# Assignment - 2, Problem Statement - 1
# CSCI 5408 - Data Management, Warehousing, Analytics

**Problem Statement:** Summarize the given research papers in ~500 words (1 page) each.

---

**Research Paper [1]:** Analysis of Joins and Semi-joins in Centralized and Distributed Database Queries.

**Central Idea of Discussion:** Comparison of query processing and performance metrics such as Query Cost, Memory used, CPU Cost, Input-Output Cost, Sort Operations, Data Transmission, Total Time and Response Time between join and semi-join queries along with alternative query plans for centralized and distributed database management systems.

**Summary Illustration:** The summary is structured in the format as illustrated [3] in the below diagram.

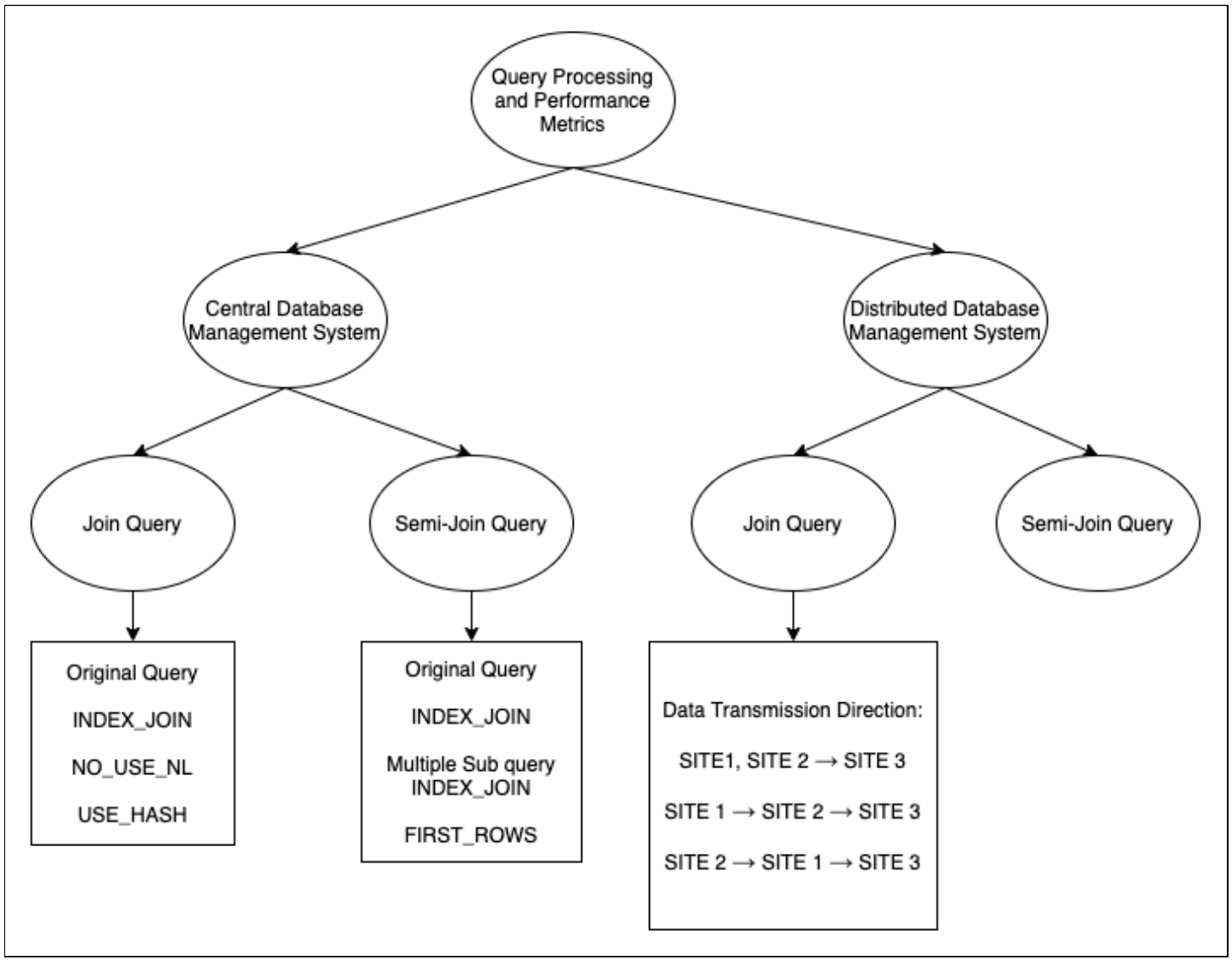


Figure (1) Comparison Chart

**Centralized and Distributed DBMS:** Data is a vital component in database management systems. There are primarily two types, namely, Centralised and Distributed database management systems, which have their advantages and disadvantages. For instance, it's easier to access and perform various operations across different tables in a Centralized DBMS as the data is placed in a central repository. However, in Distributed DBMS, the data is logically interconnected but distributed across different sites connected with a wired or wireless network, thereby increasing data transmission/exchange and processing times. Considering relevant metrics for Centralized and Distributed DBMS, we can understand the significance and compare data access performance between joins and semi-joins.

**Join and Semi-join:** Are important operations in database and relation theory. Joins are special cases of a cartesian product, where the specific conditions are checked against the tables a join is performed on before concatenation. Often, joins and semi-joins can be used in place of each other and have their pros and cons over each other. For instance, semi-joins reduce data exchange but increase processing costs and cannot be used if the sub-query is on an OR branch of the WHERE clause. In contrast, a simple join does not have processing costs overhead. Still, higher data transmission may not be ideal for distributed DBMS.

**Experimental Analysis:** The above Figure (1) represents the various experimental analysis performed on joins and semi-joins in centralized and distributed database systems.

In a **centralized DBMS**, a join operation performed better than a semi-join in every aspect, such as query cost, bytes accessed, IO Cost, and CPU Cost. However, when the alternative query plans of joins and semi-joins are analyzed using the Toad application by Oracle. It is evident that the semi-join can perform better than a join in some instances, but the differences are not significantly high. For example, a join has better logical reads and memory for the Alternate Query Plan II, but the semi-join has better performance in plan cost, sort rows, and scan rows.

A similar analysis is performed in a **Distributed DBMS**. Since DDBMS consists of a cluster of computers coupled with one and another over a communication media, it is crucial to track resource consumption, data exchanged/bytes transferred, total time (Local Processing cost and Communication cost), and response time (Starting to Completion of a query). After comparing the response and total times of semi-join and 3 join types: Transfer both join tables at Site 1 & 2 to a resultant site (Site 1, Site 2 -> Site 3), apply join at site 2, and transmit the to a resultant site (Site 1 -> Site 2 -> Site 3), apply join at site 1, and transmit to a resultant site (Site 2 -> Site 1 -> Site 3), it is clear that the amount of data transmission across sites is the least in semi-join.

The **conclusion** would be subjective, based on the metrics that are given importance; for instance, semi-joins could be a better choice if data transmission is a crucial metric for a DDBMS. Similarly, if CPU and processing costs are important in a centralized DBMS, joins would be better.

**Did you find any topic of interest in this paper?**
The comparison with alternate query paths to check the performance of semi-join and joins is interesting; often, I have measured the performance of a query for different indexes and with the use of subqueries. Furthermore, I further validated the alternate query plans on another database to understand the differences better.

**Note:** The above summary of the research paper is ~550 words or 1 page and presented in two pages to accommodate Figure (1).

**Research Paper [2]:** A Survey on Distributed Deadlock and Distributed Algorithms to Detect and Resolve Deadlock.

**Central Idea of Discussion:** Comparision of advantages and disadvantages of various Deadlock detection and resolution algorithms in a Distributed Database System.

**Summary Illustration:** The summary is structured in the format as illustrated [3] in the below diagram.

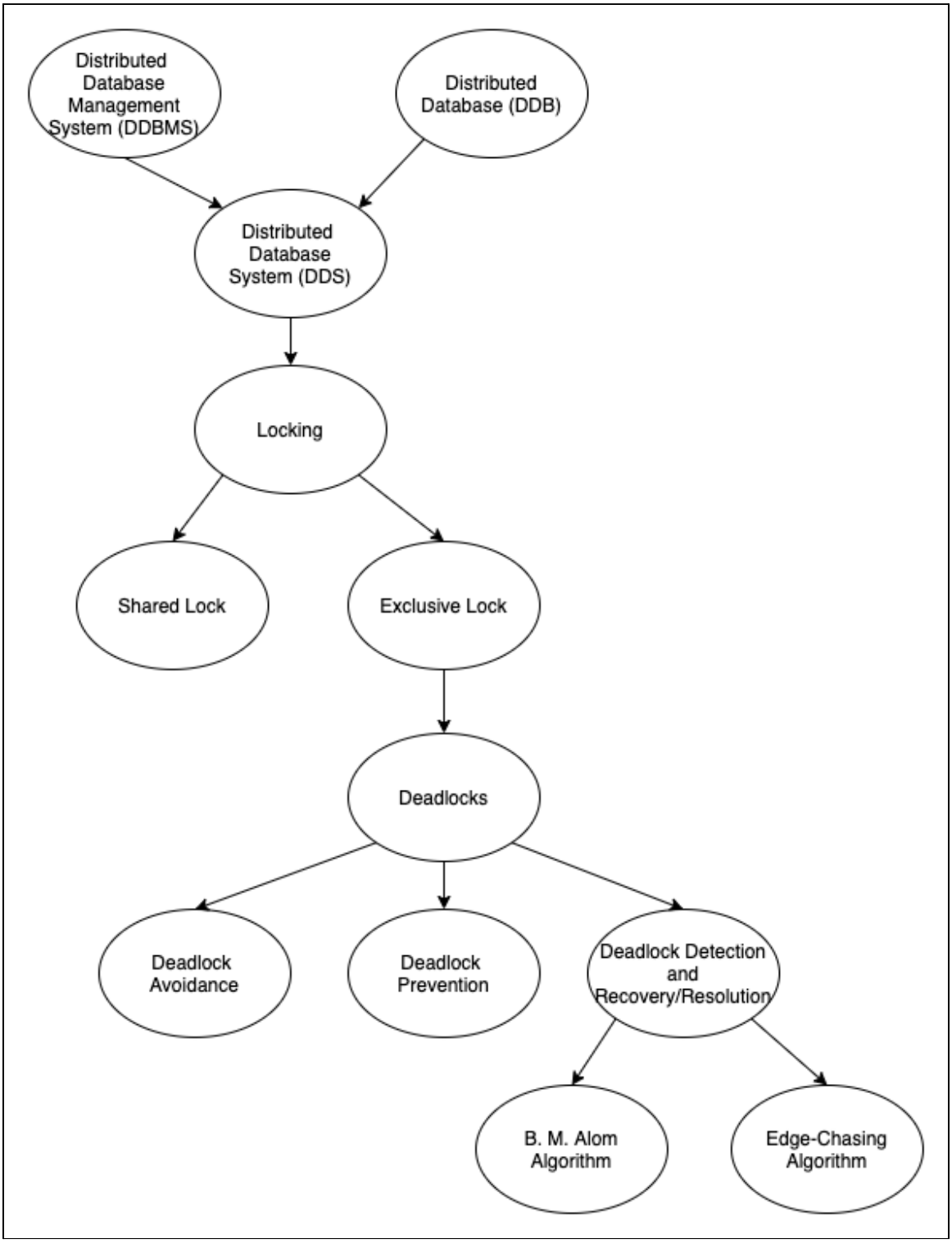


Figure (2)

**DDB, DDBMS, and DDS:** A Distributed Database (DDB) consists of two or more computers that contain logically interconnected data connected over a communication media. The application that manages distributed database(s) is called a Distributed Database Management System (DDBMS). The combination of DDB and DDBMS is called a Distributed Database System (DDS).

**Shared Locks and Exclusive Locks:** Shared lock is used only for read operations, where N number of transactions share a lock to read the same data item(s). On the other hand, an exclusive lock comes into the picture when the same data item(s) has to be manipulated or read by different transactions. A **deadlock** occurs when two are more transactions are independent of each other for acquiring lock to a data item(s) and can occur only because of an exclusive lock. For example, if a transaction (TA) locks on a data item (DA) and another transaction (TB) locks on a data item (DB), if the TA tries to access DB or if TB tries to access DA, it results in a deadlock.

**Handling Deadlocks:** The probability of deadlocks in transactions is a lot more likely in a Distributed Database System; while there are several techniques to handle deadlocks, such as deadlock avoidance and prevention, they are difficult to implement, has a performance overhead, and are inefficient in a distributed system—Wait-for-graphs (WFG) for commonly used to detect a deadlock by checking for cycles within the graph locally and globally. For example, WFG, along with other techniques such as probe computation, checks the status at local sites and uses probes to detect global deadlocks. Another variation is using directed transaction wait-for-graphs; however, since this requires a central controller, it has an overhead of more messages to process. The most commonly used technique is deadlock detection and recovery. Noteworthy algorithms to consider for Deadlock detection and resolution are B. M. Alom and Edge-Chasing Algorithm.

In **B. M. Alom Algorithm**, whenever deadlock cycles are detected in WFG, it uses two tables, one to store all the transactions in a deadlock and another table to keep the priority of the transactions; the priority here is essential, as the transaction with the least priority would be the first to be killed. The disadvantage of this algorithm is that the deadlock detection is not effective if the priority of transactions changes.
On the other hand, **Edge-Chasing Algorithm**, whenever a transaction is waiting for another transaction (blocked state), a probe message is propagated to all the transactions the blocked transaction depends on. If the probe reaches the initiator, it indicates the presence of a deadlock, very similar to the B. M. Alom Algorithm, the deadlock is resolved by aborting the dependent transaction with the least priority. Alternatively, the timestamp can be used in the place of priority to give importance to older transactions. A disadvantage is not detecting deadlocks if the initiator is not a part of the deadlock cycle.

To **conclude**, the research paper talks about algorithms for deadlock detection, avoidance, prevention and resolution, and different techniques for prioritizing transactions without going into many details.

**Did you find any topic of interest in this paper?**
It is interesting to think of deadlock management in an abstract sense. One can try different algorithms for managing deadlock by keeping it abstract, quantifying the algorithm's performance, and comparing it with each other. However, it can be hard to achieve complete abstraction for the deadlock detection and prevention algorithm.

**Note:** The above summary of the research paper is ~550 words or 1 page and presented in two pages to accommodate Figure (2).

**References:**

[1] M. Sharma and G. Singh, "Analysis of Joins and Semi-joins in Centralized and Distributed Database Queries," 2012 International Conference on Computing Sciences, Phagwara, 2012, pp. 15-20, doi: 10.1109/ICCS.2012.15.

[2] V. Kate, A. Jaiswal and A. Gehlot, "A survey on distributed deadlock and distributed algorithms to detect and resolve deadlock," 2016 Symposium on Colossal Data Analysis and Networking (CDAN), Indore, 2016, pp. 1-6, doi: 10.1109/CDAN.2016.7570873.

[3] Draw.io Flowchart. Accessed on: Oct 20, 2021. [Online]. Available: https://draw.io/