

ECE563 Seminar

Transaction Processing

Anil Singhar

Date: 5th March 2007

Outline

1. Definitions
2. Fundamental Issues (Atomicity & Conflict Detection)
3. Commit protocols (2-Phase, 3-Phase)
4. TP Architectures (Shared Nothing, Shared Disks, Shared Everything)
5. Application areas
6. Summary

1. Definitions

What is a Transaction?

An ***inseparable*** list of operations which must be executed ***either in its entirety or not at all*** – (i.e. **Atomic**).

e.g.

begin transaction

{

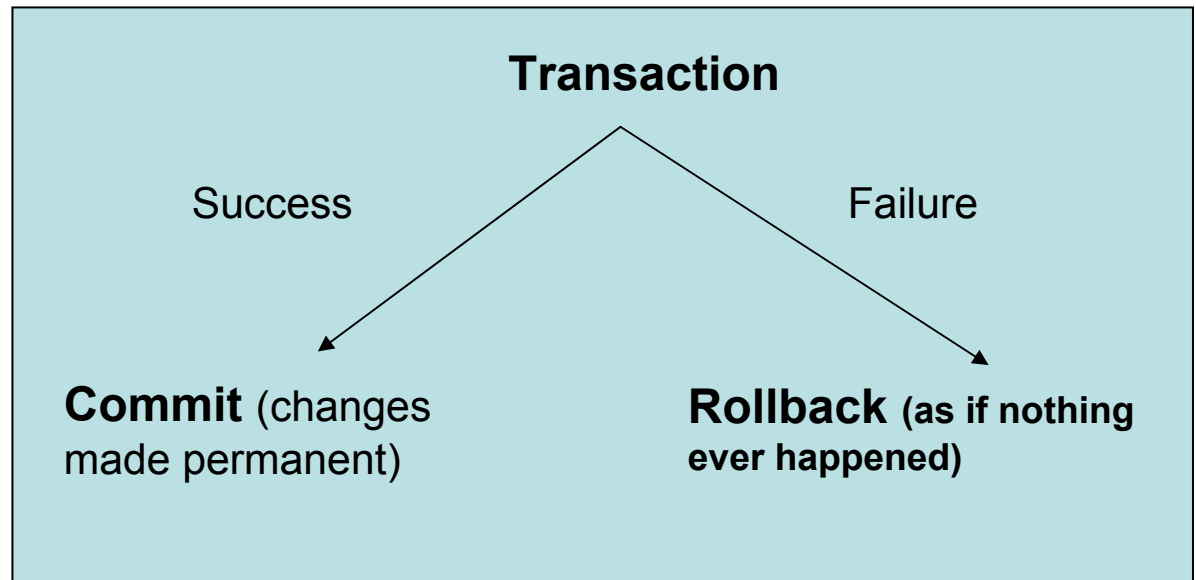
debit checking account

credit savings account

update history log

commit transaction

}



Imagine a situation where an independent thread of execution that monitors account balance is allowed to intervene and take some action.

Why do we need transactions?

-To maintain data integrity and guarantee consistency of distributed data

2a. Atomicity

- Characterizes a Transaction
- Traditional Solutions (**e.g. Critical Sections**)
 - Conservative / Pessimistic
 - Because they prevent concurrent execution of the atomic block (even when a different thread is not accessing the same data in the critical section)

So how much data a Critical section should lock?

- A trade off
- a matter of granularity: smaller the granularity; better we can exploit parallelism
- But smaller the granularity, higher is the cost of conflict detection (Only if one could easily detect the dependencies...)

Optimistic execution – Transactions

- Threads execute in parallel whenever possible – (e.g. threads not accessing the same data inside a CS should execute concurrently)
- A Transaction's implementation should:
 - ensure ordering (serialization) wherever necessary
 - while detecting concurrency efficiently
- So “***Conflict Detection***” is at the heart of an efficient implementation of Transaction.

2b. Conflict Detection – Static (Compiler)

(Conflict Detection ~ Data dependence Analysis)

Advantages:

- + Depends on ability of the Compiler to detect that threads access disjoint data
- + Compiler may remove conflict free code essentially reducing granularity and exposing more parallelism
- + Highly efficient since they avoid run-time overhead

Disadvantages:

- Program should be compile-time analyzable (e.g. programs using pointers to manipulate data are not so)
- Conflicts between transactions and among transactions and other program sections must be considered

2b. Conflict Detection – Dynamic (Hardware)

(Transactions perform conflict detection, rollback and commit in hardware)

Previous technique implemented in hardware

Advantages:

+ Substantially faster than software

Disadvantages:

- Less flexible - obviously
- Rollback involves wasted effort attempting the transaction
- Size of the buffer is a big overhead; while small hardware buffers enable faster conflict detection, they may severely limit the size of a transaction that can be executed – if the buffer fills up during transaction, parallel execution stalls.

2b. Conflict Detection

(Compiler-Assisted Runtime - Part software part hardware)

- For threads not provably conflict free, the compiler still can assist by ***narrowing*** the set of addresses (***memory references***) that may conflict
- This “conflict set” – that need to be monitored is substantially smaller
- At the end of the transaction follow the usual mechanism of commit or rollback

Advantages:

- Eases the burden on the hardware – so makes it faster and less complex
- Less memory requirement for the buffer
- Most dynamically scheduled processors already have the ability to roll back

Disadvantages:

- Interpreters may perform this redirection on the fly evidently there is significant overhead associated with the software implementation.

3. Commit Protocols

- Commit is an important phase of a Transaction
- Choice of a commit protocol is an important design decision
- Commit Protocols are used to design systems that exhibit a well defined behavior in the event of a failure. These systems may or may not perform the specified function during failures, but they may facilitate actions suitable for recovery.

Some classical techniques:

- Two Phase Commit (a Blocking protocol – but guarantees global atomicity)
- Three-Phase commit (a non-Blocking protocol)

Both these techniques designate one node as Master (or coordinator) and all others as cohorts in group of networked nodes.

Assumptions in Commit Protocols

1. Each node has some stable storage with write-ahead log (stores re-do and undo information)
2. No node crashes forever & no more than one node crashes at a time
3. The data in the write-ahead log is never lost or corrupted in a crash
4. Any two nodes can communicate with each other.

2-PC vs. 3-PC

- 3-PC is non-blocking (unlike 2-PC)
- 3-PC places an upper bound on the amount of time required before a transaction either commits or aborts.
- This property ensures that if a given transaction is attempting to commit via 3PC and holds some resource locks, it will release the locks after the timeout.

3a. 2-Phase Commit Protocol

Phase 1. Commit-request

- coordinator sends a ***query to commit*** message to all cohorts.
- cohorts execute the transaction up to the point where they will be asked to commit. They each write an entry to their *undo log* and *redo log*.
- If (transaction succeeds) Each cohort replies with an **agreement** message,
- Else or an **abort** message if the transaction failed.
- The coordinator waits until it has a message from each cohort.

Phase 2. Commit

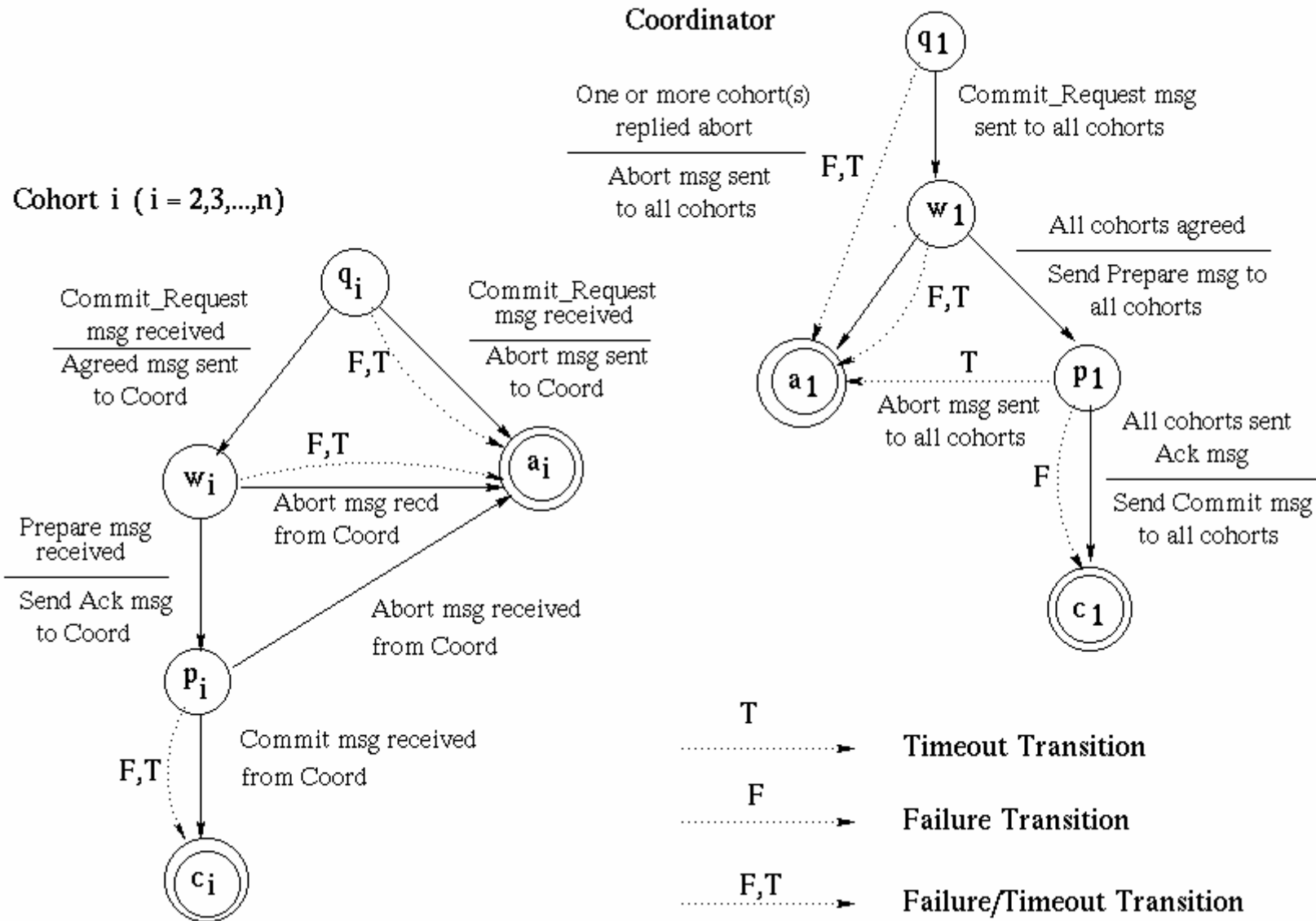
If (Coordinator received an agreement message from *all* cohorts) - **Success**

- Coordinator sends **commit** message to all cohorts.
- Cohorts complete operation & release all the locks and resources held during the transaction.
- Cohorts send an **acknowledgement** to the coordinator.
- Coordinator completes the transaction when all the acknowledgements have been received.

Else (at least one cohort sent an abort message) - **Failure**

- Coordinator sends a **rollback** message to all the cohorts.
- Cohorts undo the transaction using the undo log & release the resources and locks held during the transaction.
- Cohorts send **acknowledgement** to the coordinator.
- Coordinator completes the transaction when all acknowledgements have been received.

3b. 3-Phase Commit Protocol



At time $t = 0$: (i.e. before the commit protocol begins) All the sites are in state q . If the coordinator fails while in state q , all the **cohorts** perform the **timeout transition**, thus aborting the transaction. Upon recovery, the **coordinator** performs the **failure transition**.

Phase I (Send Commit Requests)

- Coordinator in state **w1**, and each cohort in one of 3 possible states:
 1. state **a** (i.e. the cohort has already sent an abort message to the coordinator)
 2. state **w** (i.e. the cohort has received the **Commit_Request** message)
 3. state **q**

If a cohort fails, the coordinator times out waiting for the agreed message from the failed cohort. In this case, the coordinator aborts the transaction and sends abort messages to all the cohorts.

Phase II (send prepare messages)

- If (Everyone **agreed** in phase 1)
 - Coordinator sends a **Prepare** message to all the cohorts
 - If (failed while sending Prepare message)
 - Abort the transaction upon recovery. Cohorts will timeout waiting for **Prepare** message eventually.
 - Else
 - Coordinator sends Abort message to all the cohorts.
- Cohort receives **Prepare** message.
 - Cohort sends an **Acknowledge** message to the coordinator.

Phase III

- If (Everyone acknowledged in phase 2)
 - Coordinator sends a **Commit** message to all the cohorts.
 - If (Failed while sending Commit messages)
 - Commit the transaction upon recovery. The cohorts time out waiting for the Commit message. They commit the transaction according to the timeout transition from state pi.
 - else (i.e. at least one cohort failed in acknowledging)
 - Coordinator times-out, and aborts. Sends abort messages to all cohorts. Failed cohort, after recovery will abort the transaction.
- Cohort receives **Commit** message
 - Commits the transaction.

Disadvantage of 2-PC:

- A cohort can block indefinitely
- If the coordinator fails permanently, some cohorts will never resolve their transactions. This has the effect that resources are tied up forever.
- Works under rather strong assumptions. In order for the scheme to work reliably, both the coordinator and the participating resource managers independently must be able to guarantee proper completion, including any necessary restart/redo operations.

Disadvantage of 3-PC:

- Cannot recover in the event the network is segmented in any manner (e.g. if the network of nodes were to be separated into two equal halves, each half would continue on its own)

Relational Database Basics

- Consists of **Relations** (*files*)
- *Relations* in turn contain **tuples** (*records*)
- Each *tuple* is formed out of a template of **attributes** (*fields*)

- *Relations* – created, updated and queried by SQL statements
e.g. SELECT-PROJECT : a simple & common operation (also called Scan) that produces a row-column subset of a relational table by applying some predicate P to each tuple t
SELECT telephone_number / the output attribute(s) */*
FROM telephone_book / the input relation */*
WHERE last_name = j` Smithj` ; / the predicate */*

- *The Output of a query can be fed into a new query or can act as operands to another relational operator, can be displayed on a terminal, and so on... The uniformity of the data and operators allow the queries to be arbitrarily composed into dataflow graphs.*

4a. TP Architecture (Shared Nothing)

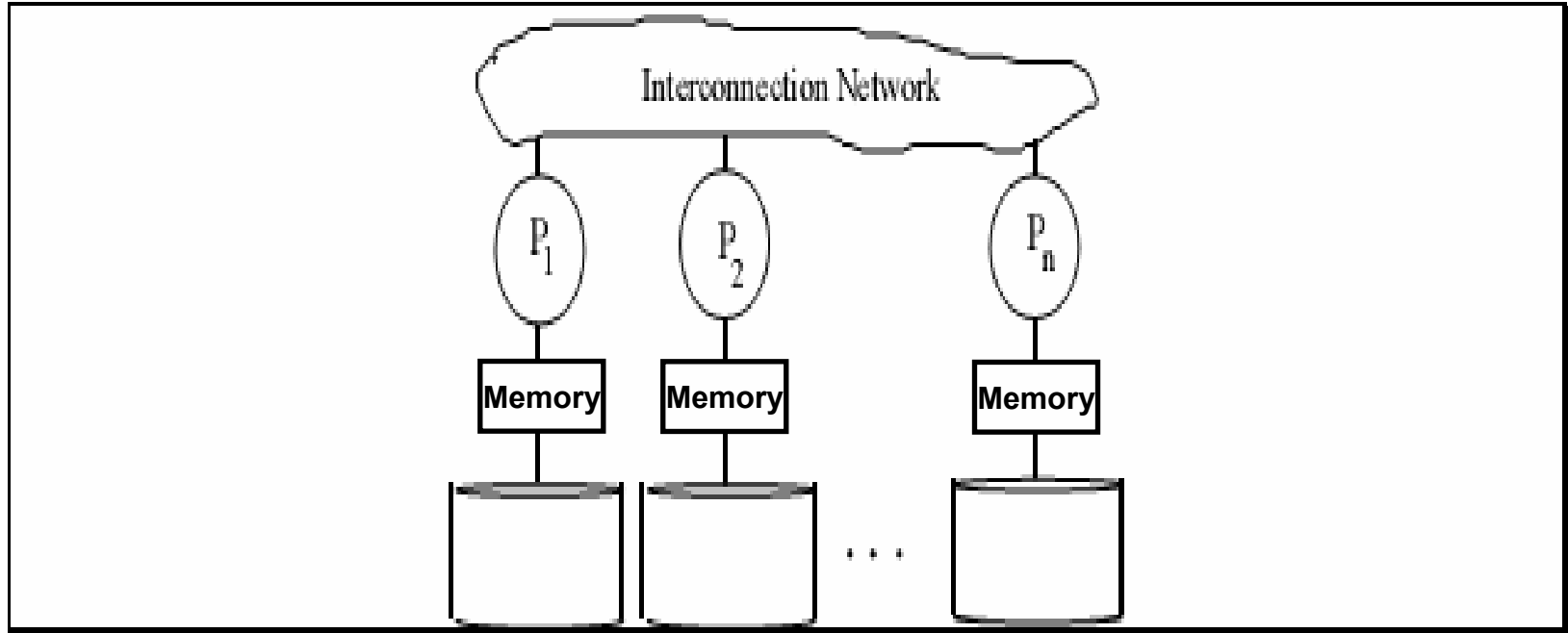


Figure 3. The basic shared-nothing design. Each processor has a private memory and one or more disks. Processors communicate via a high-speed interconnect network. Teradata, Tandem, nCUBE, and the newer VAXclusters typify this design.

Shared Nothing Architecture

(~Message Passing Architecture)

(in context of Database Applications, “**Scalability**” is a Key requirement)

Advantages:

- Minimizes interference by minimizing resource sharing
- Doesn't need a very high speed network
- Raw data/disk access are done locally – only filtered data passed to clients
- Moves only questions and answers through the networks in case of Databases
- Highly scalable - up to 1000s of processors that do not interfere with one another.
- Achieve linear Speedup & scale-up on complex queries and on-line transaction processing workloads

Disadvantages:

- Load imbalance
- If the application is not well parallelizable, or if various attributes of a record is distributed over several systems (e.g a rather complex query which is formed by cascading several simpler queries – where out put of one query being fed to the input of another) it may not be well suited for a shared nothing architecture

4b. TP Architectures

(Shared Memory (Shared Everything) and Shared Disks)

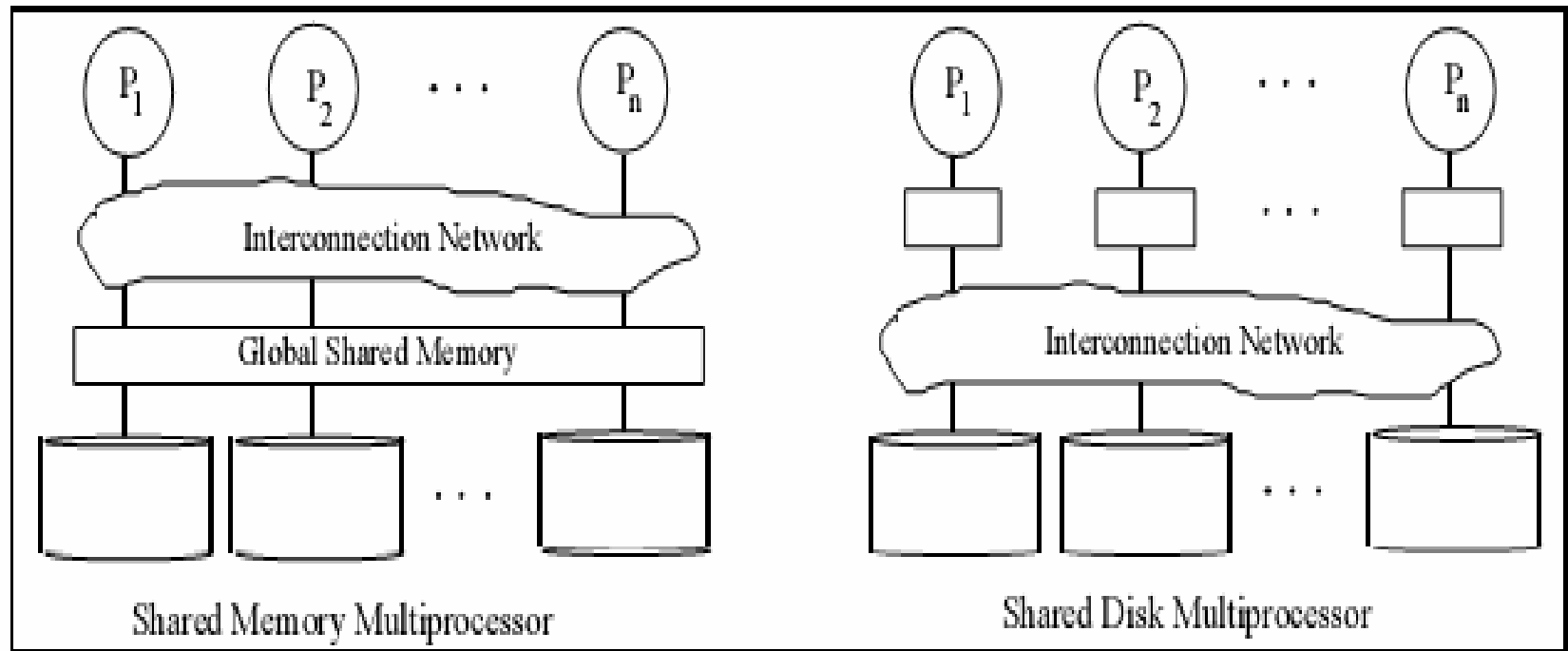


Figure 4. The shared-memory and shared-disk designs. A shared-memory multi-processor connects all processors to a globally shared memory. Multi-processor IBM/370, VAX, and Sequent computers are typical examples of shared-memory designs. Shared-disk systems give each processor a private memory, but all the processors can directly address all the disks. Digital's VAXcluster and IBM's Sysplex typify this design.

Advantages:

- Probably makes the life of a programmer easier as in case of general parallel programming.
- *Not so long ago, people preferred these architectures over shared nothing ones, because*
- *Memory was expensive – hence it was wise to sharing it*
- *Communication overhead was a relatively large.*
- *But not anymore...*

Disadvantages:

- The bottom line is...the more we share resources, the more we are likely to interfere

Application Areas

Any database driven application one can think of such as Banking, E-Commerce, Research, Airline ticket Booking and so on..

Where/How *parallel transaction processing* could make a difference?

Anywhere, provided the process can be expressed in terms of transactions.

Summary

1. Transaction processing models may be suited to certain application areas provided the computation is amenable to transactions.
2. Conflict Detection ability, Atomicity, Commit Protocols are some key aspects that constitute a reliable transaction processing model
3. What architecture should be chosen to host a transaction based system depends on many factors, such as nature of distributed data, frequency (*think of common case !*) and complexity of queries. E.g. shared nothing architectures are being increasingly proven to be best suited for data base applications.

Questions ?

Thank You !