CSE 512: Distributed and Parallel Database Systems

Lecture 1

Instructor: Mohamed Sarwat

What is a database

- A very large, integrated collection of data.
- Models real-world *enterprise.*
  - Entities (e.g., students, courses)
  - Relationships (e.g., Mike is taking CSE 412)

Distributed System

A collection of independent computers that appears to its users as a single coherent system.

Distributed System

- Multiple autonomous components
- Components are not shared by all users
- Software runs in concurrent processes on different processors
- Multiple Points of control
- Multiple Points of failure

Why Distribution ?

Multiple Processors

Multiple Memory Systems

Multiple Storage

Why Distribution

Sharing Information and Services

More System Components means:

- More Availability
- More Reliability
- Better Fault Tolerance
- More Scalability

Availability

If a component goes down, another component is available to perform the task

Scalability

- Adaption of distributed systems to
  - Accommodate more users
  - Accommodate large volume of data (big data)
  - respond faster (reduce application latency)
- Usually done by adding more and/or faster processors.

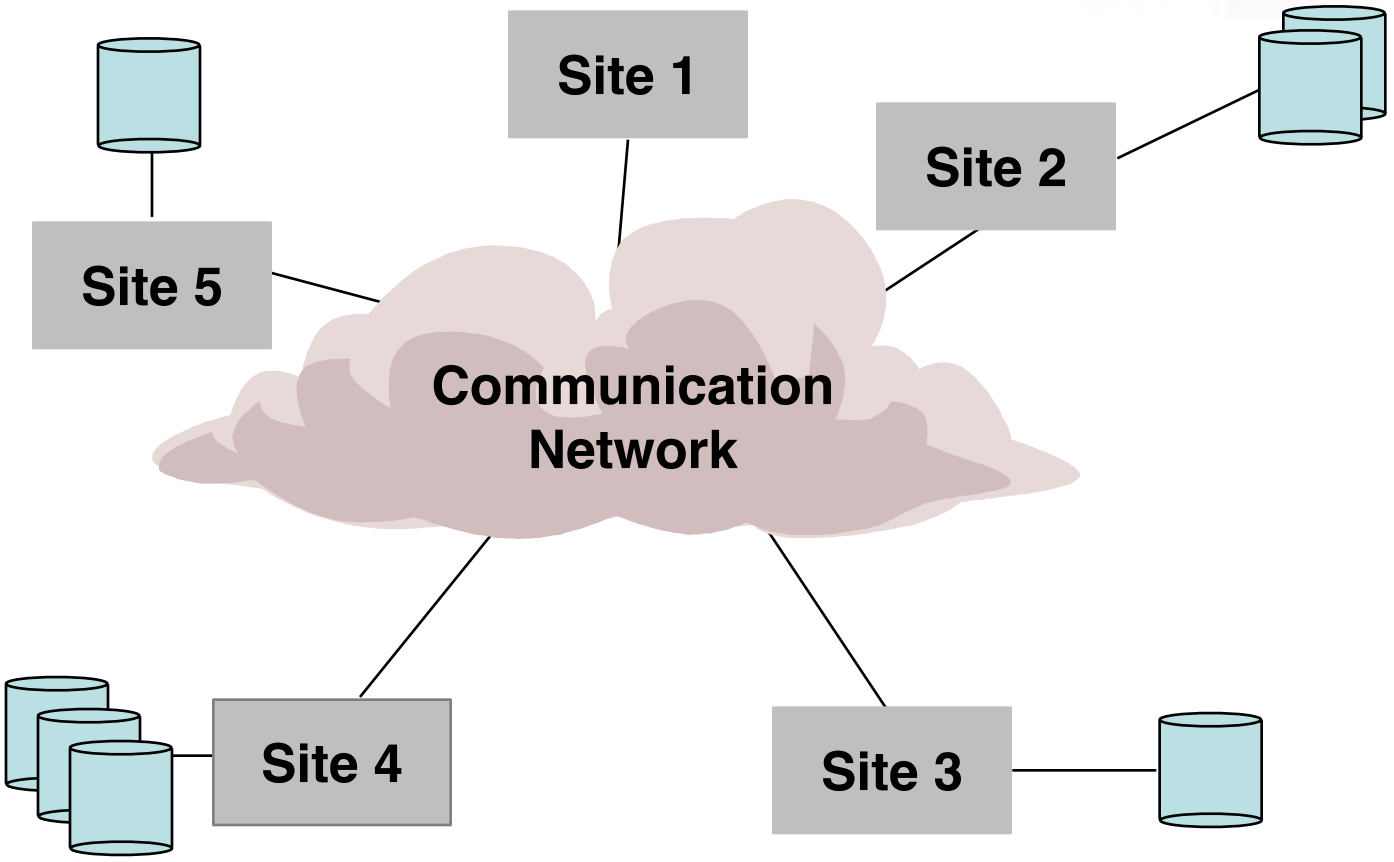
Reliability / Fault Tolerance

- Hardware, software and networks fail!
- Distributed systems must maintain availability even at low levels of hardware/software/network reliability.
- Fault tolerance is achieved by
  - Recovery strategy
  - Redundancy (replication)

Transparency

Distributed systems should be perceived by users and application programmers as a whole rather than as a collection of cooperating components.

Distributed DBMS Environment



What is a Distributed Database System?

A distributed database (DDB) is a collection of multiple, *logically interrelated* databases distributed over a *computer network*.

A distributed database management system (D–DBMS) is the software that manages the DDB and provides an access mechanism that makes this distribution transparent to the users.

Distributed database system (DDBS) = DDB + D–DBMS

Distributed DBMS Promises

- Transparent management of distributed, fragmented, and replicated data
- Improved reliability/availability through distributed transactions
- Improved performance
- Easier and more economical system expansion

Transparency

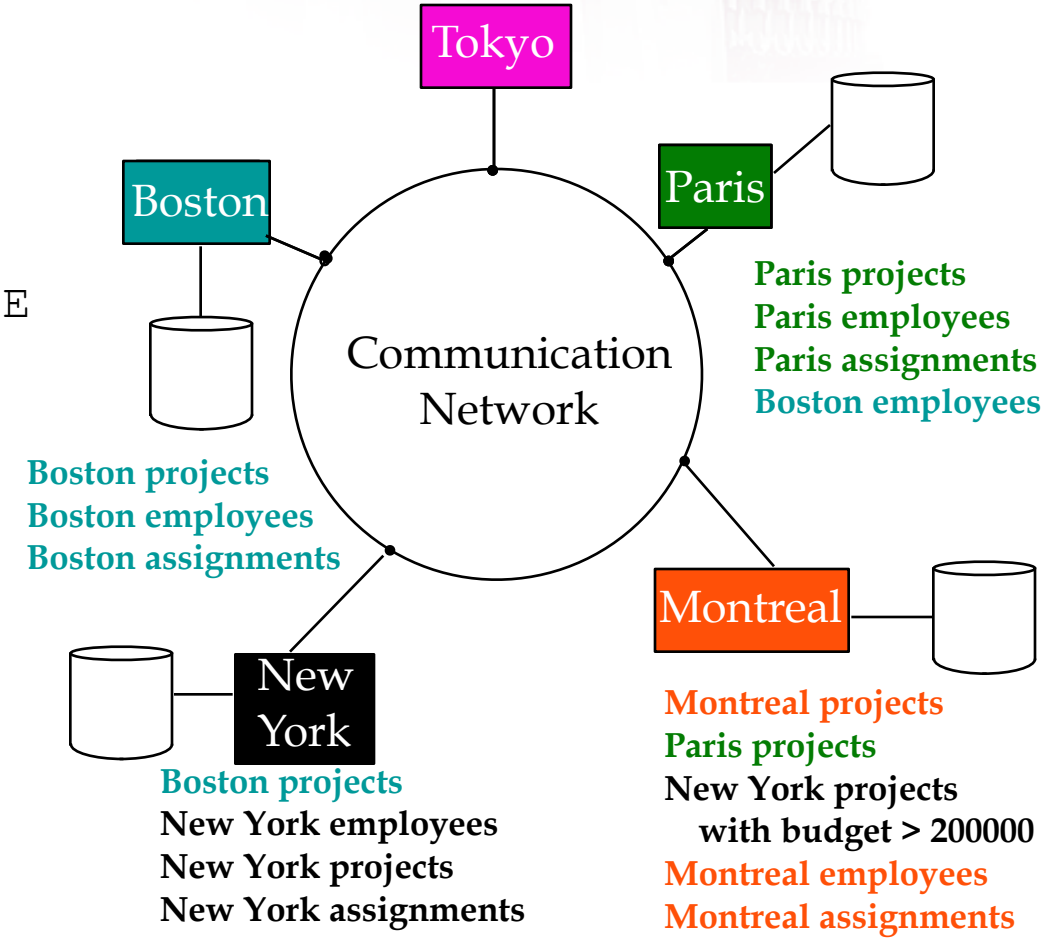
- Transparency is the separation of the higher level semantics of a system from the lower level implementation issues.

- Fundamental issue is to provide **data independence** in the distributed environment

Why Transparency is Key ?

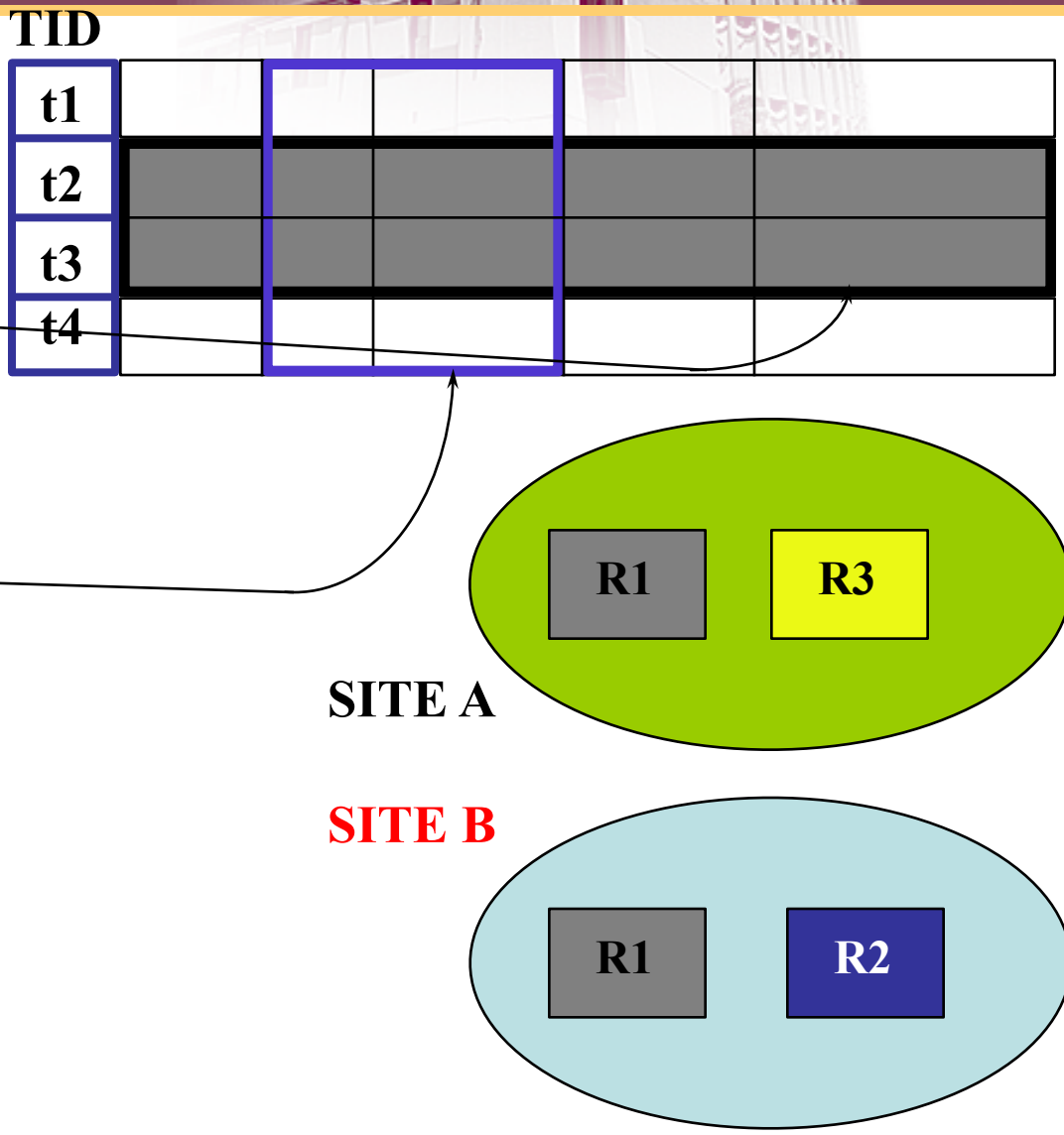
Transparent Access

```
SELECT ENAME,SAL
FROM   EMP,ASG,PAY
WHERE  DUR > 12
AND    EMP.ENO = ASG.ENO
AND    PAY.TITLE = EMP.TITLE
```



Storing Data

- Fragmentation
  - Horizontal:
  - Vertical:
- Replication



Fragmentation

Easiest Way to Do Fragmentation ?

Fragmentation Alternatives – Horizontal

$PROJ_1$: projects with budgets less than $200,000

$PROJ_2$: projects with budgets greater than or equal to $200,000

PROJ

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris
P5	CAD/CAM	500000	Boston

$PROJ_1$

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York

$PROJ_2$

PNO	PNAME	BUDGET	LOC
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris
P5	CAD/CAM	500000	Boston

Fragmentation Alternatives – Vertical

$PROJ_1$: information about project budgets

$PROJ_2$: information about project names and locations

PROJ

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris
P5	CAD/CAM	500000	Boston

$PROJ_1$

PNO	BUDGET
P1	150000
P2	135000
P3	250000
P4	310000
P5	500000

$PROJ_2$

PNO	PNAME	LOC
P1	Instrumentation	Montreal
P2	Database Develop.	New York
P3	CAD/CAM	New York
P4	Maintenance	Paris
P5	CAD/CAM	Boston

Which are good fragmentations?

Example:

$\mathbf{F} = \{ F_1, F_2 \}$

$F_1 = \sigma_{sal<10} E$ $\quad\quad$ $F_2 = \sigma_{sal>20} E$

Which are good fragmentations?

Example:

$\mathbf{F} = \{ F_1, F_2 \}$

$F_1 = \sigma_{sal<10} E$ $\quad\quad$ $F_2 = \sigma_{sal>20} E$

- Problem: Some tuples lost!

Which are good fragmentations?

Second example:

$\mathbf{F} = \{ F3, F4 \}$

$F3 = \sigma_{sal<10\ E}$ $\quad F4 = \sigma_{sal>5\ E}$

Which are good fragmentations?

Second example:

$\mathbf{F} = \{ F3, F4 \}$

$F3 = \sigma_{sal<10\ E}$ $F4 = \sigma_{sal>5\ E}$

- Tuples with $5 < sal < 10$ are duplicated...

$\Rightarrow$ Prefer to deal with replication explicitly

Example: $\mathbf{F} = \{ F_5, F_6, F_7 \}$

$F_5 = \sigma_{sal \leq 5} E$ $F_6 = \sigma_{5 < sal < 10} E$

$F_7 = \sigma_{sal \geq 10} E$

☞ Then replicate F_6 if convenient

(part of allocation problem)

Desired Properties for Fragmentation

- Completeness
  - Decomposition of relation R into fragments R_1, R_2, ..., R_n is complete if and only if each data item in R can also be found in some R_i

Desired Properties for Fragmentation

- Reconstruction
  - If relation R is decomposed into fragments R_1, R_2, ..., R_n, then there should exist some relational operator ∇ such that

$$R = \nabla_{1 \le i \le n} R_i$$

Desired Properties for Fragmentation

- Disjointness
  - If relation R is decomposed into fragments R_1, R_2, ..., R_n, and data item d_i is in R_j, then d_i should not be in any other fragment R_k ($k \neq j$).

Activity

PROJ

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris
P5	CAD/CAM	500000	Boston

Give an example of a fragmentation that does not satisfy completeness

Give an example of a fragmentation that does not satisfy reconstruction

Give an example of a fragmentation that does not satisfy disjointness

Why Replication ?

When does it become challenging?

Why Replication?

- Gives increased availability.
- Faster query evaluation.
- Updates are Challenging

Potentially Improved Performance

- Proximity of data to its points of use
  - Requires some support for fragmentation and replication