SHARED-NOTHING PARTITIONING

Iztok Savnik

Literature

- Sam Lightstone, Toby Teorey, Tom Nadeau, Physical Database Design, Morgan Kaufmann Publishers, 2007.
  - Chapter 6

Shared-nothing architecture

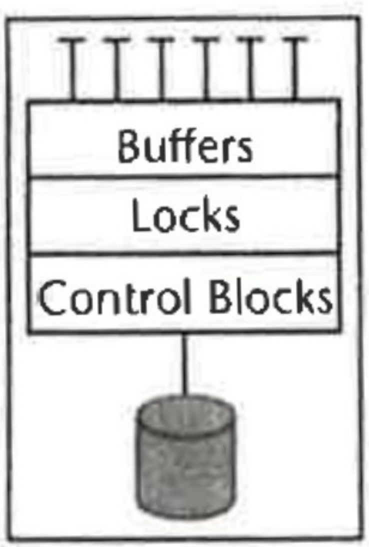
- Henry Ford (1863-1947)
  - Nothing is particulary hard if you divide it into small jobs.
- Shared-nothing is a divide-and-conquer strategy for solving
  - Hard problems defined over large data sets
- Divide and conquer is a well known technique in CS
  - Large data set is split into parts
  - One computer solves one part

Shared-nothing architecture

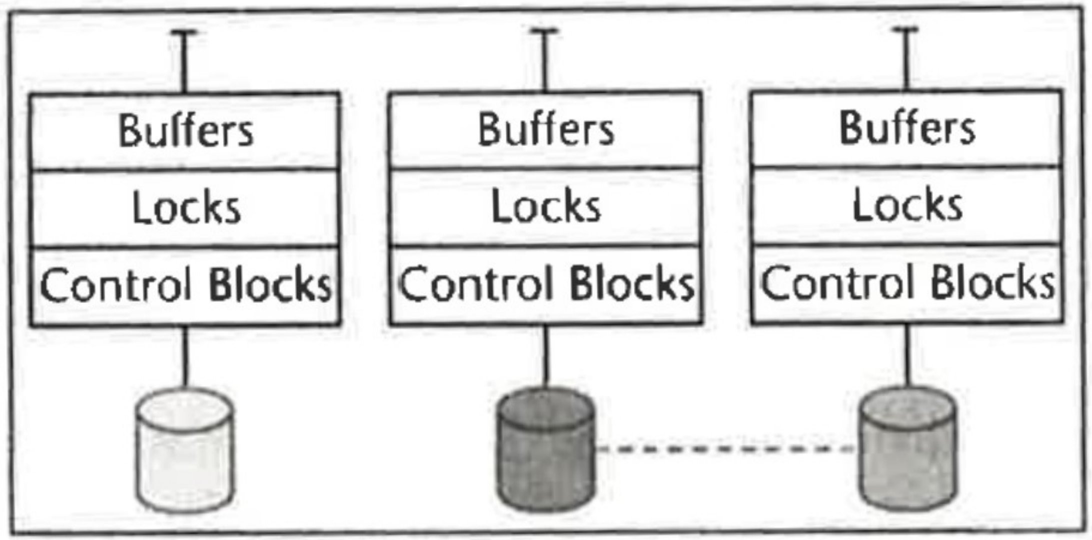
- One computer solves one part
  - Queries are executed in a fraction of the time
- Partial results are integrated into final solution

Shared-nothing architecture

- Extremes of parallel-processing architecture
  - Shared everything and
  - Shared nothing
- Shared everything
  - Single computer: shared memory, shared disk and shared bank of CPUs
  - Symetric Multiprocessor (SMP)
    - Nonuniform Memory Architecture (NUMA)
    - NUMA is subset of SMP



Shared Everything



Shared Nothing

npb9, shared-nothing

Shared-nothing architecture

- Shared nothing
  - Sets of relatively independent servers working cooperatively on subsets of a problem
  - Occasionally these servers will need to share data
    - High-spead interconnect
  - Ability to scale out to a very large number of servers

Shared-nothing architecture

- Three major products that offer shared-nothing architecture
  - DB2, Data Partitioning Facility
    - Components, servers, disks and network interconnections
  - Informix, Extended Parallel Server (XPS)
    - Commodity components
  - NCR Teradata
    - Commodity disk servers, special hardware interconnect

Shared-nothing architecture

- IBM Netezza
  - Shared-nothing architecture
  - Powerful business inteligence
- Shared-nothing architectures continue to dominate the industry for large complex data
  - Complex analysis is often required

Shared-nothing architecture

- Several servers collaborate to solve single problem
  - Each server owns a fragment of data
    - No access to other disks
    - Each server operates on a distinct subset of the database
      - Uses its own resources to perform analysis of the fragment
    - Resultas are gathered by the designated server
  - Each server is called »node« or a »partition«

Shared-nothing architecture

- The designated server that collects the results is called »coordinator«
  - May itself include a partition
  - Gathers the results and reports to the client
  - Has the deeper view of the cluster (where is what, which nodes do which task, ...)
  - Has no view on the activity of nodes except through network communications

Shared-nothing architecture

- To see how this improves performance and scalability consider the following simple aggregation query

```
Select SUM(SALES) from
MYSCHEMA.SALESDATA
where SALESDATE < '2006-11-17' and SALESDATE >
    '2004-01-01'
```

- If there is an index on [YEAR,SALES] the DBMS will scan the index

Shared-nothing architecture

- Example database
  - We have 3M entries in the database for a given range (1.1.2004–17.11.2006)
    - Single server would need to access 3M keys
  - We have 10 nodes => 300K keys per node
  - Each node computes sum of the 300K sales
  - 10 summations are passed to the coordinator
  - Each of nodes needed 1/10 of the time
- This impressive scale out has not been achieved with other multiprocessor architectures
- Shared-nothing MPPs have several thousand nodes

Shared-nothing scale out

- Casual observer could say that
  - the benefits achieved are simply the result of applying 10 times the processing power
  - Perhaps the use of single server with 10 times CPU, memory, etc. scales just as well.
- Problems with trying to increase processing time linearly by simply buying larger servers
  - First
    - server with 10 times the CPU power, 10 times memory, 10 times bus bandwidth, etc. may not be possible

Shared-nothing scale out

  - Even the largest NUMA systems are constrained by their bus architecture
  - What if num of nodes was 100 or 1000? Would it be possible to buy 1000 times faster system?
- Second
  - It is difficult to design the algorithms that scale linearly to perform parallel processing
  - If we could increase all resources by 10 on single machine
  - What could be the improvement of the algorithms?

Shared-nothing scale out

- There is »N^2 effect« discussed later
  - Natural consequence of how joins are processed in shared-nothing systems
  - Exponential gain in efficiency of join processing

Key concepts and terms

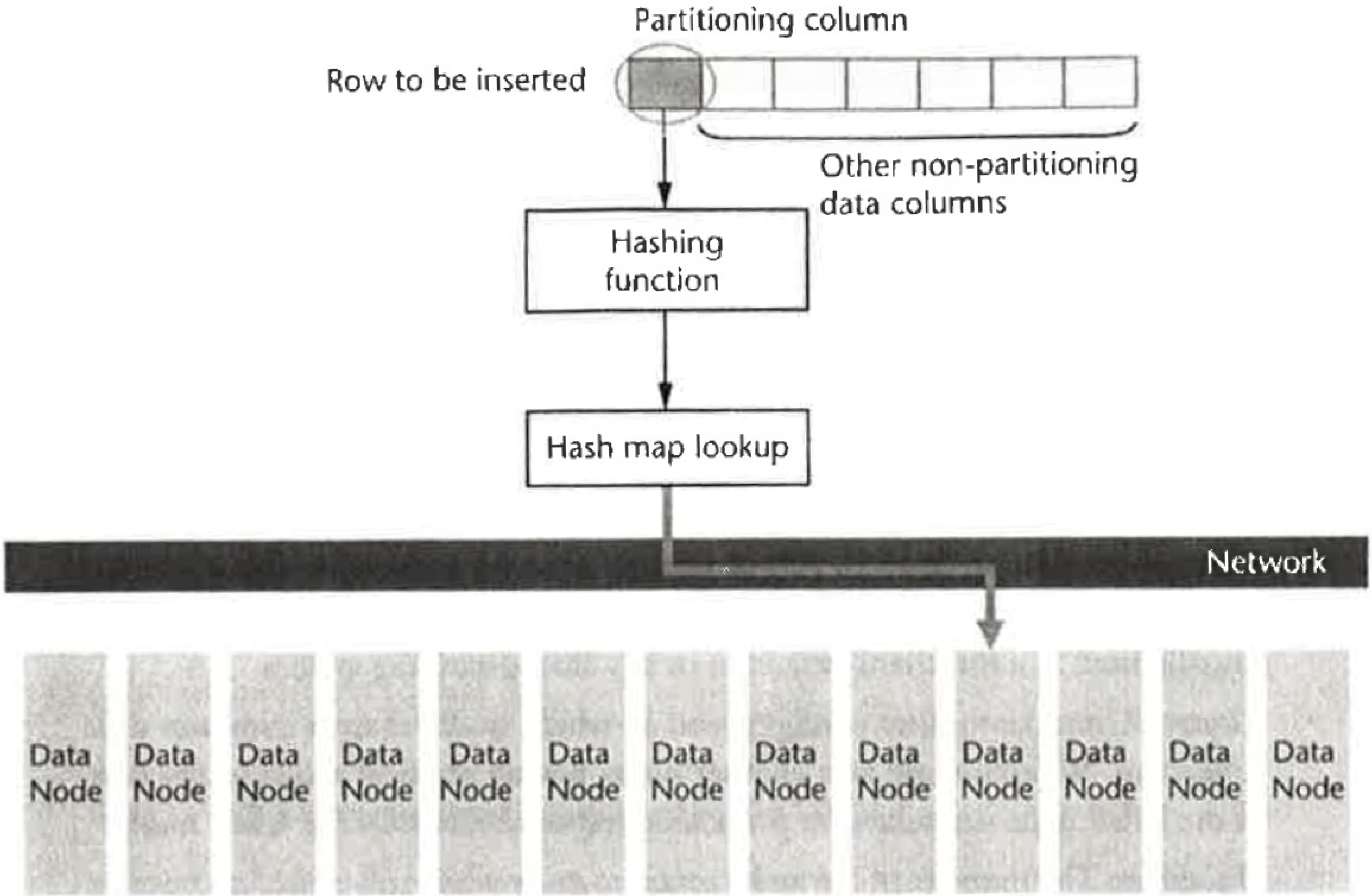
- Shared-nothing architecture
- Massively-parallel processing (MPP)
- Massively parallel processor (MPP)
- Cluster
- Scalability
- Linearity

Hash partitioning

- Most products that use shared-partitioning distribute records to the database nodes using hash function
  - Hash function maps one or more column values of each record to the numeric value
- Depending on the particular configuration
  - Shared-nothing system can have different number of nodes
  - The hash value can not be directly converted to node number

Hash partitioning

- *Hash map* or *partition map*
  - Mapping from hash values of records to node numbers
  - Each time the new node is added to MPP the hash map needs to be recomputed
  - Partition map has as many or more entries than the largest number of nodes supported



npb9, shared-nothing

Hash partitioning

- In DB2 the partitioning columns are defined during the table creation process
  - Extension of CREATE TABLE DDL
- In Teradata the partitioning is defined by primary index
  - Can be different to primary key of table (often is)
  - If primary index and primary key exist, the later is implemented as secondary index
  - User can have either unique or nonunique primary index

Hash partitioning

- Design goal for good partitioning
  - Minimize the data skew accross the nodes
  - Maximize colocation between tables for joins
- Choosing partitioning columns
  - Should have fairly large number of distinct values
  - Relatively limited data skew is a good design practise
- More on skew and collocation

Pros and cons of shared nothing

- 3 most commonly mentioned architectures for multiprocessor high transaction rate systems
  - shared memory (SM)
    - multiple processors shared a common central memory
  - shared disk (SD)
    - multiple processors each with private memory share a common collection of disks
  - shared nothing (SN)
    - neither memory nor peripheral storage is shared among processors

Pros and cons of shared nothing

- Michael Stonebreaker [1986]
  - On the benefits and drawbacks of shared nothing architecture
  - A table comparing attributes [1..3]
    - 1 is superior
- Difficulty of transaction management (1,2)
  - SM – few changes
  - SD – more complex, lock table hotspot
  - SN – difficult, distr.deadlock det., multiphase commit

System Feature	Shared nothing	Shared memory	Shared disk
Difficulty of concurrency control	2	2	3
Difficulty of crash recovery	2	1	3
Difficulty of data base design	3	2	2
Difficulty of load balancing	3	1	2
Difficulty of high availability	1	3	2
Number of messages	3	1	2
Bandwidth required	1	3	2
Ability to scale to large number of machines	1	3	2
Ability to have large distances between machines	1	3	2
Susceptibility to critical sections	1	3	2
Number of system images	3	1	3
Susceptibility to hot spots	3	3	3

npb9, shared-nothing

Pros and cons of shared nothing

- Data base design (row 3)
  - SM, SD – difficult, SN – harder
- Load balancing (row 4)
  - SN – hard, SM, SD - easier
- Next five points are fairly straightforward (rows 5-9)
- Critical sections
  - SM – hard, SD – less hard, SN – no problem

Pros and cons of shared nothing

- System images (row 10)
  - – SM – one image, SN, SD – one per CPU
  - – More administration
- Hot spots
  - – All susceptible to hot spots

Pros and cons of shared nothing

- Conclusions
  - SM does not scale to a large number of CPUs
  - SD excels at nothing
  - Justifications of flaws of SN
    - Stonebraker’s paper: problems are unlikely to be very significant
    - Recent internet DBMSs prove above

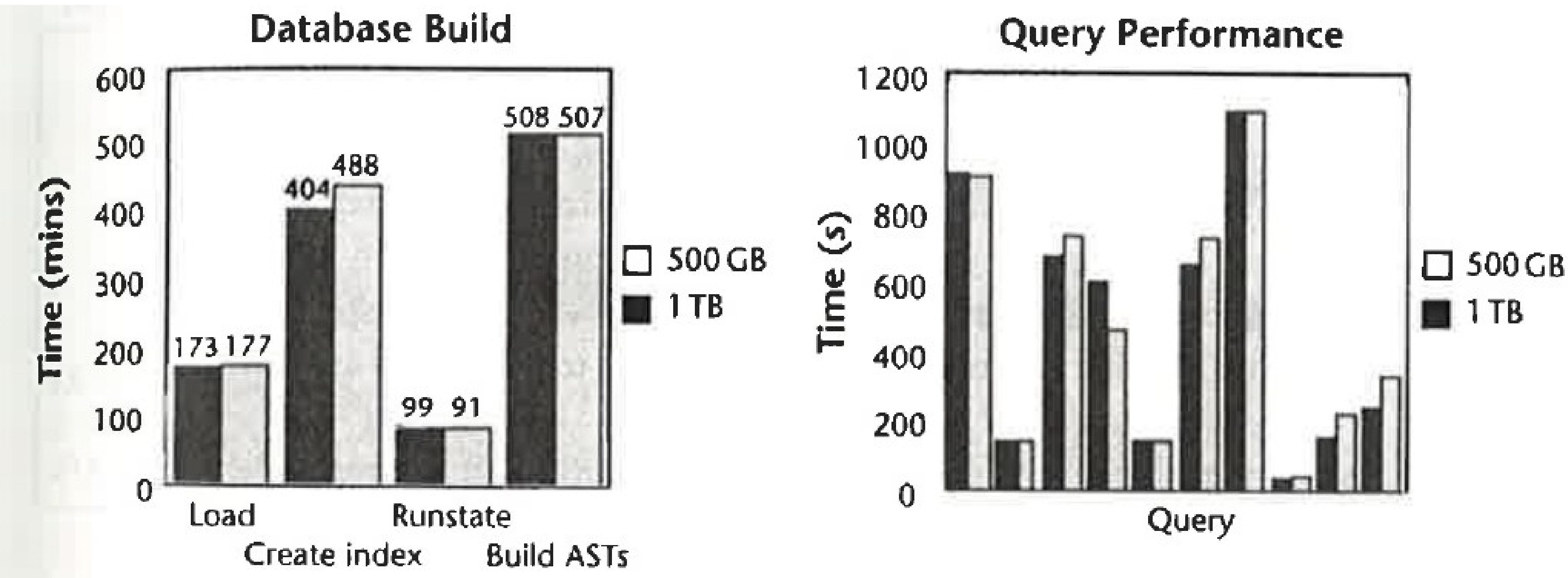
Pros and cons of shared nothing

- More conclusions
  - SN improves bandwidth and scalability
  - SN reduces susceptibility to critical sections
  - Negative aspects of SM can be summarized as »complexity«
    - Harder to design and manage
    - Complexity is critical limitation for mainstream?
    - The ability to outperform combined with advances in self-managing systems is shrinking these concerns

Pros and cons of shared nothing

- Positive aspects – what sets SM arch. apart
  - Impresive linearity and scale-out from complex bussiness inteligence workloads
- Experiment Goddard, 2005
  - Single-server 24-way system, 0.5 TB
  - Two-node 1 TB system, identical HW
    - Each system stores 0.5 TB
    - Data is hash partitioned, SN architecture
  - Experimental data
    - Near-linear scalability for DB build processing and query execution performance !

Experiment Goddard, 2005



npb9, shared-nothing

Pros and cons of shared nothing

- How to explain the dramatic difference in scalability?
- N^2 effect
  - Consider computational complexity of 30x30 join
  - Uniprocessor: single CPU performs 30x30
  - SMP with 3 CPUs working on the problem can be solved in 1/3 the time
  - 3-way MPP can solve this in parallel on 3 nodes obtaining 10x10 join on all 3 nodes

Pros and cons of shared nothing

- 30-way MPP can solve this in parallel on 30 nodes obtaining 1x1 join on all 30 nodes
- We assume data has been distributed perfectly
  - Join data is perfectly collocated
  - Part of computation has been done by distribution
  - In practice carful selection of the partitioning keys can give good collocation

N^2 effect

Processor type	Computational complexity	Total execution time in arbitrary units
Uniprocessor	$30 \cdot 30$	900
SMP (3-way)	$30 \cdot 30/3$	300
MPP (3-way)	$10 \cdot 10 \cdot 3/3$	100
MPP (30-way)	$1 \cdot 1 \cdot 30/30$	1

npb9, shared-nothing

Skew and collocation

- First impression
  - Node operates on fragment of data
  - Results are merged in a trivial way
  - Leading to amazing peformance
- Challenges
  - Getting the design right
  - The scalability is rearly perfectly linear

Skew and join collocation

- Most SN systems use hash partitioning
  – Columns are selected as the »partitioning keys|columns«
  – Hash partitioning assigns records to nodes
- Ultimate goal of SN architecture
  – Keep all nodes busy working in a linear fashion on larger DB problems
  – Minimize the degree of data sharing
- Two serious problems
  – Skew and join collocation

Data skew

- In order for SM to be effective data must be distributed evenly
  - If some node has significantly more data, computation will take longer
  - Comp.time is limited to the slowest node
  - In the case of range partitioning this is nearly impossible
    - Each data range includes different number of records
    - Example: sales records partitioned by date

Data skew

- Even very fair hash function large density of data at some points may seriously skew data
  - Example:
    - Hash partitioning sales into weeks may give huge peek in the last week of December
    - Nodes would therefore store disproportionate volume of data
- The solution
  - Not improvement of the hash function
  - Be careful what columns are chosen for partitioning
    - Date of sale may be poor choice
    - Product ID would also seriously skew the data since some products will be more popular

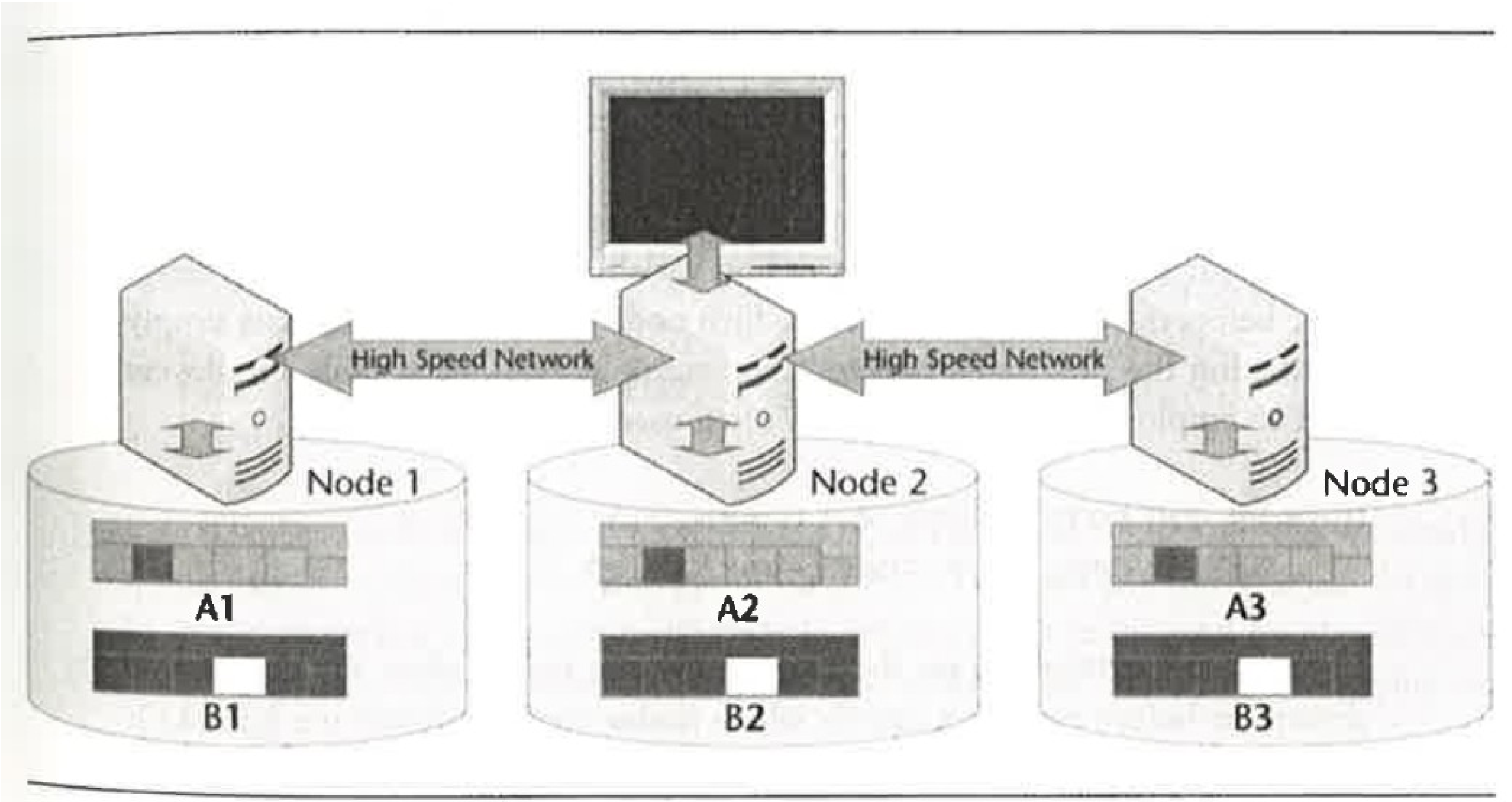
Data skew

- To achieve even distribution select columns that:
  - Have several times more unique values than the number of nodes in the system
  - Have reasonable even distribution of data

Collocation

- In order for SN architecture to scale well
  - Communication should be kept to a minimum
- A common problem
  - Tables that need to be joined are not collocated
  - Data from one node needs to be shiped to another node
  - This is expensive and can cripple benefits of SN

Collocation between two tables



Collocation

- Collocation is placement of rows from different tables that contain related data in the same DB node
  - Tables A and B (from figure) are hashed accross 3 nodes
  - Shaded sections show the join data
  - If data is collocated, the shaded section of A will join to shaded sction of B
  - Worst case scenario: entire data for one table has to be shipped to other nodes for joining

Collocation

- Interesting and challenging design goal
  - Find partitioning keys that achieve good collocation as well as even distribution
  - Rows in collocated tables with the same partitioning key value are always placed in the same DB partition