“Redesign of Distributed Relational Databases”
Perspectives after thirty years!

Kamalakar Karlapalem
IIIT, Hyderabad, India
kamal@iiit.ac.in
April 28th 2023
at College of Computing, Georgia Tech, Atlanta, USA.

Based on Joint DASFAA 2023 tutorial with Satya Valluri, Databricks USA.
Outline

• Background
• Redesign of Distributed Relational Databases
• Aspects of Current Distributed Database Processing
• Local Parallel/Distributed & Globally distributed
• Summary
Background
A distributed database as a collection of multiple, logically interrelated databases located at the nodes of a distributed system.
  - Multiple sites connected by a WAN (may be LAN)

A Distributed database Management system as the software system that permits the management of the distributed database and makes the distribution transparent to the users.
  - Key point is ‘distribution transparent to users’.
SELECT ENAME, SAL
FROM EMP, ASG, PAY
WHERE DUR > 12
AND EMP.ENO = ASG.ENO
AND PAY.TITLE = EMP.TITLE
Email kamal@iiit.ac.in for any questions on this slide
Distributed Database

Communication Subsystem

DBMS Software

User Query

User Application

DBMS Software

User Query

User Application

User Query
Distributed Query Execution – high level view

1. Identify the relations of distributed database the query specifies
2. Check whether the relations are stored in one site or multiple sites
3. Determine which relations from which sites need to be accessed
4. Identify data transfer from one site to another to perform query operations
5. Steps 3-4, are determine the query execution plan by considering the data access cost and data transfer costs
6. Query result is got by executing the query execution plan
Distributed Database Design Problem

Given a set of queries (SQL statements on Relations)

• [Fragmentation] Determine the fragments of Relations so that queries access as less irrelevant data as possible

• [Allocation] Allocate the fragments to sites so that queries transfer as less data as possible between the sites

No optimal or best possible solution for the above problems.
Mixed Fragmentation Methodology
Mixed fragmentation Methodology

Given a set of queries

- [Horizontal Fragments (HF)] Use predicates (Where clauses of SQL query) to horizontally fragment a relation
- [Vertical Fragments (VF)] Use attributed accessed (Select part of SQL query) to vertically fragment a relation
- Apply both HF and VF to get grid cells (small fragment) each pertaining to one VF and one HF
- Merge grid cells to get Fragments called Mixed fragments. Queries access mixed fragments

A mixed fragmentation methodology for initial distributed database design

Navathe, Karlapalem, Ra

Journal of Computer and Software Engineering 1995
### Mixed Fragments

#### Table 1
<table>
<thead>
<tr>
<th>Emp#</th>
<th>Name</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Tom</td>
<td>LA</td>
</tr>
<tr>
<td>4</td>
<td>Ann</td>
<td>NY</td>
</tr>
</tbody>
</table>

#### Table 2
<table>
<thead>
<tr>
<th>Emp#</th>
<th>Name</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ian</td>
<td>All</td>
</tr>
<tr>
<td>2</td>
<td>Jim</td>
<td>SF</td>
</tr>
<tr>
<td>7</td>
<td>Ram</td>
<td>BOM</td>
</tr>
</tbody>
</table>

#### Table 3
<table>
<thead>
<tr>
<th>Emp#</th>
<th>Name</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Kate</td>
<td>NO</td>
</tr>
<tr>
<td>6</td>
<td>John</td>
<td>LDN</td>
</tr>
</tbody>
</table>

#### Table 4
<table>
<thead>
<tr>
<th>Emp#</th>
<th>Name</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Mary</td>
<td>STL</td>
</tr>
<tr>
<td>9</td>
<td>Vijay</td>
<td>DEL</td>
</tr>
</tbody>
</table>

#### Diagram
- $f_1$, $s_1$, $f_2$, $s_2$, $f_3$, $s_3$, $f_4$, $s_4$, $f_5$, $s_5$
Redesign Problem
Redesign Problem

• Due to changes in queries or applications a given fragmentation and allocation schema does not perform well and has to be changed.

• **Corrective Redesign:** Every once in a while use the fragmentation and allocation algorithms to generate a new design based on revised set of queries.

• **Preventive Redesign:** In case we know the queries that will execute next, one can change the design to suit those set of queries.

• Adaptive Redesign: The design adapts to the current query mix. The resign is a continuous process identifying queries whose performance has to be enhanced through redesign.

Email kamal@iiit.ac.in for any questions on this slide
Materialization Problem
Materialization Problem

• A populated distributed database that adheres to a given fragmentation and allocation schema exists.
• A new fragmentation and allocation schema is decided.
• The data in the populated distributed database has to be materialized as per the new fragmentation and allocation schema.
• Materialization takes time, and would require handling of queries and transactions while it happens.
• Algorithms to correctly materialize new fragmentation and allocation schema were developed.
<table>
<thead>
<tr>
<th>Employee</th>
<th>Name</th>
<th>City</th>
<th>Dept</th>
<th>Proj</th>
<th>Salary</th>
<th>Bonus</th>
<th>School</th>
<th>Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Jan</td>
<td>Ad</td>
<td>10</td>
<td>23</td>
<td>20K</td>
<td>1k</td>
<td>GeTech</td>
<td>BS</td>
</tr>
<tr>
<td>2</td>
<td>Jim</td>
<td>SF</td>
<td>11</td>
<td>24</td>
<td>50K</td>
<td>2k</td>
<td>UnivIL</td>
<td>MS</td>
</tr>
<tr>
<td>3</td>
<td>Tom</td>
<td>LA</td>
<td>2</td>
<td>231</td>
<td>20K</td>
<td>1k</td>
<td>Duke</td>
<td>BS</td>
</tr>
<tr>
<td>4</td>
<td>Ann</td>
<td>NY</td>
<td>2</td>
<td>231</td>
<td>65K</td>
<td>1k</td>
<td>GeTech</td>
<td>BS</td>
</tr>
<tr>
<td>5</td>
<td>James</td>
<td>MO</td>
<td>11</td>
<td>25</td>
<td>25K</td>
<td>1k</td>
<td>UI-IIT</td>
<td>BS</td>
</tr>
<tr>
<td>6</td>
<td>John</td>
<td>LDN</td>
<td>32</td>
<td>256</td>
<td>118K</td>
<td>1k</td>
<td>GeTech</td>
<td>Ph.D</td>
</tr>
<tr>
<td>7</td>
<td>Ben</td>
<td>BOM</td>
<td>13</td>
<td>305</td>
<td>68K</td>
<td>6k</td>
<td>IT-MTech</td>
<td>BS</td>
</tr>
<tr>
<td>8</td>
<td>Mary</td>
<td>STL</td>
<td>33</td>
<td>213</td>
<td>41K</td>
<td>1k</td>
<td>NYU</td>
<td>BS</td>
</tr>
<tr>
<td>9</td>
<td>Vijay</td>
<td>DEL</td>
<td>33</td>
<td>213</td>
<td>49K</td>
<td>1k</td>
<td>GeTech</td>
<td>BS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Employee</th>
<th>Name</th>
<th>City</th>
<th>Dept</th>
<th>Proj</th>
<th>Salary</th>
<th>Bonus</th>
<th>School</th>
<th>Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ian</td>
<td>Ad</td>
<td>10</td>
<td>123</td>
<td>30K</td>
<td>1k</td>
<td>GeTech</td>
<td>BS</td>
</tr>
<tr>
<td>2</td>
<td>Jim</td>
<td>SF</td>
<td>11</td>
<td>124</td>
<td>50K</td>
<td>2k</td>
<td>UnivIL</td>
<td>MS</td>
</tr>
<tr>
<td>3</td>
<td>Tom</td>
<td>LA</td>
<td>2</td>
<td>155</td>
<td>33K</td>
<td>1k</td>
<td>Duke</td>
<td>BS</td>
</tr>
<tr>
<td>4</td>
<td>Ann</td>
<td>NY</td>
<td>2</td>
<td>123</td>
<td>65K</td>
<td>1k</td>
<td>GeTech</td>
<td>BS</td>
</tr>
<tr>
<td>5</td>
<td>James</td>
<td>MO</td>
<td>11</td>
<td>125</td>
<td>25K</td>
<td>1k</td>
<td>UI-IIT</td>
<td>BS</td>
</tr>
<tr>
<td>6</td>
<td>John</td>
<td>LDN</td>
<td>32</td>
<td>256</td>
<td>118K</td>
<td>1k</td>
<td>GeTech</td>
<td>Ph.D</td>
</tr>
<tr>
<td>7</td>
<td>Ben</td>
<td>BOM</td>
<td>13</td>
<td>305</td>
<td>68K</td>
<td>6k</td>
<td>IT-MTech</td>
<td>BS</td>
</tr>
<tr>
<td>8</td>
<td>Mary</td>
<td>STL</td>
<td>33</td>
<td>213</td>
<td>41K</td>
<td>1k</td>
<td>NYU</td>
<td>BS</td>
</tr>
<tr>
<td>9</td>
<td>Vijay</td>
<td>DEL</td>
<td>33</td>
<td>213</td>
<td>49K</td>
<td>1k</td>
<td>GeTech</td>
<td>BS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Employee</th>
<th>Name</th>
<th>City</th>
<th>Dept</th>
<th>Proj</th>
<th>Salary</th>
<th>Bonus</th>
<th>School</th>
<th>Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ian</td>
<td>Ad</td>
<td>10</td>
<td>123</td>
<td>30K</td>
<td>1k</td>
<td>GeTech</td>
<td>BS</td>
</tr>
<tr>
<td>2</td>
<td>Jim</td>
<td>SF</td>
<td>11</td>
<td>124</td>
<td>50K</td>
<td>2k</td>
<td>UnivIL</td>
<td>MS</td>
</tr>
<tr>
<td>3</td>
<td>Tom</td>
<td>LA</td>
<td>2</td>
<td>155</td>
<td>33K</td>
<td>1k</td>
<td>Duke</td>
<td>BS</td>
</tr>
<tr>
<td>4</td>
<td>Ann</td>
<td>NY</td>
<td>2</td>
<td>123</td>
<td>65K</td>
<td>1k</td>
<td>GeTech</td>
<td>BS</td>
</tr>
<tr>
<td>5</td>
<td>James</td>
<td>MO</td>
<td>11</td>
<td>125</td>
<td>25K</td>
<td>1k</td>
<td>UI-IIT</td>
<td>BS</td>
</tr>
<tr>
<td>6</td>
<td>John</td>
<td>LDN</td>
<td>32</td>
<td>256</td>
<td>118K</td>
<td>1k</td>
<td>GeTech</td>
<td>Ph.D</td>
</tr>
<tr>
<td>7</td>
<td>Ben</td>
<td>BOM</td>
<td>13</td>
<td>305</td>
<td>68K</td>
<td>6k</td>
<td>IT-MTech</td>
<td>BS</td>
</tr>
<tr>
<td>8</td>
<td>Mary</td>
<td>STL</td>
<td>33</td>
<td>213</td>
<td>41K</td>
<td>1k</td>
<td>NYU</td>
<td>BS</td>
</tr>
<tr>
<td>9</td>
<td>Vijay</td>
<td>DEL</td>
<td>33</td>
<td>213</td>
<td>49K</td>
<td>1k</td>
<td>GeTech</td>
<td>BS</td>
</tr>
</tbody>
</table>
Multiple Query Optimization Solution
Estimating Transaction Response Time
Local and Distributed Transactions

Local Transaction

- Begin Transaction
- Initiation Phase
- Database Access & Processing Phase
- Commitor Abort Phase
- End Transaction

Distributed Transaction

- Begin Transaction
- Initiation Phase
- Sub-Transaction Spawning
- Initiation Phase
- Database Access & Processing Phase
- Commitor Abort Phase
- End Transaction
Simulation Model for Lock Manager Contention
10 transactions/second

15 Transactions per second
Preventive Redesign
Preventive Redesign Policy

• Use Markov Decision Process

• Basis
  • If the current set of applications changes from set A to set B, and if the current distributed database design is D1, then the policy gives the Design that needs to be used for the longer-term optimal execution of all transactions.

• Inputs: Probability Transition Matrix, Reward Matrix, Materialization cost Matrix

• Output: Redesign Policy
Discrete Markov Process and Redesign Policy

Legend:
A - Application
D - Design
● - State of Markov Process

Discrete Markov Process

Change Point

Redesign Policy for the State A_i D_j

Email kamal@iiit.ac.in for any questions on this slide
Salient Points – in 1992

• Redesign incurs materialization cost
• Preventive redesign works when the application classes are stated clearly
• Adaptive design works when materialization costs can be amortized
• Corrective redesign is straightforward but incurs materialization costs.
Design Principles of Modern Distributed Database Systems
Design principles of modern distributed DBMS

- Are there a set of fundamental design principles behind building modern distributed database systems?
- Can these principles be organized into a set of design decision dimensions?
  - What are the tradeoffs of the design decisions?
- Design dimensions
  - D1: Resource Sharing Model
  - D2: Is physical storage optimized for querying/updates?
  - D3: Distribution Transparency
  - D4: Storage and Compute Separation
  - D5: Storage Compute Capability
D1: Resource Sharing Model

1. Shared Everything
   a. All nodes share access to single pool of resources (memory, processors & disk storage)
   b. All nodes run the same DBMS software and have access to the same data and metadata
   c. Examples: IBM DB2, Oracle Database, and Microsoft SQL Server
   d. However, for better scalability and performance these systems moved away to other hybrid models

2. Shared Disk
   a. Each node has its own memory and processors
   b. All processes can access the same disk
   c. Each node runs its own instance of the DBMS software, but all of the instances share access to the same disk storage
   d. Examples: Oracle Real Application Clusters (RAC), Microsoft SQL Server Failover Clustering, and IBM DB2 PureScale
D1: Resource Sharing Model

3. Shared Memory
   a. All nodes can read and manipulate data from same physical address space (main memory)
   b. Ideal for high-performance, parallel processing applications: multiple nodes can access and manipulate the same data at the same time, without the need for explicit communication between them
   c. Examples: Oracle TimesTen, IBM solidDB, and SAP HANA

4. Shared Nothing
   a. Each node has its own independent memory, processors, and disk storage
   b. Can scale horizontally, by adding more nodes to the system as needed
   c. Data is partitioned across multiple nodes, with each node responsible for a portion of the data
   d. Examples: Apache Cassandra, MongoDB Sharded Clusters, and Amazon Redshift.
D2: Is physical storage optimized for querying/updates?

- Is the physical storage layout of the data optimized so that it can be efficiently read and updated by queries later?
- Two main dimensions
  - D2.1: Physical shape of the data: Columnar
  - D2.2: Partitioning/Sharding based on column values
D2: Is physical storage optimized for querying/updates?

D2.1: Columnar representation of the data

- Data is stored in columns rather than rows
- More efficient storage and processing of data
- Well-suited for analytical workloads and reporting that typically involve aggregations and computations over large datasets

Example Systems
- Apache Cassandra, Amazon Redshift, Google BigQuery, Vertica

Hybrid Systems
- SAP HANA
  - Hybrid row and column storage based on characteristics of the data
- MemSQL
  - Row-based layout for transaction processing and columnar storage for analytics workloads
D2: Is physical storage optimized for querying/updates?

D2.2: Partitioning/Sharding based on column values

- Split large datasets into smaller, more manageable chunks
- Typically range-based, hash-based, or list-based
- Offers scalability, fault tolerance, and high availability
- Example Systems
  - Apache Cassandra: Uses ring-based partitioning scheme to distribute data across a cluster of nodes.
  - MongoDB: Uses sharding to horizontally scale data across multiple servers.
  - Apache HBase: Range-based partitioning to store data in HDFS
  - CockroachDB: Range-based partitioning to store data across a cluster of nodes.
  - Google Cloud Spanner: Uses a combination of range-based partitioning and TrueTime, a globally synchronized clock, to achieve strong consistency and high availability.
D3: Distribution Transparency

- Degree to which the distributed nature of the system is hidden from users and applications
  1. **Location transparency:** Hides the physical location of data
     a. Users interact with the system as if all data were located in a single location
  2. **Replication transparency:** Hides the replication of data across multiple nodes
     a. Users interact with the system as if there were only a single copy of the data
  3. **Transaction transparency:** Hides the distributed nature of transaction processing
     a. Users interact with the system as if all transactions were processed at a single location
  4. **Fragmentation transparency:** Hides the partitioning/sharding of data
     a. Users interact with the system as if all data were stored in a single location
D3: Distribution Transparency

D3.1: Location Transparency

*Offer location transparency*

- Google Cloud Spanner
- Amazon Aurora
- CockroachDB
- Microsoft Azure Cosmos DB
- Oracle RAC
- Apache Cassandra
- Yugabyte
- Redis
- Apache Ignite
- MongoDB
- Databricks
- Snowflake
D3: Distribution Transparency

D3.2: Replication Transparency

Offer replication transparency

- Oracle RAC
- MongoDB
- Apache Cassandra
- CockroachDB
- TiDB
- Google Cloud Spanner
- Amazon Aurora
- FoundationDB
- Yugabyte

No/partial replication transparency

- Amazon SimpleDB
  - users cannot explicitly control the replication process or configure replication across regions
D3: Distribution Transparency

D3.3: Transaction Transparency

**Offer transaction transparency**

- Oracle RAC
- Apache Cassandra
- Amazon Aurora
- Google Cloud Spanner
- CockroachDB
- Redis
- Apache Ignite
- YugabyteDB
- VoltDB
- TiDB
- MemSQL

**No/partial transaction transparency**

- Amazon S3
  - Object store with no support for transactions
- Amazon SimpleDB
  - NoSQL database that supports eventual consistency
### D3: Distribution Transparency

#### D3.4: Fragmentation Transparency

<table>
<thead>
<tr>
<th>Offer fragmentation transparency</th>
<th>No/partial fragmentation transparency</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Google Cloud Spanner</td>
<td>- Redis Cluster</td>
</tr>
<tr>
<td>- Amazon Aurora</td>
<td>- sharding and partitioning of data is done manually by the user</td>
</tr>
<tr>
<td>- CockroachDB</td>
<td></td>
</tr>
<tr>
<td>- Microsoft SQL Server</td>
<td></td>
</tr>
<tr>
<td>- Oracle RAC</td>
<td></td>
</tr>
<tr>
<td>- Yugabyte</td>
<td></td>
</tr>
<tr>
<td>- Redis</td>
<td></td>
</tr>
<tr>
<td>- Apache Ignite</td>
<td></td>
</tr>
<tr>
<td>- VoltDB</td>
<td></td>
</tr>
</tbody>
</table>
D4: Storage and Compute Separation

- Storage and compute disaggregation: storage and processing of data are separated
  - Data is stored on a set of distributed storage nodes
  - Processing of the data is handled by a set of compute nodes
- Different from traditional database systems where data storage and processing are closely coupled in a single server
- Offers the ability to scale compute and storage resources independently, improved fault tolerance, and better resource utilization
- Challenges
  - Network latency due to large data movement
  - Data consistency
  - Security
D4: Storage and Compute Separation

With disaggregation
- Amazon Aurora
- Amazon Redshift
- Google Bigtable
- Microsoft Azure Cosmos DB
- Apache Cassandra Reaper
- Apache Hadoop
- Apache Spark
- CockroachDB
- TiDB
- YugabyteDB
- Databricks Delta Lake
- Snowflake
- Apache Iceberg
- Presto

No storage and compute disaggregation
- Oracle RAC
D5: Storage Compute Capability

- Typically, distributed databases use traditional storage devices such as HDDs or SSDs for storing data and separate compute nodes for processing queries and executing transactions.
- In-memory databases store data in memory and processed by the same nodes that store the data.
- Some distributed databases use specialized hardware, such as field-programmable gate arrays (FPGAs) or graphics processing units (GPUs), for accelerating certain types of computations.
D5: Storage Compute Capability

Distributed databases that use FPGAs with storage include:

- **AWS Elasticache for Redis**: FPGA-accelerated computation on Amazon EC2 F1 instances.
- **MemSQL**: FPGAs to accelerate query processing
- **Microsoft Azure SQL Database**: Feature called ‘Accelerated Database Recovery’ which uses FPGA-accelerated log processing to speed up database recovery.
- **OceanBase (Alibaba)**: FPGA-accelerated database service. Supports real-time data processing and analytics.
D5: Storage Compute Capability

Distributed databases that use GPUs with storage include:

- **BlazingSQL**: GPU-accelerated data science libraries for analytics
- **MapD/OmniSciDB**: Distributed analytics and visualization platform that uses GPUs
- **Kinetica**: Distributed in-memory database for real-time analytics
- **BrytlytDB**: Distributed GPU-accelerated relational DBMS
- **PG-Strom**: extension for the PostgreSQL that uses GPUs
- **ZillizDB**: Open-source distributed DBMS that offers GPU-accelerated data processing engine based on Apache Arrow
D5: Storage Compute Capability

Other example systems

- **Google Bigtable**: Distributed KV store
  - Uses Google's proprietary Colossus file system that can perform some computation, such as filtering and aggregation
  - *Cloud Bigtable filters* allow developers to specify code that is executed at various stages of data retrieval to do data validation, aggregation, transformation, and access control

- **YugabyteDB, CockroachDB**: Use RocksDB that is capable of performing basic computation on the data it stores, such as filtering, sorting, and aggregation
Summary

- **D1: Resource Sharing Model**
  - Shared Everything, Disk, Memory and Nothing
- **D2: Is physical storage optimized for querying/updates?**
  - Columnar vs Row storage
  - Partitioning/Sharding
- **D3: Distribution Transparency**
  - Location, Replication, Fragmentation and Transaction
- **D4: Storage and Compute Separation**
- **D5: Storage Compute Capability**
  - FPGA, GPU, Specialized file systems
Relook Distributed Database Design
Circa 2023
Parallel Query Execution Strategies

- Parallel DBMS community also considers some of the techniques (fragmentation, partitioning) of distributed database design.
- Systems like Gamma, and Teradata – use fixed horizontal fragmentation techniques based on random, range, and hash placement of rows across the parallel system.
- Onus is on simple query data localization and result combination.
- Current solutions like Map-Reduce, and others expand the horizontal fragmentation for efficient execution of queries while keeping query optimization simple.
- Vertica uses extreme vertical partitioning for storage and efficient query execution.
Local Parallel, Global Distributed Architecture (LPGD)

Global Distributed –
Consider globally distributed database with fragmentation and allocation with its aim to reduce – irrelevant data access and irrelevant data transfer

Local Parallel –
Use any of the parallel processing solutions to execute queries efficiently

Use any of the parallel processing solutions to execute queries efficiently

Use any of the parallel processing solutions to execute queries efficiently

Email kamal@iiit.ac.in for any questions on this slide
LPGD Architecture

• There is transparency between local parallel and globally distributed.
• Local systems can use efficient local storage and parallel processing solutions without impacting the globally distributed database.
• Global distributed database design can use advanced fragmentation and allocation techniques to determine local databases.
• The transparency and execution are complimentary.
• The distributed database designer can design while considering the parallel processing capability of local systems.
Query / Data Dependency

Data↓Query Access→

20% of queries 80% of Queries

20% of Data

Fragmentation
Sharding

80% of Data

Sharding

Fragmentation & Sharding

The data skew can provide for a 20%/80% split of data access and impact fragmentation. Knowing which fragment to access or not will reduce query execution cost.

18th April 2023
Email kamal@iiit.ac.in for any questions on this slide
Fragmentation and Storage Compute Capability

• One way to model storage compute capability is that storage dynamically delivers fragments based on query requirements.

• Fragmentation can complement storage computing capability by efficiently processing partial selects on fragments. Hence a finer level of fragmentation is not needed.

• The allocation can be simplified because of fewer fragments.
Distributed Database Designer’s choice

• Even if 2 TB of irrelevant data is accessed, it adds to about 1000 seconds to query execution time + additional data transfer and compute time, if any.

• Modern distributed database systems can execute queries that access even more irrelevant data and move irrelevant data.

• Thus, a coordinated effort to map fragmentation got from queries to exploit judiciously sharding to reduce irrelevant data access in a dynamic environment is a challenging problem.

• Over time, as database sizes even increase, the cost of storage and computing increases, and the dollar cost for query execution becomes a concern.

• Reward-based approaches to dynamically decide on sharding can work better if the relations are already fragmented.
Summary

• Introduced 30-year-old work on ‘redesign of distributed relational databases’

• Gave an overview of current distributed database systems architecture across various dimensions

• Provided a fragmentation perspective for distributed database designers for designing their databases.
References

References

16. J. Kleppmann, Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems, O'Reilly Media, 2017
References

References

36. Link: https://www.memsq.com/resources/the-no-limits-database/
40. IBM DB2: IBM DB2 with BLU Acceleration: The Always-On Enterprise Data Warehouse:  
41. Oracle Database: Oracle Database 19c: Data Management for the Modern Age:  
42. Microsoft SQL Server: Microsoft SQL Server 2019: Transform Data into Insights:  
43. Oracle Real Application Clusters (RAC): Oracle Real Application Clusters 12c Release 2 Technical Overview:  
   https://www.oracle.com/database/technologies/high-availability/rac/overview.html
44. Microsoft SQL Server Failover Clustering: SQL Server Failover Clustering (Windows):  
45. IBM DB2 PureScale: IBM DB2 pureScale: Scale out for cloud and analytics workloads:  
46. Oracle TimesTen In-Memory Database Overview and Architecture by Oracle Corporation, 2015. Available at:  
47. IBM solidDB Universal Cache: A Distributed Memory Cache for High Performance Applications by IBM Corporation, 2009. Available at:  
References

50. Amazon Redshift Architecture by Amazon Web Services, Inc., 2021. Available at: https://docs.aws.amazon.com/redshift/latest/dg/c_redshift-architecture.html
58. Distributed SQL Database - YugabyteDB. https://www.yugabyte.com/
59. Redis: https://redislabs.com/
References

64. Amazon Aurora documentation, https://aws.amazon.com/rds/aurora/
70. Presto documentation, https://prestodb.io/docs/current/
73. AWS Elasicache for Redis: https://aws.amazon.com/elasticache/redis/
75. OceanBase by Alibaba Cloud: https://www.alibabacloud.com/product/oceanbase
76. BlazingSQL: https://blazingsql.com/
78. Kinetica: https://www.kinetica.com/
79. BrytlytDB: https://www.brytlyt.com/
80. PG-Strom: https://github.com/heterodb/pg-strom
81. ZillizDB: https://github.com/zilliztech/db