“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

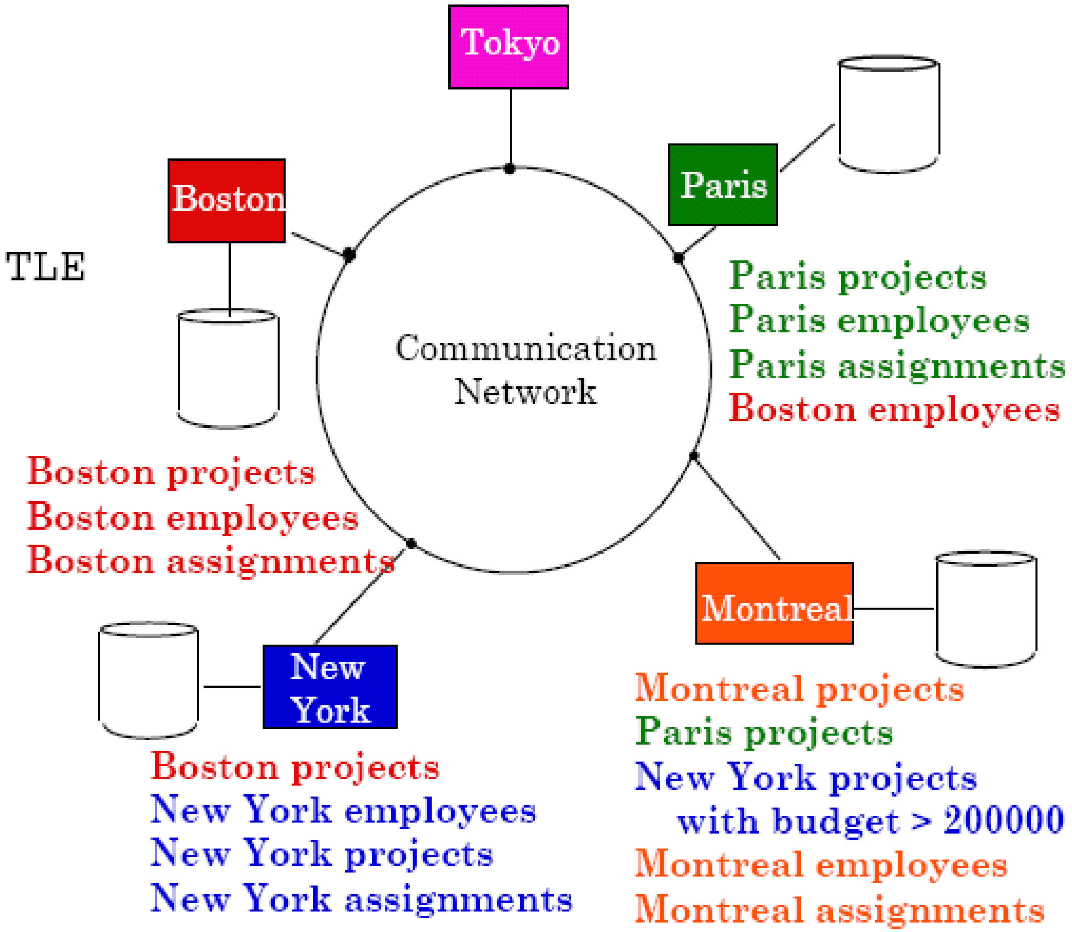
- Background
- Redesign of Distributed Relational Databases
- Aspects of Current Distributed Database Processing
- Local Parallel/Distributed & Globally distributed
- Summary

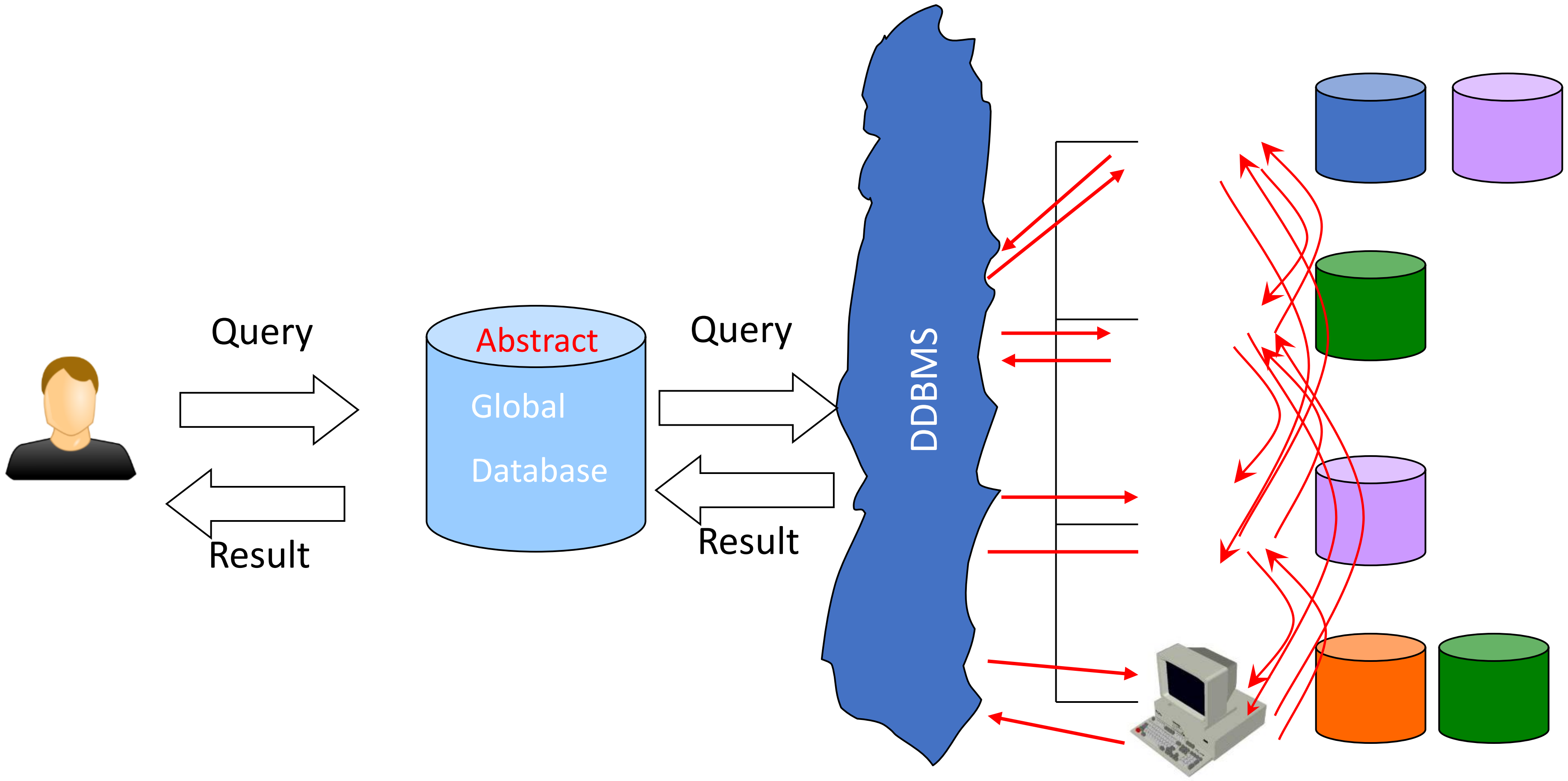
Background

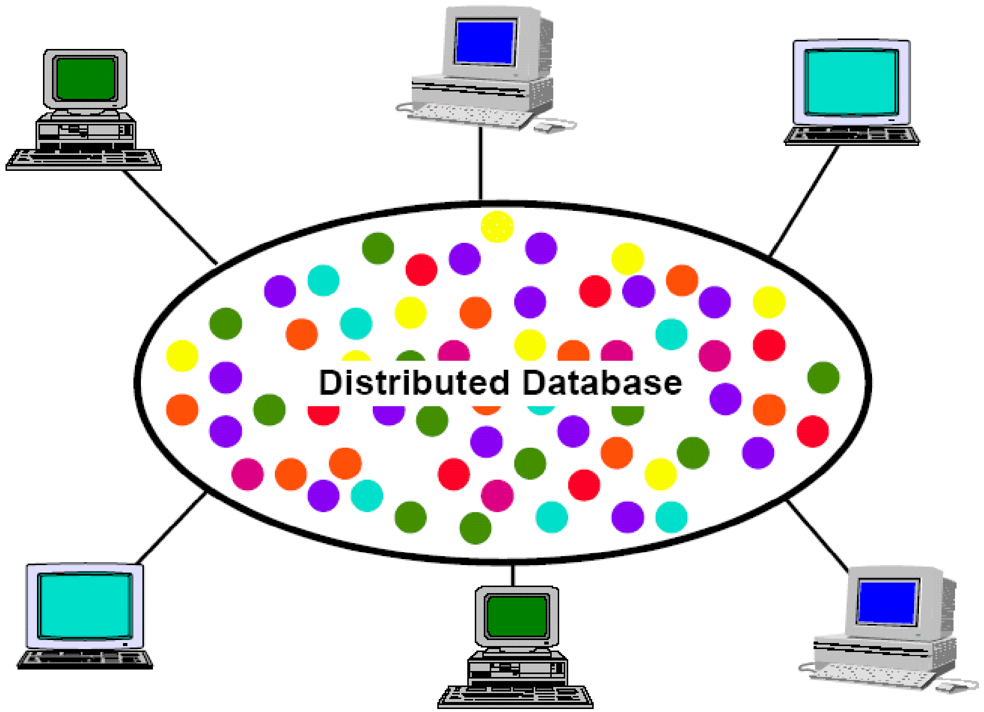
Distributed Database (1992) Ozsu & Valduriez

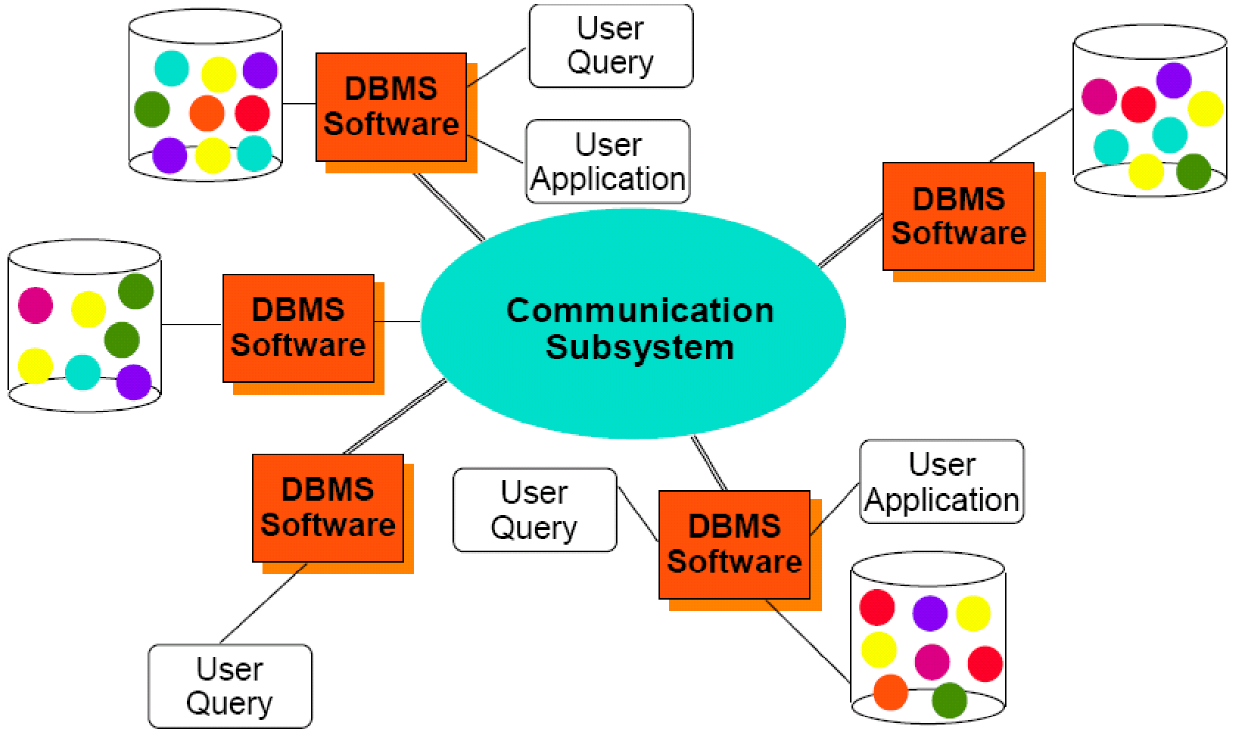
- A distributed database as a collection of multiple, logically interrelated databases located at the nodes of a distributed system.
  - Multiple sites connected by a WAN (may be LAN)
- A Distributed database Management system as the software system that permits the management of the distributed database 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
```




Query
Result
Abstract
Global
Database
Query
Result
DDBMS


Distributed Database


User
Query
DBMS
Software
User
Application
DBMS
Software
DBMS
Software
Communication
Subsystem
DBMS
Software
User
Query
DBMS
Software
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

1

Emp#	Name	City
3	Tom	LA
4	Ann	NY

Emp#	Dept#	Proj#
3	2	231
4	3	231

Emp#	Salary	Bonus
3	30k	1k
4	65k	5k

Emp#	School	Degree
3	Duke	BS
4	CalTech	MS

2

Emp#	Name	City
1	Ian	Atl
2	Jim	SF
7	Ram	BOM

Emp#	Dept#	Proj#
1	10	123
2	11	124
7	13	165

Emp#	Salary	Bonus
1	20k	1k
2	50k	2k
7	68k	6k

Emp#	School	Degree
1	GaTech	BS
2	UnivFL	MS
7	IIT	M.Tech

3

Emp#	Name	City
5	Kate	NO
6	John	LDN

Emp#	Dept#	Proj#
5	21	123
6	22	145

Emp#	Salary	Bonus
5	33k	2k
6	110k	8k

Emp#	School	Degree
5	UCB	BS
6	GaTech	Ph.D.

4

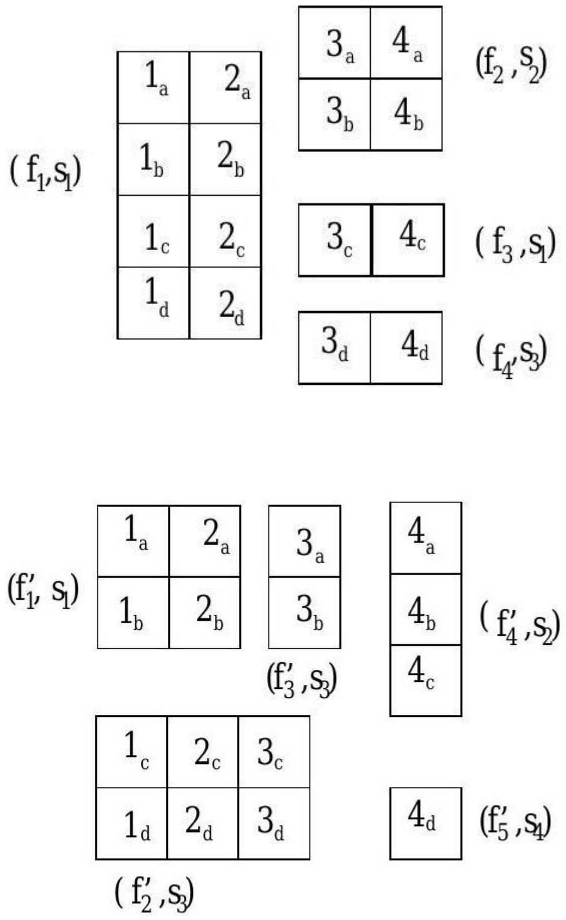
Emp#	Name	City
8	Mary	STL
9	Vijay	DEL

Emp#	Dept#	Proj#
8	33	213
9	33	213

Emp#	Salary	Bonus
8	45k	3k
9	40k	2k

Emp#	School	Degree
8	NYU	BS
9	GaTech	BS

a b c d



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.

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.

Employee

EMPLOYEE								
Emp#	Name	City	Dept#	Proj#	Salary	Bonus	School	Degree
1	Ian	Atl	10	123	20K	1K	GaTech	BS
2	Jim	SF	11	124	50K	2K	UnivFL	MS
3	Tom	LA	2	231	30K	1K	Duke	BS
4	Ann	NY	3	231	65K	5K	CalTech	MS
5	Kate	NO	21	123	33K	2K	UCB	BS
6	John	LDN	22	145	110K	8K	GaTech	Ph.D.
7	Ram	BOM	13	165	68K	6K	IIT	M.Tech
8	Mary	STL	33	213	45K	3K	NYU	BS
9	Vijay	DEL	33	213	40K	2K	GaTech	BS

Emp#	Name	City
3	Tom	LA
4	Ann	NY

Emp#	Dept#	Proj#
3	2	231
4	3	231

Emp#	Salary	Bonus
3	30k	1k
4	65k	5k

Emp#	School	Degree
3	Duke	BS
4	CalTech	MS

Emp#	Name	City
1	Ian	Atl
2	Jim	SF
7	Ram	BOM

Emp#	Dept#	Proj#
1	10	123
2	11	124
7	13	165

Emp#	Salary	Bonus
1	20k	1k
2	50k	2k
7	68k	6k

Emp#	School	Degree
1	GaTech	BS
2	UnivFL	MS
7	IIT	M.Tech

Emp#	Name	City
5	Kate	NO
6	John	LDN

Emp#	Dept#	Proj#
5	21	123
6	22	145

Emp#	Salary	Bonus
5	33k	2k
6	110k	8k

Emp#	School	Degree
5	UCB	BS
6	GaTech	Ph.D.

Emp#	Name	City
8	Mary	STL
9	Vijay	DEL

Emp#	Dept#	Proj#
8	33	213
9	33	213

Emp#	Salary	Bonus
8	45k	3k
9	40k	2k

Emp#	School	Degree
8	NYU	BS
9	GaTech	BS

EDP				
Emp#	Name	City	Dept#	Proj#
1	Ian	Atl	10	123
2	Jim	SF	11	124
3	Tom	LA	2	231
4	Ann	NY	3	231
7	Ram	BOM	13	165

ESB		
Emp#	Salary	Bonus
1	20k	1k
2	50k	2k
3	30k	1k
4	65k	5k
7	68k	6k

ESD1		
Emp#	School	Degree
1	GaTech	BS
2	UnivFL	MS
3	Duke	BS
4	CalTech	MS
7	IIT	M.Tech
5	UCB	BS
6	GaTech	Ph.D.

EDPS						
Emp#	Name	City	Dept#	Proj#	Salary	Bonus
5	Kate	NO	21	123	33k	2k
6	John	LDN	22	145	110k	8k
8	Mary	STL	33	213	45k	3k
9	Vijay	DEL	33	213	40k	2k

ESD2		
Emp#	School	Degree
8	NYU	BS
9	GaTech	BS

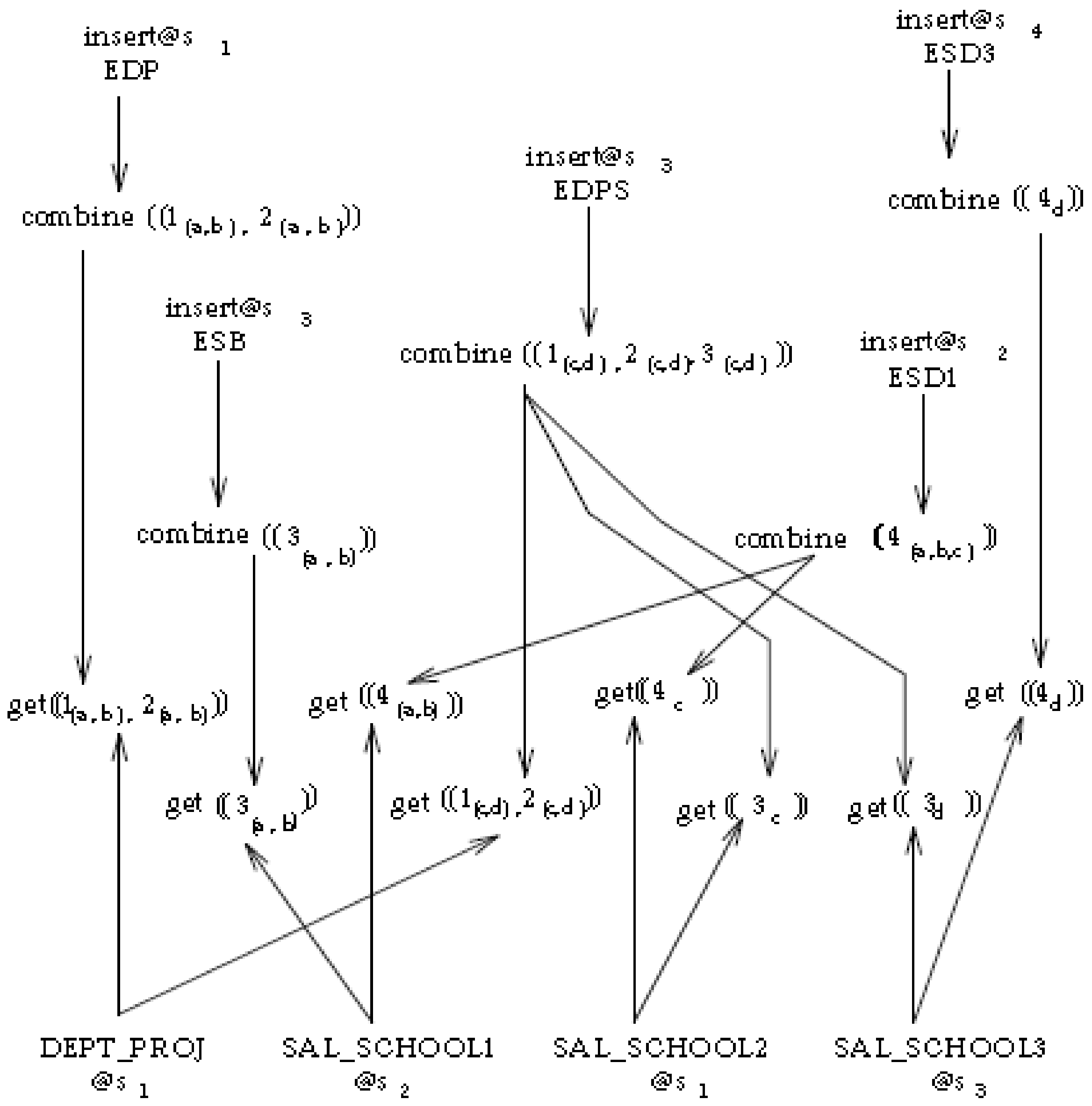
DEP_PROJ				
Emp#	Name	City	Dept#	Proj#
1	Ian	Atl	10	123
2	Jim	SF	11	124
3	Tom	LA	2	231
4	Ann	NY	3	231
7	Ram	BOM	13	165
5	Kate	NO	21	123
6	John	LDN	22	145
8	Mary	STL	33	213
9	Vijay	DEL	33	213

SAL_SCHOOL1				
Emp#	Salary	Bonus	School	Degree
1	20k	1k	GaTech	BS
2	50k	2k	UnivFL	MS
3	30k	1k	Duke	BS
4	65k	5k	CalTech	MS
7	68k	6k	IIT	M.Tech

SAL_SCHOOL2				
Emp#	Salary	Bonus	School	Degree
8	45k	3k	NYU	BS
9	40k	2k	GaTech	BS

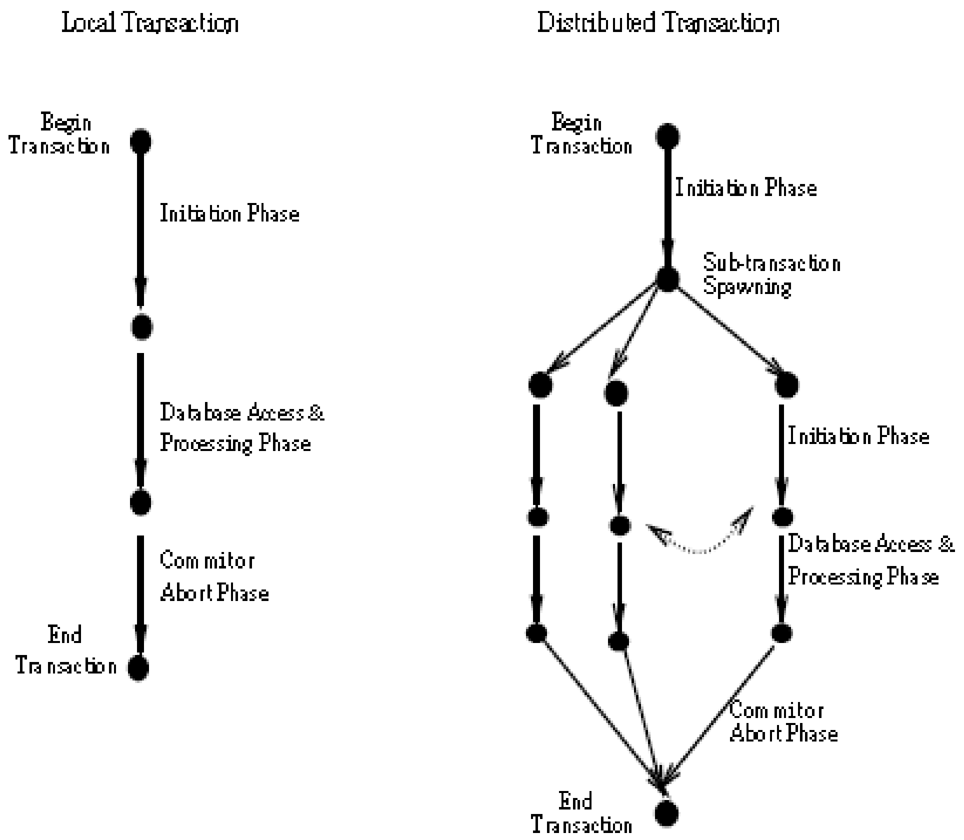
SAL_SCHOOL3				
Emp#	Salary	Bonus	School	Degree
5	33k	2k	UCB	BS
6	110k	8k	GaTech	Ph.D.

Multiple Query Optimization Solution

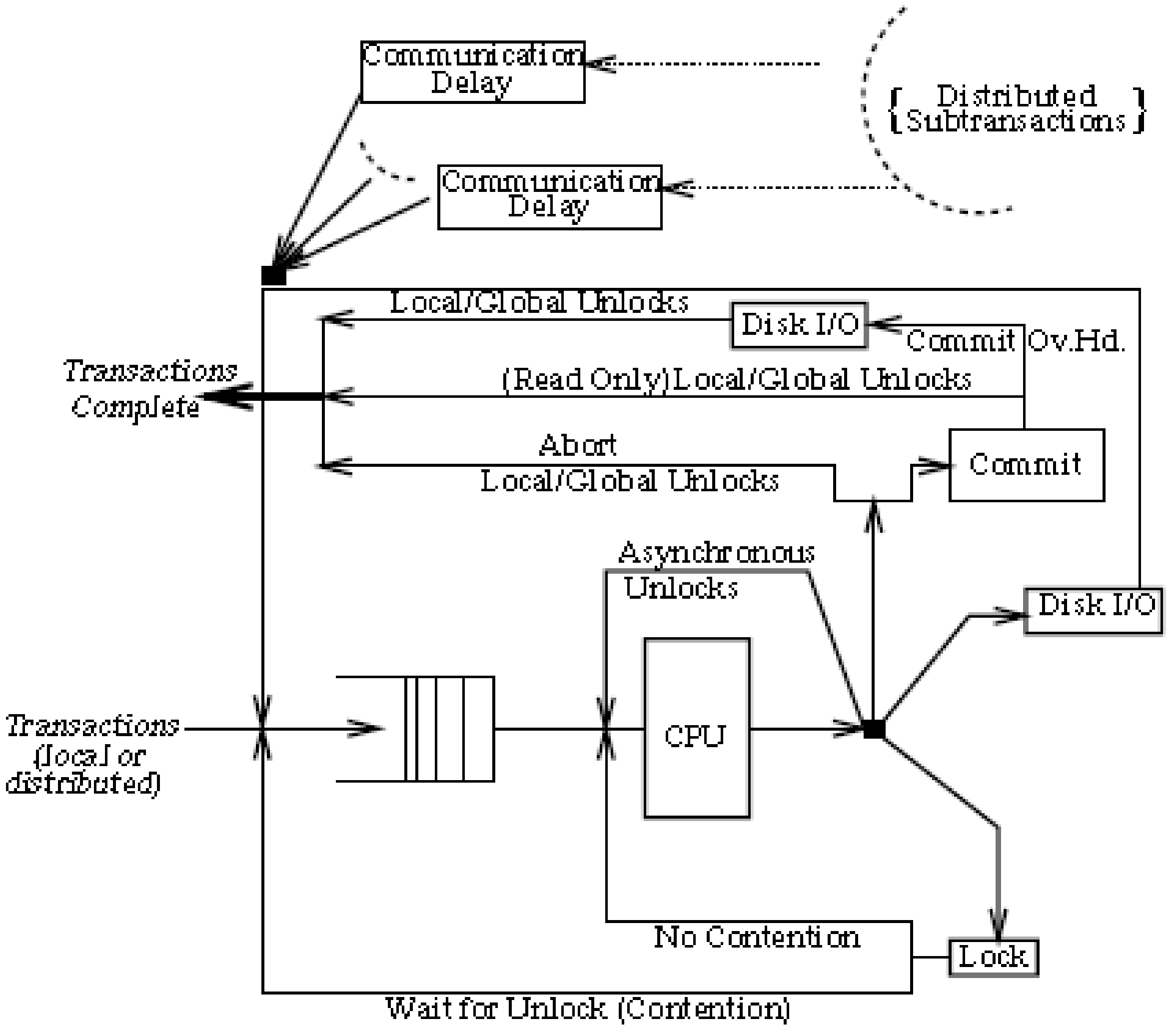


Estimating Transaction Response Time

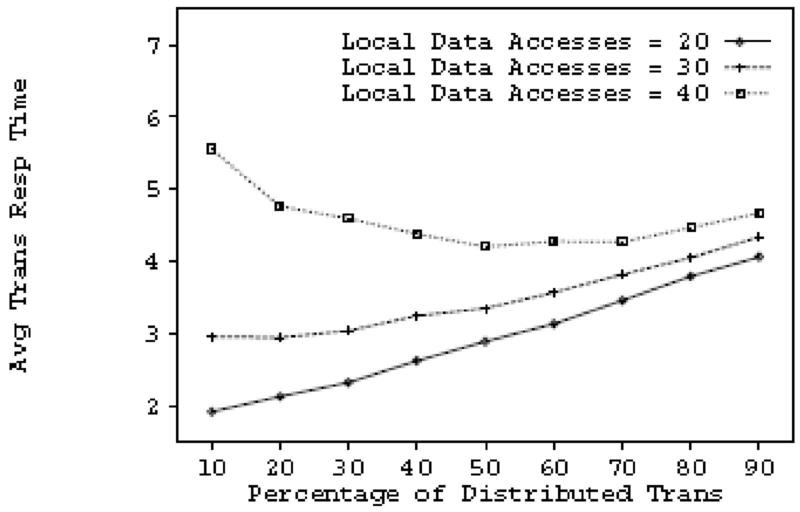
Local and Distributed Transactions

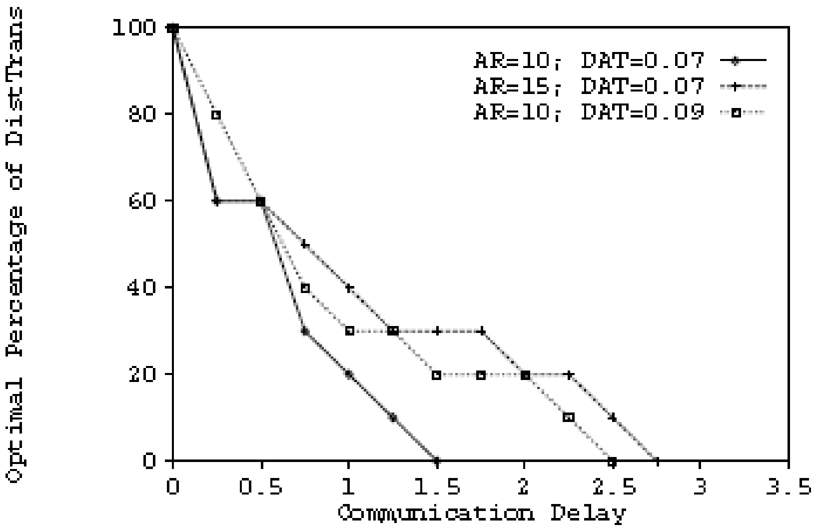


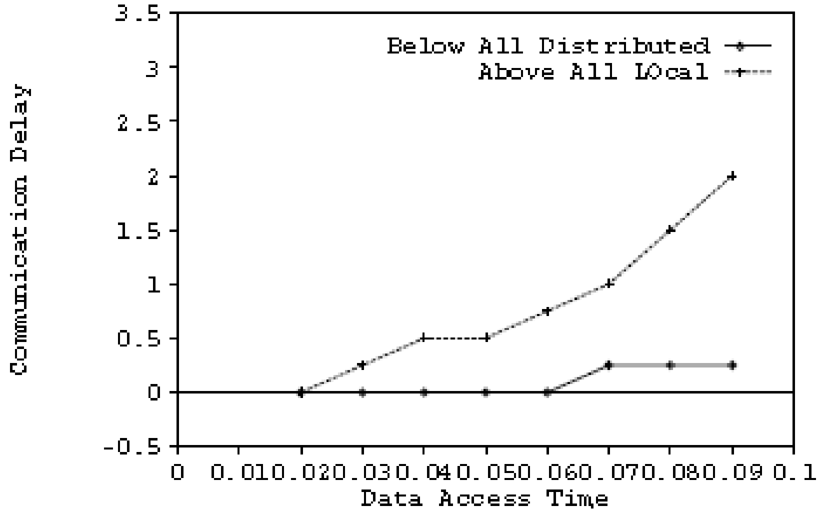
Simulation Model for Lock Manager Contention



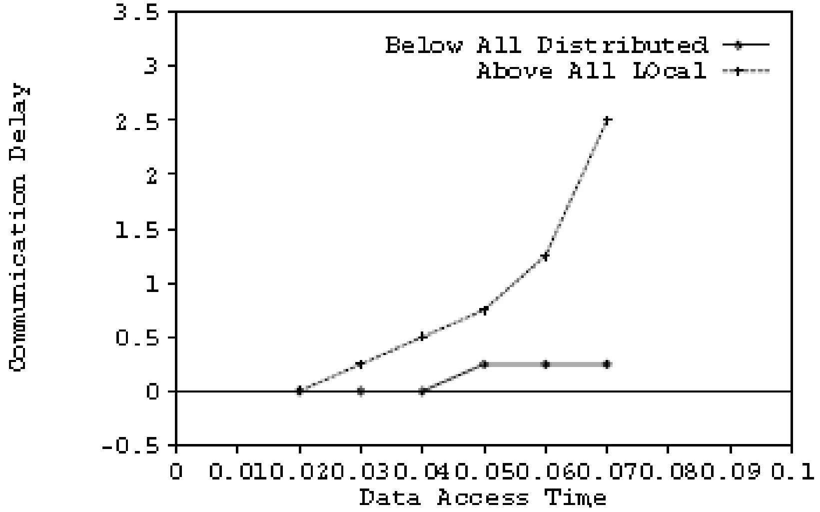
Results







10 transactions/second



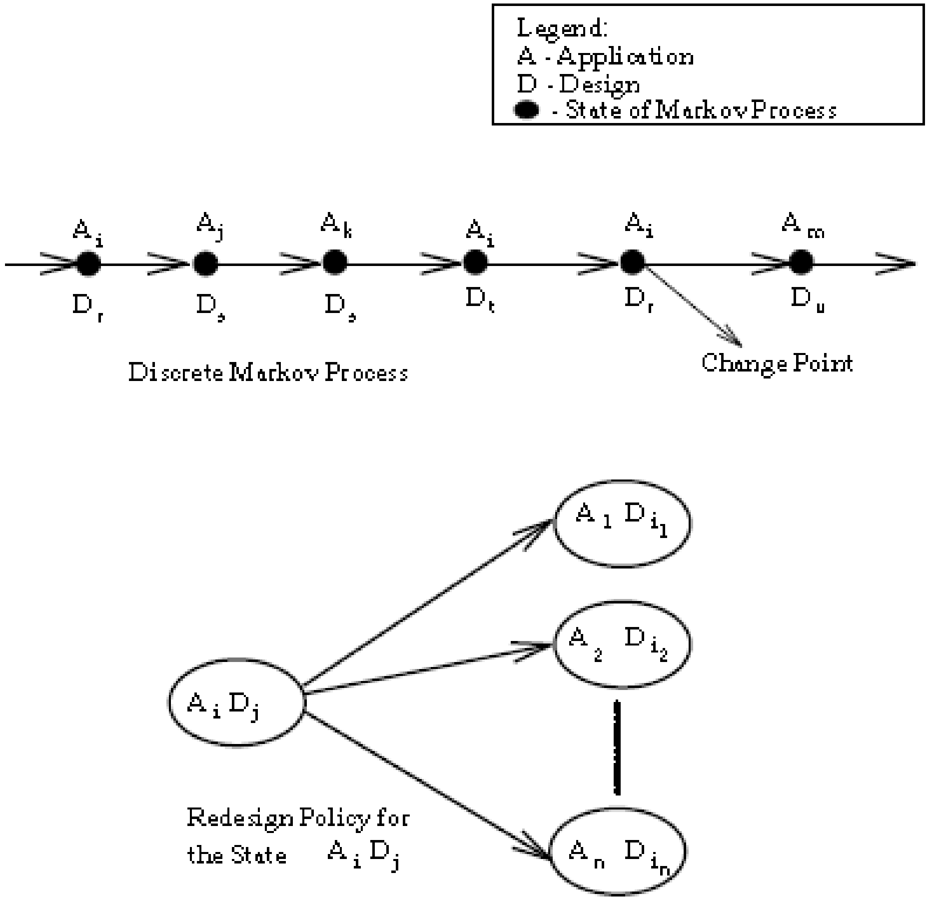
15 Transactions per second

Preventive Redesign

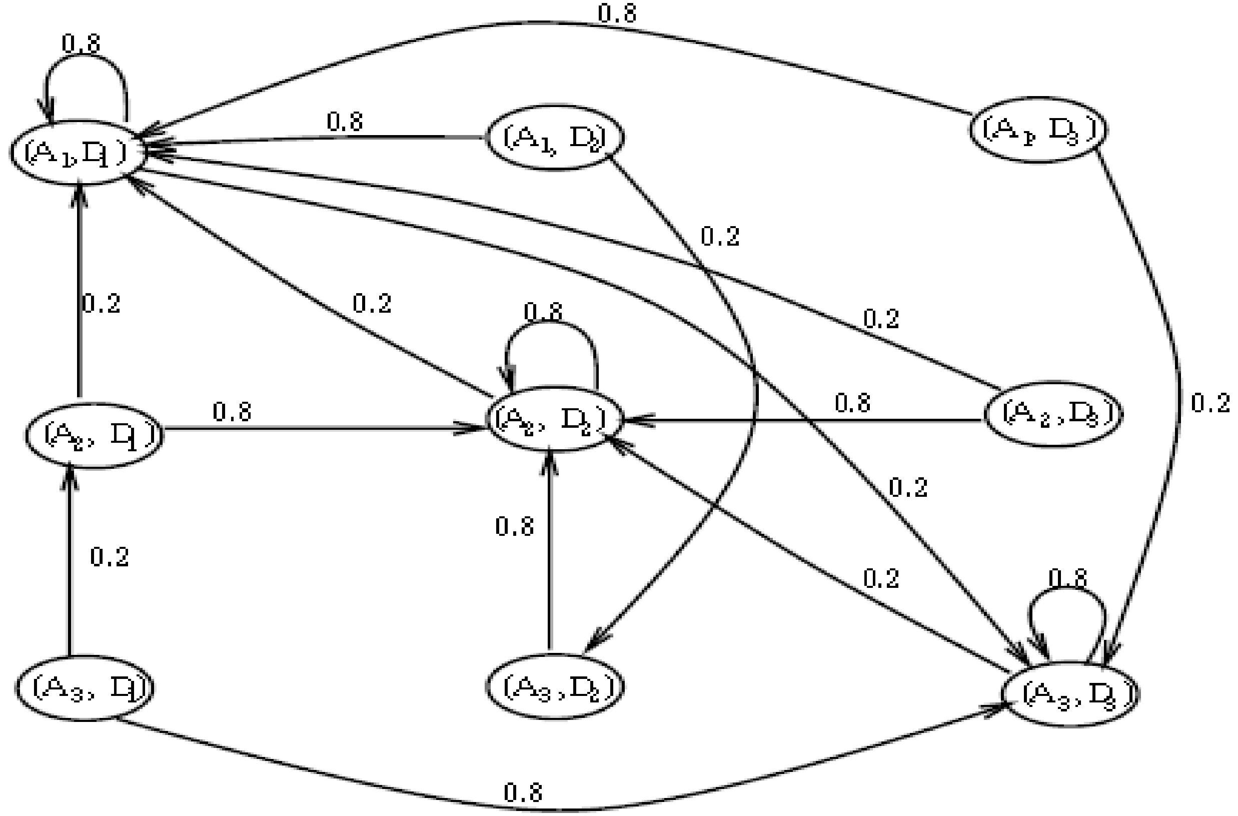
Preventive Redesign Policy

- 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



Optimal Redesign Policy



K. Karlapalem, S. B. Navathe and M. Ammar, Optimal Redesign Policies to Support Dynamic Processing of Applications on a Distributed Relational Database System, Information Systems, Vol. 21, No. 4, pages 353-367, September, 1996.

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

Modern Database Systems Landscape

Apache Impala
CouchDB
Citus Data
Apache Cassandra
MongoDB
Vitess
Amazon Redshift
PostgreSQL
SAP HANA
Databricks
CockroachDB
Azure Cosmos DB
Google BigQuery
Teradata
IBM DB2
DuckDB
Amazon Aurora
SQL Server
IBM Netezza
Oracle
PrestoDB
VoltDB
Yugabyte
AlloyDB
Vitesse
Teradata
SQLLite
Percona
MySQL
FoundationDB
Neo4j
MariaDB
TiDB
Google Bigtable
Snowflake
Apache Hive
Amazon SageMaker
Amazon DynamoDB
Redis

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

Offer fragmentation transparency

- Google Cloud Spanner
- Amazon Aurora
- CockroachDB
- Microsoft SQL Server
- Oracle RAC
- Yugabyte
- Redis
- Apache Ignite
- VoltDB

No/partial fragmentation transparency

- Redis Cluster
  - sharding and partitioning of data is done manually by the user

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

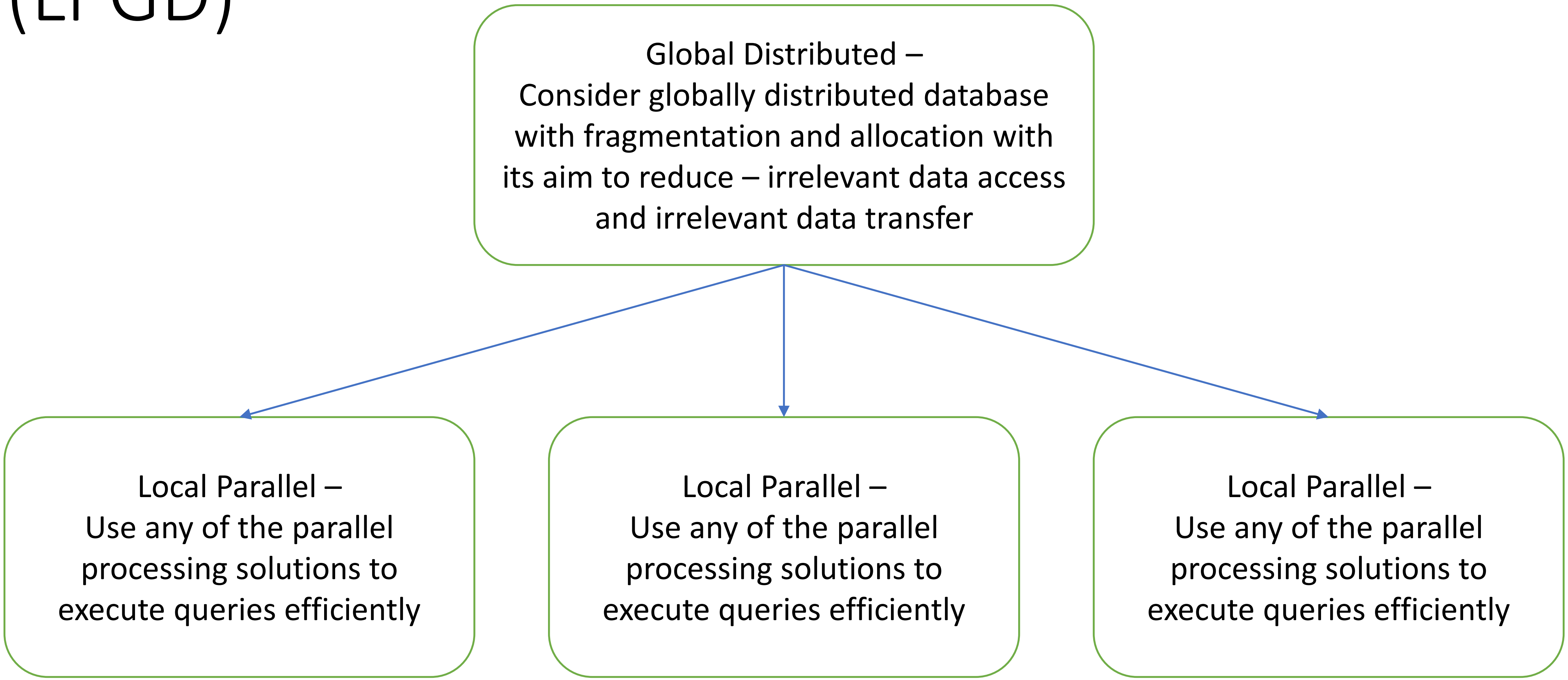
- D1: Resource Sharing Model
  - Shared Everything, Disk, Memory and Nothing
- D2: Is physical storage optimized for querying/updates?
  - Columnar vs Row storage
  - Partitioning/Sharding
- D3: Distribution Transparency
  - Location, Replication, Fragmentation and Transaction
- D4: Storage and Compute 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)



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.

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

1. Kim, W., & Garcia-Molina, H. (1995). Partitioning Large Databases. In Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data (pp. 206-217).
2. Ozsu, M. T., & Valduriez, P. (2011). Principles of Distributed Database Systems (3rd ed.). Springer.
3. Coulouris, G., Dollimore, J., & Kindberg, T. (2011). Distributed Systems: Concepts and Design (5th ed.). Addison-Wesley.
4. Chandra, T. D., & Toueg, S. (1996). Unreliable Failure Detectors for Reliable Distributed Systems. Journal of the ACM, 43(2), 225-267.
5. Gray, J. (1986). The Transaction Concept: Virtues and Limitations. In Jim Gray on distributed databases (pp. 407-414). Morgan Kaufmann.
6. Bernstein, Philip A., and Nathan Goodman. Concurrency control in distributed database systems. ACM Computing Surveys (CSUR) 13, no. 2 (1981): 185-221.
7. Sadoghi, Mohammad, Hans-Arno Jacobsen, and Bettina Kemme. Distributed transaction management: a survey. IEEE Transactions on Parallel and Distributed Systems 25, no. 4 (2014): 1010-1027.
8. G. Alonso, D. Agrawal, A. El Abbadi, and M. Kamath, Transactional consistency and automatic management in an actively replicated storage system, Proceedings of the 1996 ACM SIGMOD international conference on Management of data, 1996, pp. 83–94.
9. J. Gray, R. Lorie, G. Putzolu, and I. Traiger, Granularity of locks and degrees of consistency in a shared database, VLDB Journal, vol. 3, no. 2, pp. 248–261, 1979.
10. D. B. Lomet and M. P. Atkinson, Transactional client-server caches for web applications, Proceedings of the 2005 ACM SIGMOD international conference on Management of data, 2005, pp. 505–516.

References

11. A. Adya, Weak consistency: A generalized theory and optimistic implementations for distributed transactions, Massachusetts Institute of Technology, 1999.
12. J. Gray and A. Reuter, Transaction processing: concepts and techniques. Morgan Kaufmann, 1993.
13. D. J. Abadi, Consistency Tradeoffs in Modern Distributed Database System Design, IEEE Computer Society, pp. 398-409, 2012.
14. P. Bailis, A. Ghodsi, J. M. Hellerstein, I. Stoica, Bolt-on Causal Consistency, Proc. of the 7th ACM European Conference on Computer Systems (EuroSys), pp. 81-94, 2012.
15. S. Das, D. Agrawal, A. El Abbadi, Replication and Consistency Management in Distributed Databases, Springer, 2013.
16. J. Kleppmann, Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems, O'Reilly Media, 2017
17. Stonebraker, M. (1986). The case for shared nothing. Database engineering, 9-22.
18. DeWitt, D. J., & Gray, J. (1992). Parallel database systems: the future of high performance database systems. Communications of the ACM, 35(6), 85-98.
19. Agrawal, R., & Carey, M. (1993). Distributed database systems. ACM Computing Surveys (CSUR), 25(2), 171-236.
20. Kemper, A., & Neumann, T. (2011). Hyper: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. Proceedings of the 2011 ACM SIGMOD international conference on Management of data, 305-316.
21. Ailamaki, A., DeWitt, D. J., & Hill, M. D. (2001). DBMSs on a modern processor: Where does time go?. Proceedings 27th International Conference on Very Large Data Bases, 266-275.

References

22. Stonebraker, M., Madden, S., Abadi, D. J., Harizopoulos, S., Hachem, N., & Helland, P. (2007). The end of an architectural era (it's time for a complete rewrite). Proceedings of the 33rd international conference on Very large data bases, 1150-1160.
23. Dias, M. B., & Silva, V. T. (2016). A survey on distributed database systems. Journal of Systems and Software, 122, 262-272.
24. Stonebraker, M., Abadi, D. J., Batkin, A., Chen, X., Cherniack, M., Ferreira, etal. (2005). C-store: a column-oriented DBMS. In Proceedings of the 31st international conference on Very large data bases (pp. 553-564).
25. Abadi, D. J., Madden, S., & Ferreira, M. (2008, April). Integrating compression and execution in column-oriented database systems. In Proceedings of the 2008 ACM SIGMOD international conference on Management of data (pp. 671-682).
26. Graefe, G. (2010). Modern B-tree techniques. Foundations and Trends in Databases, 3(4), 203-402.
27. Apache Arrow (2021). Columnar format. Retrieved from https://arrow.apache.org/docs/format/Columnar.html
28. Apache Parquet (2021). Columnar storage. Retrieved from https://parquet.apache.org/documentation/latest/
29. Stonebraker, M., Brown, P. F., Zhang, D., & Hellerstein, J. M. (2007). Towards database virtualization for database-as-a-service. In Proceedings of the 2007 ACM SIGMOD international conference on Management of data (pp. 119-130).
30. Leis, V., Kemper, A., Neumann, T., & Schubert, S. (2018). The Design and Implementation of Modern Column-Oriented Database Systems. Foundations and Trends in Databases, 8(1-2), 1-259.

References

31. Plattner, Hasso, et al. A common database approach for OLTP and OLAP using an in-memory column database. Proceedings of the 35th SIGMOD international conference on Management of data. 2009.
32. Härder, Theo, and Hasso Plattner. Architecture of the SAP HANA database. IEEE Data Eng. Bull. 33.1 (2010): 16-26.
33. Farooq, Muhammad Bilal, et al. SAP HANA: A review of history, architecture, and performance. Journal of Advanced Research in Dynamical and Control Systems 10.6 (2018): 1836-1847.
34. Halverson, Paul, et al. SAP HANA database backup and recovery using storage snapshot integration. (2016).
35. MemSQL: The No-Limits Database. MemSQL Inc. Technical Whitepaper.
36. Link: https://www.memsql.com/resources/the-no-limits-database/
37. Nishtala, R., Agrawal, P., Chaudhuri, S., Das, S., Kamath, P., Madden, et al, R. (2013, June). The design and implementation of modern column-oriented database systems. In Proceedings of the VLDB Endowment, 6(7), 539-550.
38. Kallman, R., Kimura, H., Natkins, J., Rasin, A., Rosenthal, A., & Zdonik, S. B. (2015, April). H-Store vs. MemSQL: comparing in-memory database technologies. In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (pp. 1547-1558).
39. Kallman, R., Kimura, H., Natkins, J., Rasin, A., Rosenthal, A., & Zdonik, S. (2013, June). H-store: a high-performance, distributed main memory transaction processing system. In Proceedings of the VLDB Endowment, 6(7), 701-712.

References

40. IBM DB2: IBM DB2 with BLU Acceleration: The Always-On Enterprise Data Warehouse: https://www.ibm.com/docs/en/db2/11.5?topic=acceleration-always-enterprise-data-warehouse
41. Oracle Database: Oracle Database 19c: Data Management for the Modern Age: https://www.oracle.com/database/technologies/data-management.html
42. Microsoft SQL Server: Microsoft SQL Server 2019: Transform Data into Insights: https://www.microsoft.com/en-us/sql-server/sql-server-2019
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): https://docs.microsoft.com/en-us/sql/sql-server/failover-clusters/windows/sql-server-failover-cluster-instances-sql-server?view=sql-server-ver15
45. IBM DB2 PureScale: IBM DB2 pureScale: Scale out for cloud and analytics workloads: https://www.ibm.com/docs/en/db2/11.5?topic=scale-scale-out-cloud-analytics
46. Oracle TimesTen In-Memory Database Overview and Architecture by Oracle Corporation, 2015. Available at: https://docs.oracle.com/database/timesten-18.1/TTODB/timesten-in-memory-database-overview.htm
47. IBM solidDB Universal Cache: A Distributed Memory Cache for High Performance Applications by IBM Corporation, 2009. Available at: https://www.ibm.com/support/knowledgecenter/SSEPH2_14.1.0/com.ibm.db2.luw.admin.ha.doc/doc/c0010585.html

References

48. Apache Cassandra Architecture by Apache Cassandra Project, 2021. Available at: https://cassandra.apache.org/doc/latest/architecture/index.html
49. Sharded Cluster Concepts by MongoDB, Inc., 2022. Available at: https://docs.mongodb.com/manual/sharding/
50. Amazon Redshift Architecture by Amazon Web Services, Inc., 2021. Available at: https://docs.aws.amazon.com/redshift/latest/dg/c_redshift-architecture.html
51. Google BigQuery. https://cloud.google.com/bigquery
52. Vertica Systems. Vertica - Analytics Platform: Unified Analytics Warehouse. https://www.vertica.com/
53. Apache HBase. https://hbase.apache.org/
54. CockroachDB: Distributed SQL Database. https://www.cockroachlabs.com/
55. Cloud Spanner - Fully Managed Relational Database - Google Cloud. https://cloud.google.com/spanner
56. Amazon Aurora - MySQL and PostgreSQL Compatible Relational Database Built for the Cloud. https://aws.amazon.com/rds/aurora/
57. Microsoft Azure Cosmos DB: https://azure.microsoft.com/en-us/services/cosmos-db/
58. Distributed SQL Database - YugabyteDB. https://www.yugabyte.com/
59. Redis: https://redislabs.com/
60. Apache Ignite: In-Memory Data Fabric. https://ignite.apache.org/
61. Databricks: Unified Data Analytics Platform - One Cloud Platform for Massive Scale AI and Data Engineering. https://databricks.com/
62. Snowflake: Cloud Data Platform. https://www.snowflake.com/

References

63. TiDB: Huang, E., Huang, W., & Zhu, S. (2017). TiDB: A scalable distributed HTAP database. In Proceedings of the 2017 ACM International Conference on Management of Data (pp. 623-628).
64. Amazon Aurora documentation, https://aws.amazon.com/rds/aurora/
65. FoundationDB documentation, https://apple.github.io/foundationdb/
66. Amazon Simple DB documentation, https://aws.amazon.com/simpledb/
67. VoltDB: Stonebraker, M., Madden, S., & Abadi, D. J. (2013). VoltDB: an open-source relational database system for large-scale, complex systems. ACM SIGMOD Record, 42(1), 22-27.
68. Amazon S3 documentation, https://aws.amazon.com/s3/
69. Redis Cluster documentation, https://redis.io/topics/cluster-tutorial
70. Presto documentation, https://prestodb.io/docs/current/
71. Apache Iceberg documentation, https://iceberg.apache.org/
72. Google Cloud Bigtable documentation, https://cloud.google.com/bigtable/docs/
73. AWS Elasticache for Redis: https://aws.amazon.com/elasticache/redis/
74. Microsoft Azure SQL Database: https://azure.microsoft.com/en-us/services/sql-database/
75. OceanBase by Alibaba Cloud: https://www.alibabacloud.com/product/oceanbase
76. BlazingSQL: https://blazingsql.com/
77. MapD/OmniSciDB: https://www.omnisci.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