM5, nCUBE, etc
  - Non-Uniformly-Memory-Access (NUMA) Model
    - SGI Origin, NEC Cenju, etc.
  - Cache-Only-Memory-Architecture (COMA) Model
    - KSR

SMP Model

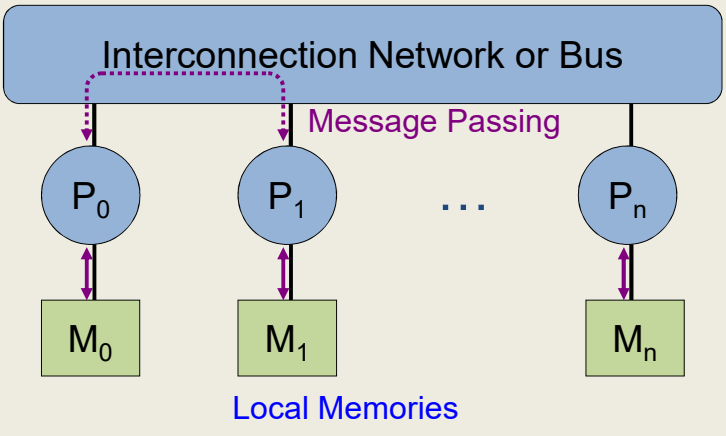
Shared Memory Parallel Model



- The shared memory becomes a bottleneck for the scalability

DMP Model

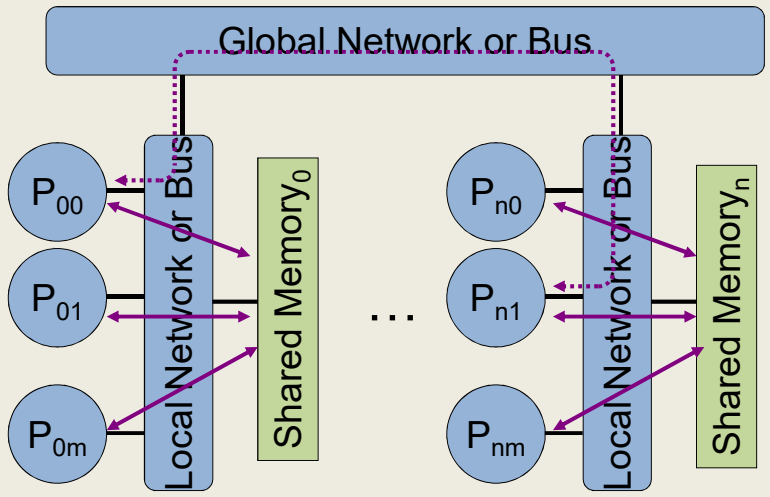
Distributed Memory Parallel Model



- This configuration has large scalability, but requires programming skills for messages

NUMA Model

Non-Uniformly-Memory-Access Model

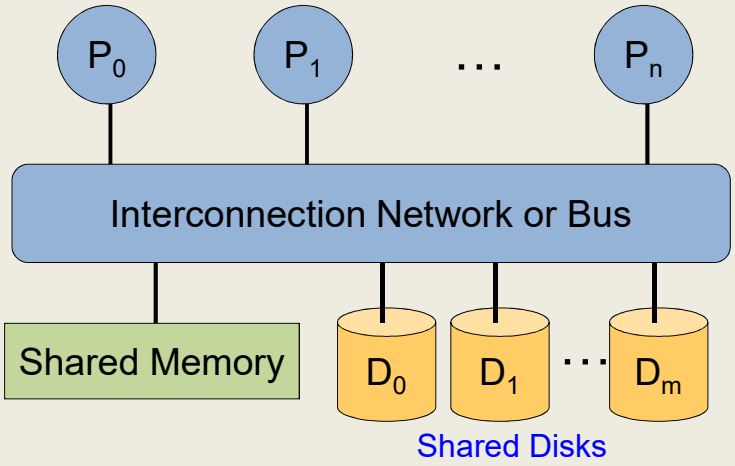


- A combination of SMP and DMP

Parallel Hardware Configuration (2)

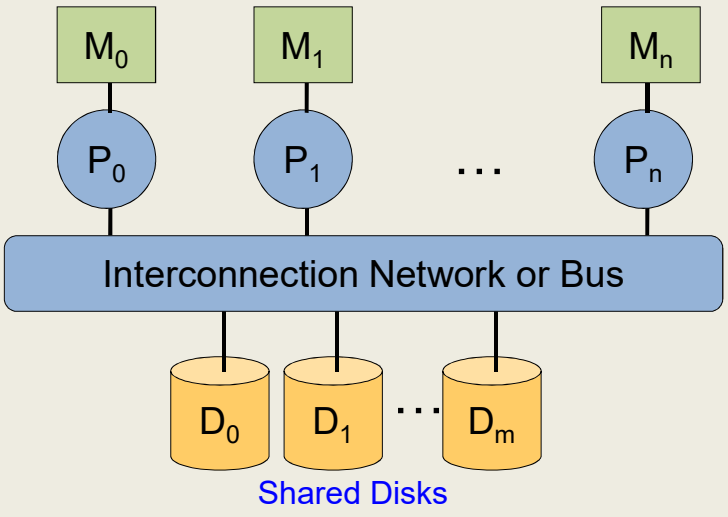
- Disk Oriented Classification
  - Shared Everything
    - All disks and memory modules are shared by the processors
    - For less than 64 disks
  - Shared Disks
    - Each processor can directory access any disk, but each processor has its own private memory
  - Shared Nothing
    - Each processor has its own private memory and dedicated disk drives
    - For more than 1024 disks

Shared Everything



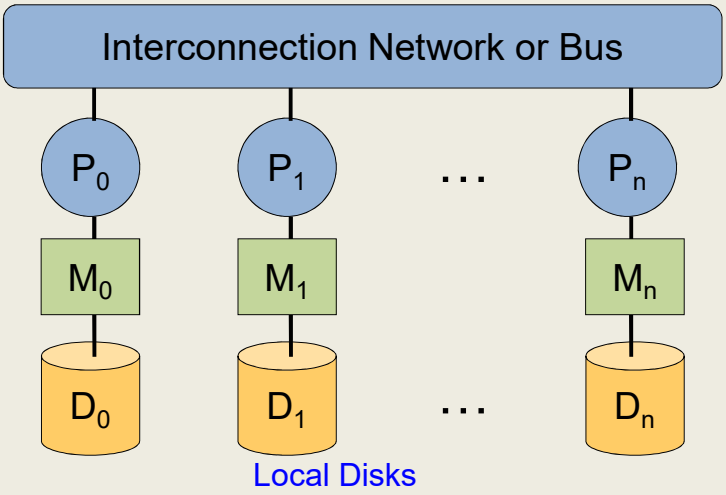
- The shared memory and shared disks becomes bottlenecks

Shared Disks



- The shared disks still be bottlenecks

Shared Nothing



- This configuration has large scalability

Communication Cost in Interconnection Network or Bus

- Focus on bandwidth requirement
- Notations:
  - Communication Cost for Memory Access: C_m
  - Communication Cost for Disk Access: C_d
  - Access Ratio for Local Memory: l_m
  - Access Ratio for Local Disks: l_d
- Bandwidth for each the configurations:
  - Shared Everything: $C_{SE} = C_m + C_d$
  - Shared Disks: $C_{SD} = (1- l_m) \times C_m + C_d$
  - Shared Nothing: $C_{SN} = (1- l_m) \times C_m + (1- l_d) \times C_d$

Comparison on Communication Cost

- When l_m, $l_d \approx 1$
  → $C_{SE} >> C_{SN}$
- Usually $C_m >> C_d$
  → $C_{SE} - C_{SD} > C_{SD} - C_{SN}$
- In DE Processing, C_d tend to be large
  → $C_{SD} >> C_{SN}$

Estimation (1)

- Suppose:
  - The number of processors = 100
  - Memory access unit = 4B/word
  - Memory access frequency = 100,000 word/s
  - Disk page size = 8KB/page
  - Disk access frequency = 1 page/s
  - Access ratio for local memory l_m = 95%
  - Access ratio for local disk l_d = 90%
- Calculate C_{SE} , C_{SD} , C_{SN}
- Consider a case of the disk access frequency becomes 10 page/s

Estimation (2)

- Memory access costs C_m :
  Memory access unit x Memory access frequency x processor #
  = 4B/word x 100,000 word/s x 100 = 40MB/s
- Disk access costs C_d :
  Disk page size x Disk access frequency x processor#
  = 8KB/page x 1 page/s x 100 = 0.8MB/s
- C_{SE} , C_{SD} , C_{SN}
  - $C_{SE} = C_m + C_d$ =40 + 0.8 = 40.8MB/s
  - $C_{SD} = (1- I_m) \times C_m + C_d$ = 0.05x40 + 0.8 = 2.8MB/s
  - $C_{SN} = (1- I_m) \times C_m + (1- I_d) \times C_d$ = 0.05 x 40 + 0.1 x 0.8 = 2.08 MB/s

large diff

small diff

- When the disk access frequency = 10 page/s
  - $C_{SE} = C_m + C_d$ =40 + 8 = 48MB/s
  - $C_{SD} = (1- I_m) \times C_m + C_d$ = 0.05x40 + 8 = 10MB/s
  - $C_{SN} = (1- I_m) \times C_m + (1- I_d) \times C_d$ = 0.05 x 40 + 0.1 x 8 = 2.8 MB/s

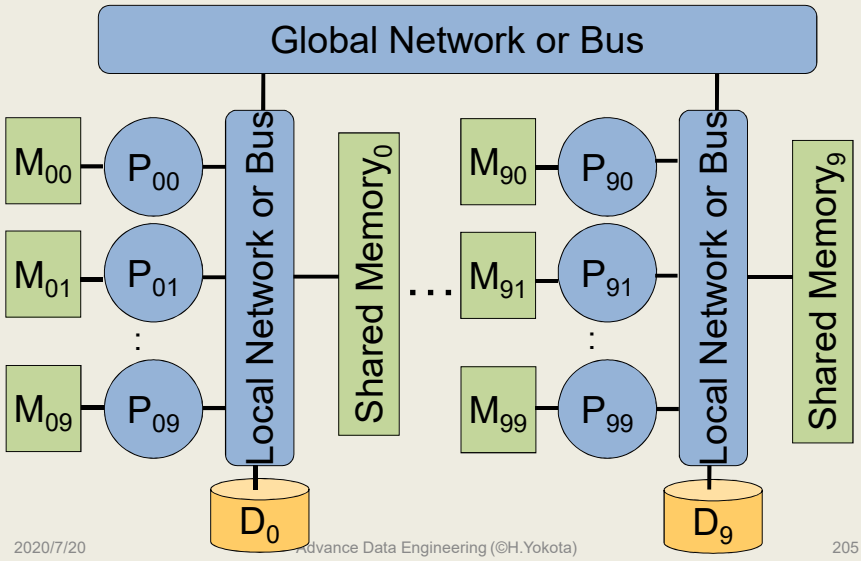
large diff

Question (8-2)

- Estimate the cost of NUMA with Disk configuration C_{NM} for the Global and Local Networks with the following assumptions (c.f. next page figure) :
  - The number of processors in a cluster = 10
  - The number of clusters = 10
  - A disk is connected to each cluster
  - Memory access unit = 4B/word
  - Memory access frequency = 100,000 word/s
  - Disk page size = 8KB/page
  - Disk access frequency = 1 page/s and 10 page/s
  - Access ratio for memories in local CPU I_{m1} = 90%
  - Access ratio for memories in local cluster I_{m2} = 5%
  - Access ratio for disks in local cluster I_d = 90%

NUMA Model with Disk

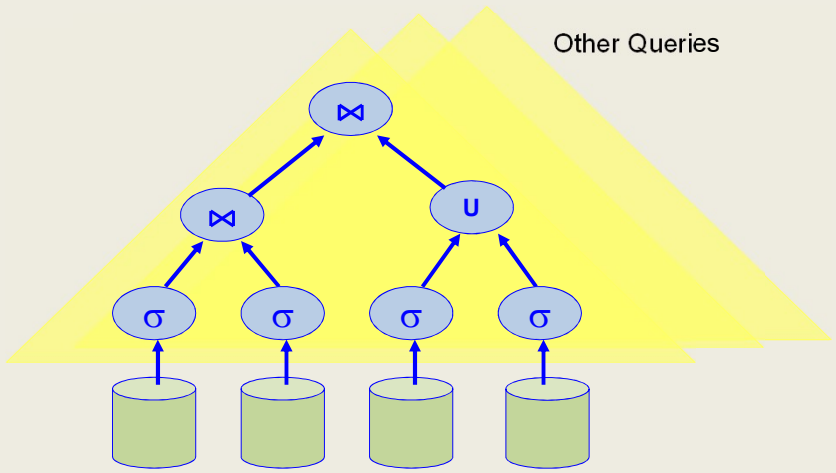
Non-Uniformly-Memory-Access Model



Layers of Parallel DB Operations

- Inter-Transaction (Inter-Query) Parallelism
  - Executing Multiple Transaction Simultaneously with Concurrency Control
- Intra-Transaction (Intra-Query) Parallelism
  - Inter-Operation Parallelism
    - Simultaneous Execution of nodes in a Query Tree
    - Pipelined Parallelism (Following Data Flow)
  - Intra-Operation Parallelism
    - Partitioned Parallelism
    - Parallel Algorithm for the operation

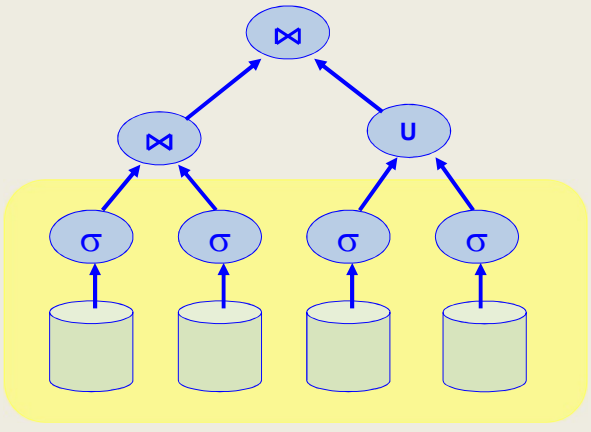
Inter-Query Parallelism



- Executing multiple queries (transactions)

Intra-Query Parallelism

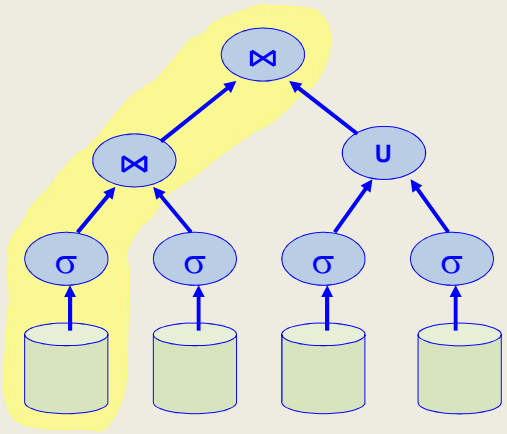
Inter-Operation Parallelism: Simultaneous Execution



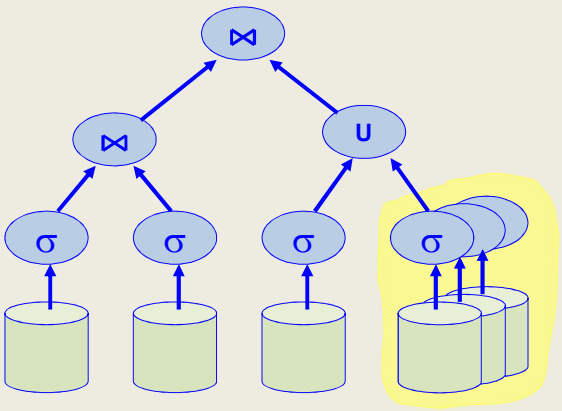
- Parallel execution of multiple operations in a tree

Intra-Query Parallelism

Inter-Operation Parallelism: Pipeline



Intra-Operation Parallelism

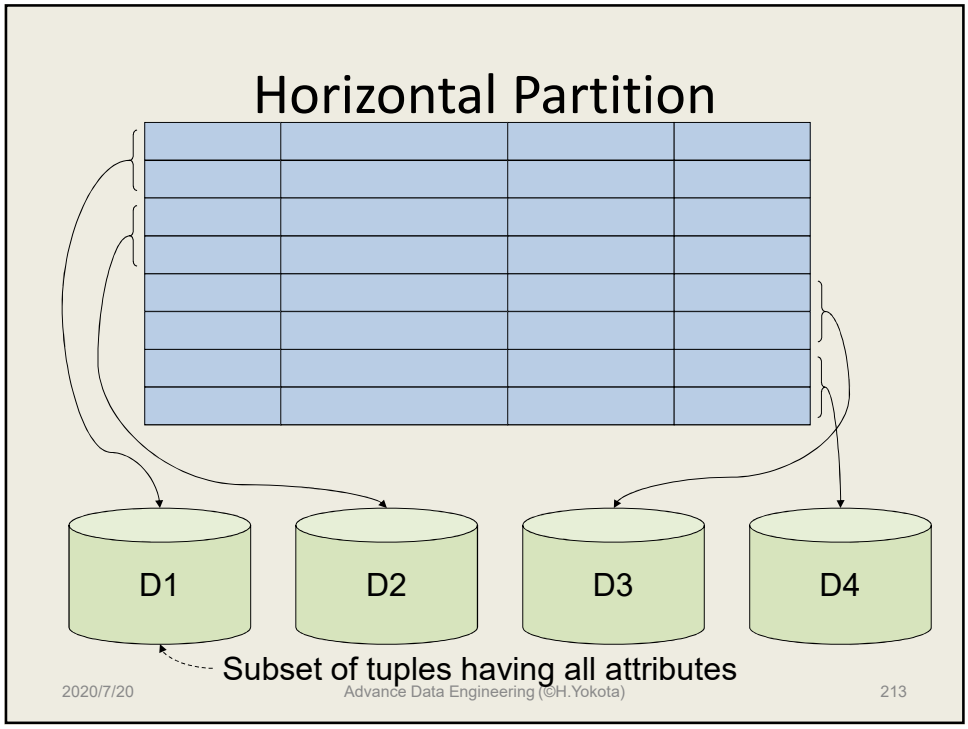


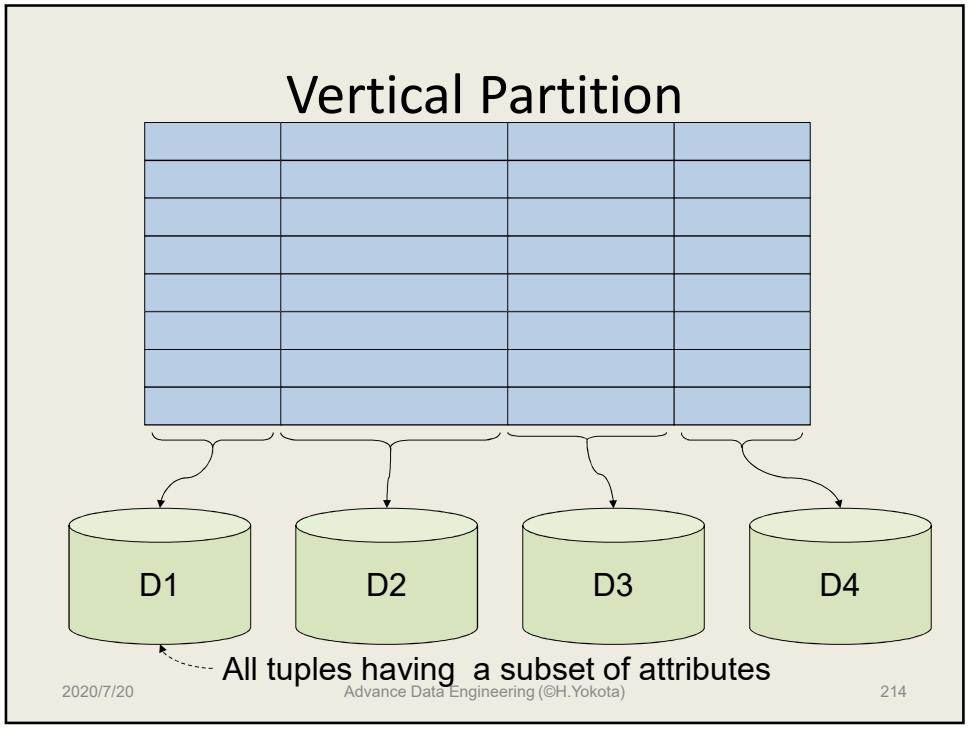
Parallel Relational Operations

- Intra Operation Parallelism
- Parallelizing Relational Operations
  - Selection / Projection Operations
    - Parallel Scan
  - Join / Aggregate Functions (with Group-By) Operations
    - Dedicated Parallel Algorithms
- The Way of Parallel Scan
  - Depend on Tuple Placement

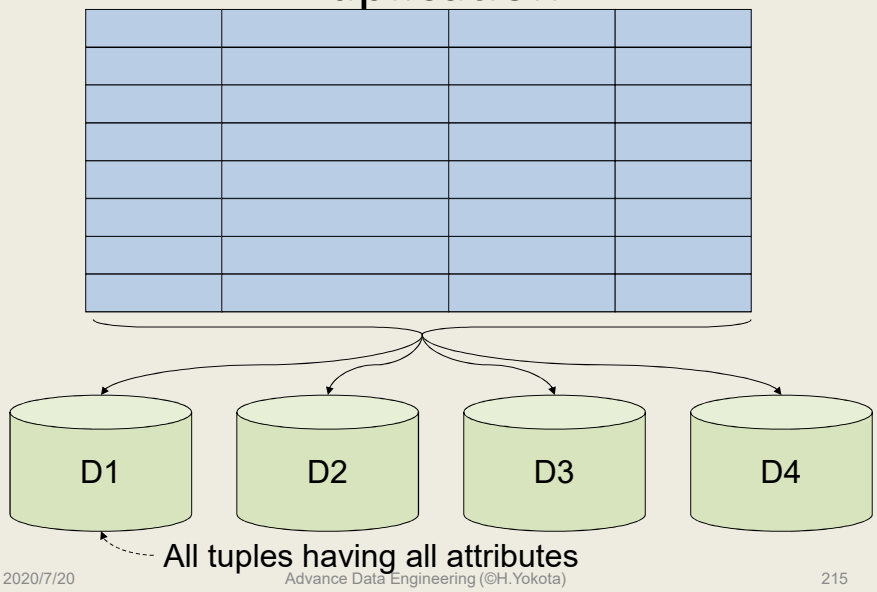
Tuple Placement

- Horizontal Partition (Fragmentation)
  - Each Processing Element has a subset of tuples
    - Round-Robin Partitioning
    - Hash Partitioning
    - Value Range Partitioning
- Vertical Partition (Fragmentation)
  - Each Processing Element has a subset of attributes of all tuples
    - Called as Transposed Files
- Duplication
  - Each Processing Element has all whole tuples


Horizontal Partition
D1
D2
D3
D4
Subset of tuples having all attributes
2020/7/20
Advance Data Engineering (©H.Yokota)
213


Vertical Partition
D1
D2
D3
D4
All tuples having a subset of attributes
2020/7/20
Advance Data Engineering (©H.Yokota)
214

Duplication



Parallel Scan for Duplicate Placement

- Query Decomposition
  - Original Query

    SELECT Product-Number, Price

    FROM Product WHERE Price > 170,000
  - Decomposed Query

    SELECT Product-Number, Price

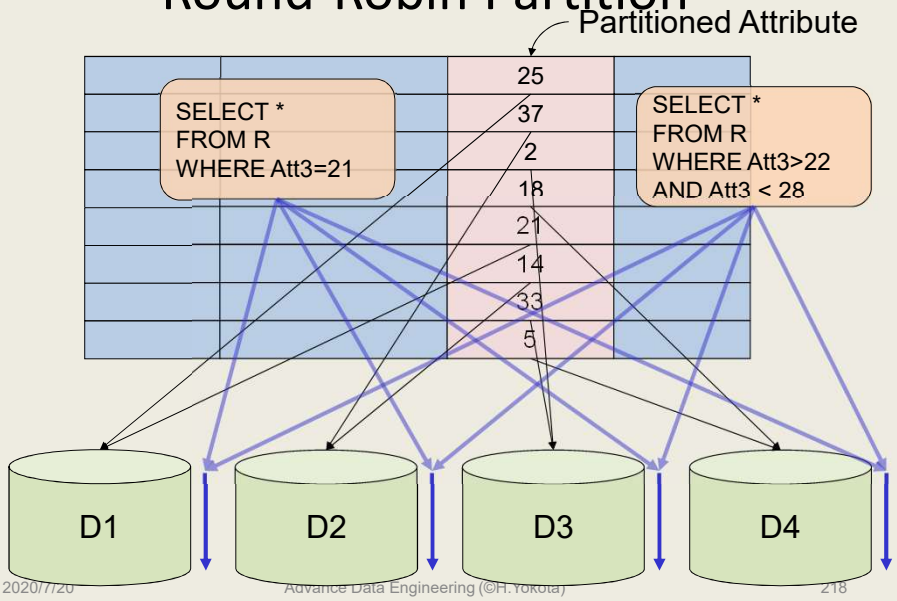
    FROM Product WHERE Price > 170,000

    AND Tuple-ID >= (i-1)*D AND Tuple-ID < i*D
  - D = {*R*}/n
  - Usually, Tuple-ID cannot be treated by user

Query Decomposition

- Assignment for one processing element
  - Comparison: {*R*}/n,
  - I/O: |*R*|/n
- No Inter-Query Parallelism
- Disk Utilization (Total)
  - Need n times Disk Space
- Update Cost
  - Update All Copies (Need Synchronization)
- Query Decomposition can also be used for Shared Disk

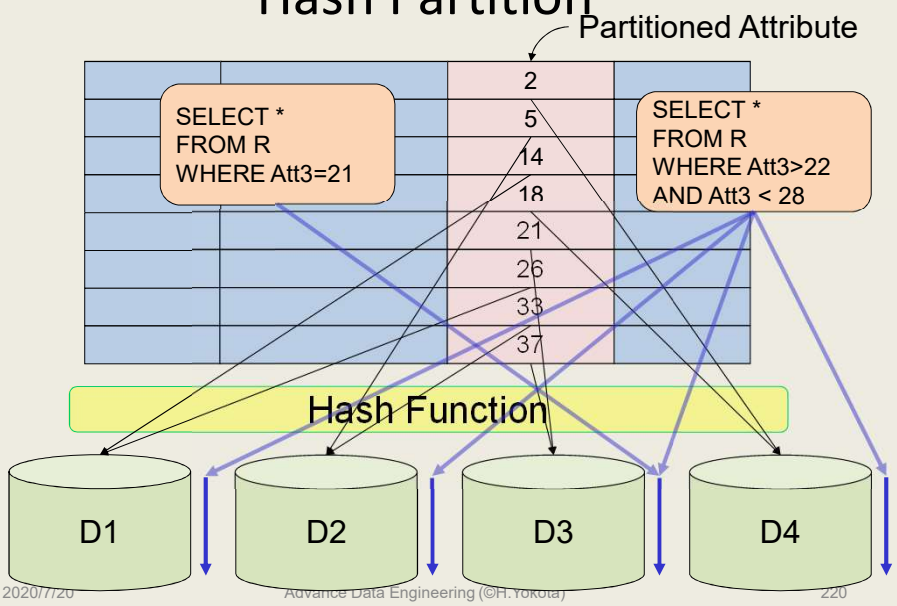
Round-Robin Partition



Round-Robin Partitioning

- Advantages
  - No skew in data distribution
    - Ideally, at most one object difference
- Disadvantages
  - Cannot determine disk storing target data
    - Every disk has to participate for each retrieval operation

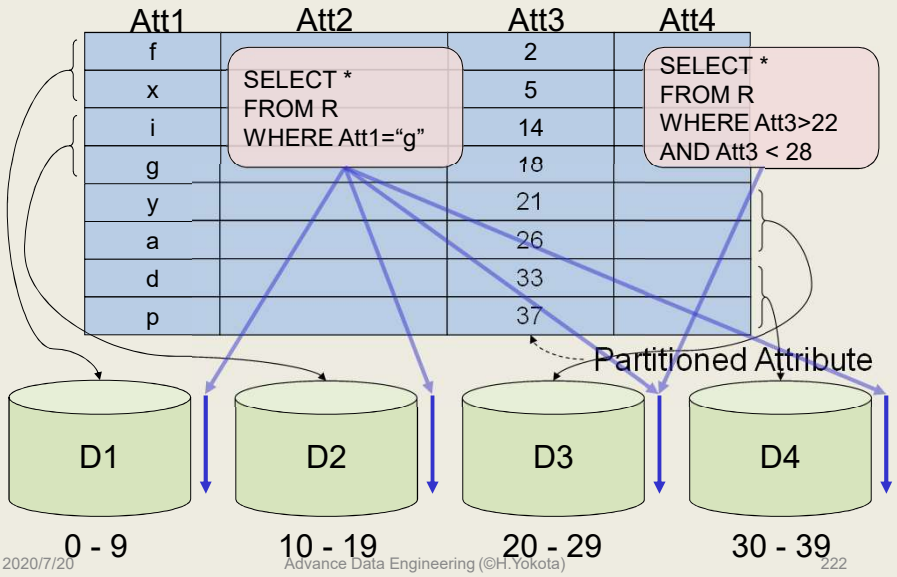
Hash Partition



Hash Partitioning

- Advantages
  - Rather small skew in data distribution
  - Can determine disk for strict match queries
- Disadvantages
  - Cannot treat range queries
  - Cannot treat I/O clustering

Value Range Partition



Value Range Partitioning

- Advantages
  - Can determine which disk should contain object data
  - Can treat range queries
  - Can cluster I/O operations for near values
- Disadvantages
  - Potentially skew the distribution of data
    - Even if the initial data allocation has no skew
    - Fixed criteria for partitioning cause skew

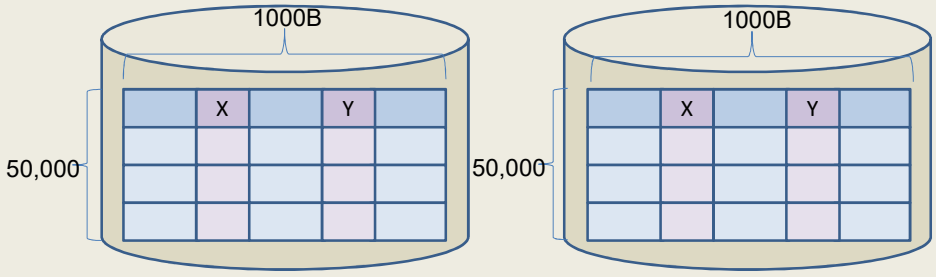
Question (8-3)

- Estimate costs (rough execution time) for Horizontal Partition
- Estimate costs (rough execution time) for Vertical Partition

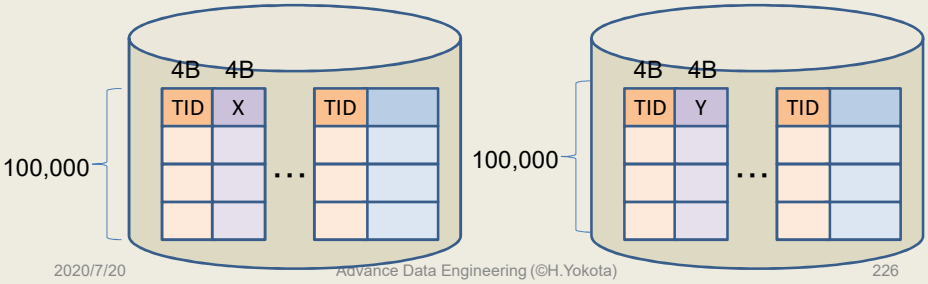
Assumptions for Question (8-2)

- Disk transfer bandwidth: 10MB/s
- Cardinality of a relation R: 100,000
- Total length of a tuple: 1,000B
- Search condition:
  a) Attribute X = 123 (4B integer)
  b) Attribute Y = 456 (4B integer)
- The number of tuples satisfying condition a: 1,000
- The number of tuples satisfying condition b: 500
- The number of tuples satisfying both condition: 100
- The size for TID: 4B
- Attributes included in output: X and Y
- The number of processors: 2
- Network bandwidth: 10MB/s

Horizontal Partition



Vertical Partition

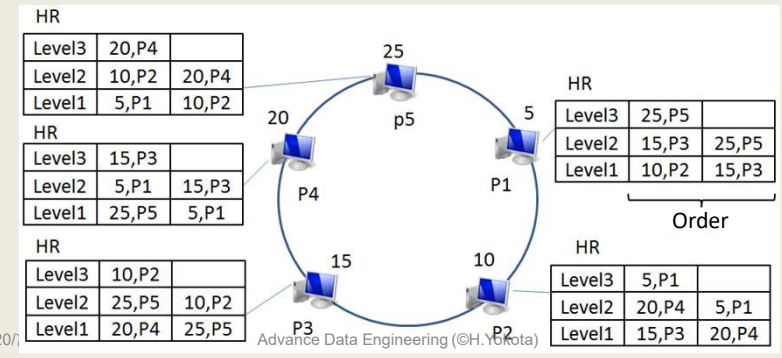


Index for Parallel Databases

- Many indexing methods have been proposed
  - At first, hash based approach cannot handle range queries
  - Some distributed hash table based methods capable of handling range queries have been proposed recently
- Distributed Hash Table(DHT) based methods
  - P-Tree, P-Ring
- Skip List based methods
  - Skip Graph
- B-tree based methods
  - Fat-Btree, a+Btree, RP*, …

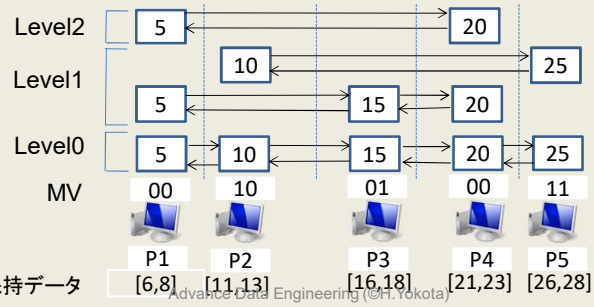
P-Ring [Machanavajihala et al. 2007]

- Based on Hash Table
  - Managing the key space distributed on nodes in a ring
- Each node has a table so called Hierarchical Ring(HR)
  - It keeps information of ***O*** neighbors in O^{L-1} distance (***O***: order, ***L***: Leverl)
  - In the under example, order is 2, Level 1 has $2^{1-1} = 1$ distance, and Lever 2 has $2^{2-1} = 2$ distance



SkipGraph [Aspnes and Shah, 2007]

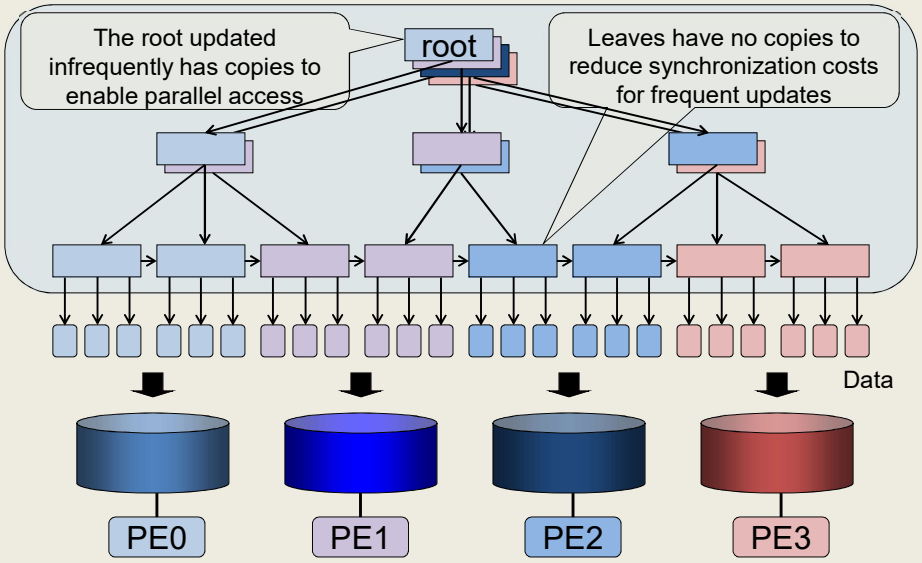
- Based on layered lists
- Each node has random bit sequence called Membership Vector（MV）
- The n-th bit from MSB in MV corresponds to link connections in LEVEL n.
  - Ex: since P1(MV : 00) and P3(MV : 01) have 1 in MSB, both of them have pointers in LEVEL 1
  - Neighboring nodes are linked in LEVEL 0



Fat-Btree

- A parallel B-tree structure we proposed in ICDE'99
- Each PE has a subtree of the whole B-tree
  - The leaf pages of the B+-tree are distributed among PEs
  - The root node and intermediate index nodes between the root node and leaf nodes allocated to the PE are contained
- The leaf pages are not duplicated
  - The leaf pages have a high update frequency
  - The nodes with a higher update frequency need a lower cost of update
- The root page and the index pages are only required for locating the leaf pages stored in each PE
  - Any node can accept access requests for data stored in any node (highly parallel access)
- Fat-Btree provides
  broad bandwidth access with low update overhead

An Example of the Fat-Btree



Advantages of Fat-Btree (vs. DHT)

- Low costs for accessing the next data
  - In many cases, continuous data are stored in very close location in a storage device by Fat-Btree
- Efficient node determination for queries
  - Nodes concerned in the queries are determined in early stage by traversing upper layer nodes of the Fat-Btree.
    - It is especially effective for range queries
- Efficient load balance
  - Data migration for balancing load can be done locally in Fat-Btree and transparent to clients

Parallel Scan for Horizontal Partition (1)

- Value Range or Hash Partitioning
  - For the Partitioned Attribute
    - No Intra-Operation Parallelism
    - Inter-Operation Parallelism (Inter-Query)
  - For the Other Attributes
    - No Inter-Operation Parallelism
    - Intra-Operation Parallelism
    - Assignment for one processing element
      - Comparison: {R}/n
      - I/O: |R|/n

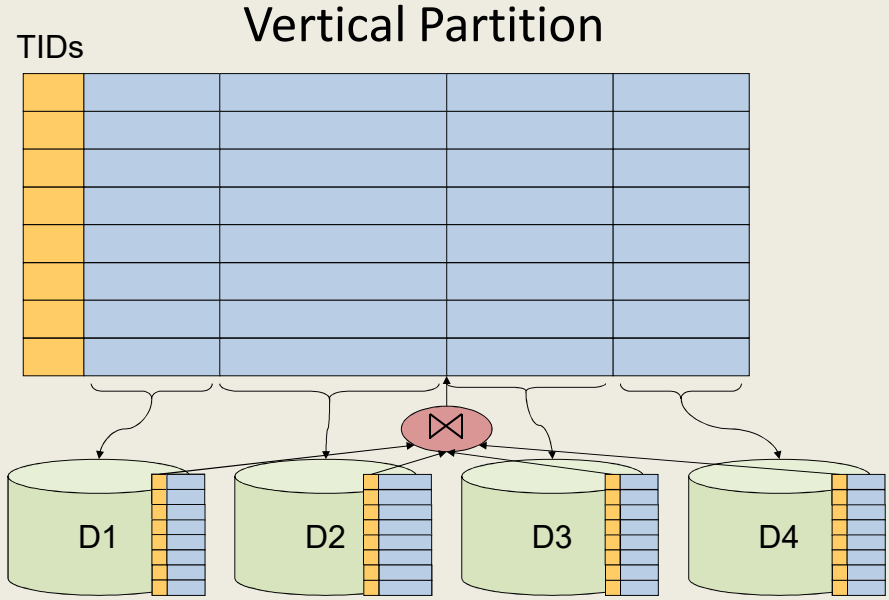
Parallel Scan for Horizontal Partition (2)

- Round-Robin Partitioning
  - No Inter-Operation Parallelism
  - Intra-Operation Parallelism
  - Assignment for one processing element
    - Comparison: {R}/n
    - I/O: |R|/n
- Disk Utilization (Total)
  - Same as a Single Disk
- Update Cost
  - Update Can be done in Parallel

Parallel Scan for Vertical Partition (1)

- Transposed File
  - A tuple is divided into sub-tuples containing some attributes
    - Each sub-tuple has the same Tuple-Identifier TID
  - Tuples can be reconstructed by TID-Join
    - TID-Join: Eq-Join of same TID
  - Merit:
    - Decrease Disk I/O for queries requiring small number of attributes
  - Demerit:
    - Cost for TID-Join (Especially for large number of attributes)

Vertical Partition



Partitioned attributes have TIDs to reconstruct original relation

Parallel Scan for Vertical Partition (2)

- Historical Background
  - The concept of vertical partitioning is NOT new
    - CPU Performance for TID-Join was a bottleneck
      - Vertical Partitioning had no effect for speed-up
  - The bottleneck move from CPU to Disk I/O
    - Less I/O by vertical partitioning becomes effective
- Partitioning Strategy
  - Depend on Queries
    - If the queries have the same pattern, vertical partitioning is a good means