CS 764: Topics in Database Management Systems
Lecture 7: Distribution Query Optimization

Xiangyao Yu
9/29/2021
DISTRIBUTED QUERY PROCESSING IN A RELATIONAL DATA BASE SYSTEM
Robert Spezios
Michael Schroeder
Raymond Wood
Electronics Research Laboratory
College of Engineering
University of California, Berkeley 94720

ABSTRACT: In this paper we present a new algorithm for retrieving and updating
data from a distributed relational data base. Within such a data base, any number of
relations can be distributed over any number of sites. Moreover, any user supplied
distribution criteria can optionally be used to specify what site a tuple belongs to.

The algorithm is an efficient way to process any query by "spreading" the qual-
ification into separate "places" using a few simple heuristics. The cost cri-
teria considered are minimum response time and minimum communications traffic.
In addition, the algorithm can optimize sequentially for two models of a communi-
cation network representing respectively ARPA NET and ETHERNET like networks.
This algorithm is being implemented as part of the IBM3831 database system.

KEYWORDS AND PHRASES: Distributed databases, relational model, distributed
decomposition, communication networks, distribution criteria.

I Introduction

In this paper we are concerned with algorithms for processing data base com-
mands that involve data from multiple machines in a distributed data base
environment. These algorithms are being implemented as part of our work in extend-
ing IBM3831 (ARPA NET, 3832) to manage a distributed database. The products
will be available in 1979 to be used with processing interactions in the data sublanguage, SQL. The specific
data model that we use is discussed in Section II. Some of our initial thoughts on
decomposition issues are presented elsewhere (Spezios, Schroeder, Wood). In this
paper we are concerned with processing interactions in the data sublanguage, SQL. The specific
data model that we use is discussed in Section II. Some of our initial thoughts on
decomposition issues are presented elsewhere (Spezios, Schroeder, Wood). In this
section we are concerned with processing interactions in the data sublanguage, SQL. The specific
data model that we use is discussed in Section II. Some of our initial thoughts on
decomposition issues are presented elsewhere (Spezios, Schroeder, Wood).

We are not concerned here with control of concurrent updates or multiple copies
(DELETE, UPDATE, NOTIFY, ORPHAN). Rather we assume that these are handled by a separate
mechanism or can be integrated into our algorithms.

This paper is organized as follows: in section II we formulate the problem by
including our view of a distributed data base and the interactions to be issued.
Then, in section III we discuss our model for the computer network. In section IV
we describe the algorithm that we have used to implement the decomposition
mechanism. In section V we discuss the decompressions of queries in a
distributed environment. There are two complications concerning updates and aggre-
gations in a distributed data base which are covered in sections V and VI. Lastly, in
section VII we give some conclusions.

Research sponsored by the U.S. Army Research Office Grant 14922-TE-0-2945, and the
Joint Services Electronics Program Contract F44622-74-C-0188.

189
Agenda

Distributed database architecture
Data partitioning
Parallel operators
Distributed query optimization
Distributed Database Architecture

Shared Memory

Example: Multicore shared-memory machine
Scale: Single machine
Distributed Database Architecture

Shared Memory

Shared Disk

Example: Network attached storage (NAS) and storage area network (SAN), some Oracle and IBM database systems

Scale: Cluster with tens of machines
Distributed Database Architecture

Shared Memory

Shared Disk

Shared Nothing

Example: Modern massively parallel databases including Google Spanner, Redshift, CosmosDB, etc.

Scale: Any number of machines
Shared-Nothing Architecture

Advantages:
- High scalability
- High availability
- Good data locality

Challenges:
- Must partition data
- Network overhead
Map tuple $i$ to disk ($i \ mod \ n$)

- **Advantage**: Simplicity, good load balancing
- **Disadvantage**: Hard to identify the partition of a particular record

Round robin
Data Partitioning

Map contiguous attribute ranges to partitions

- **Advantage**: Good locality due to clustering
- **Disadvantage**: May suffer from skewness
Map based on the hash value of tuple attributes

- **Advantage**: Good load balance, low skewness
- **Disadvantage**: Bad locality
Data Partitioning

Generally, the **partitioning function (distribution criteria)** can be any function that maps tuples to partition ID.

For example:
Perform all single-table operations
While (exists next piece of query) {
    Select processing sites and transmit data
    Run query on each site
}
Perform all single-table operations
While (exists next piece of query) {
  Select processing sites and transmit data
  Run query on each site
}

Key Challenge
Join – Single-Site

Solution 1: send all the involved tables to a single site

- **Advantage**: Single-site query execution is a solved problem
- **Disadvantage**: (1) Single site execution can be slow (2) Data may not fit in single site’s memory or disk
Join – Broadcast

Solution 2: Keep one relation partitioned and broadcast the other relation to all sites

- **Advantage**: One relation does not need to move
- **Disadvantage**: Still need to broadcast the other relation to all sites

Query plan:
- Site 1: $R_1$, $S_1$
- Site 2: $R_2$, $S_2$
- Site 3: $R_3$, $S_3$

The query plan involves a broadcast exchange operation $R \bowtie S$. The broadcast involves moving $R$ to all sites and receiving $S$, then performing the join.
Join – Co-partition

Solution 3: Partition both relations using the join key

- **Advantage**: Each site has less data to process
- **Disadvantage**: Both relations are shuffled (if not already partitioned based on join key)
Distributed Join

Single-site
- Preferred when **both relations are small**

Broadcast
- Preferred when **one relation is small**

Co-partition
- Preferred when **both relations are large**
Distributed Query Optimization

\[ \pi_A \bowtie S(B,C) \sigma_{C=10} \pi_C \bowtie T(C,D) \]

\[ \pi_A \bowtie S(B,C) \sigma_{C=10} \pi_C \bowtie T(C,D) \]

\[ \pi_A \bowtie S(B,C) \sigma_{C=10} \pi_C \bowtie T(C,D) \]
Distributed Query Optimization

Extra design complexity
- Which exchange operator to use?
- Which nodes to use to run the operator?
Example

Join order
- (project \Join supply) \Join supplier
- project \Join (supply \Join supplier)
Example

Join order
- (project $\bowtie$ supply) $\bowtie$ supplier
- project $\bowtie$ (supply $\bowtie$ supplier)

Plan 1: Send everything to Site 2
- Network traffic 250 tuples
Example

Join order
- (project $\bowtie$ supply) $\bowtie$ supplier
- project $\bowtie$ (supply $\bowtie$ supplier)

Plan 1: Send everything to Site 2
- Network traffic 250 tuples

Plan 2:
Example

Join order
- (project \(\bowtie\) supply) \(\bowtie\) supplier
- project \(\bowtie\) (supply \(\bowtie\) supplier)

Plan 1: Send everything to Site 2
- Network traffic 250 tuples

Plan 2:
- 1\(^{st}\) join: 50 tuples network traffic (on site 2)
- 2\(^{nd}\) join: depends result of 1\(^{st}\) join
Specialized Parallel Operators

Semi-join

• Example:

SELECT *
FROM T1, T2
WHERE T1.A = T2.C

Distributed Database Architecture

Storage Disaggregation

Storage-as-a-Service

Shared Disk

Shared Nothing

Storage-disaggregation architecture is popular in cloud-native databases
- Features: (1) in-storage computation, (2) high-availability, (3) shared access to storage
- More on this topic in last few lectures
Q/A – Distributed Query Optimization

Details of reduction algorithm?
How is the paper related to modern distributed databases?
How does master-slave failover work?
Fragments being a project of the relation?
Distribute data without distribution logic?
More on updates? (will cover in next few lectures)
Which is the bottleneck, network or compute?
Before Next Lecture

Submit review for

Jim Gray, et al., Granularity of Locks and Degrees of Consistency in a Shared Data Base. Modelling in Data Base Management Systems, 1976