Coordination Avoidance in Distributed Databases

Seminar 1

Ziganurova Liliia

07.02.2018

National Research University Higher School of Economics

Outline

Introduction

Introduction

The difficulties of operations over distributed computer networks are:

- delays;
- partial failures;
- uncertainty about global state, etc.

In many applications the difficulties of distributed systems design are relegated to a database tier. **The goal of the work is to study the design of database systems that provide coordination-free execution, which implies availability, low latency and scalability**.

Introduction

Objectives:

1. Define a rule for determining wether a coordination-free implementation of a given safety properties exists (**invariant confluence**).
2. Examine a set of common semantic guarantees found in databases today and apply the criterion of invariant confluence on them.
3. Find a coordination-free implementation of invariant confluent semantics.

Reasoning about distributed databases

- Should you and I be able to simultaneously reserve rooms?
- Can you reserve a room while I log in?
- Can you tweet while I change my username?

Definition: We say that two operation must **coordinate** if they cannot execute concurrently on independent copies of the database state.

The classic solution for maintaining invariants is to **serializable** isolation: execute transactions such that the end result is equivalent to some sequential execution.

The problem is that serializable semantics require coordination.

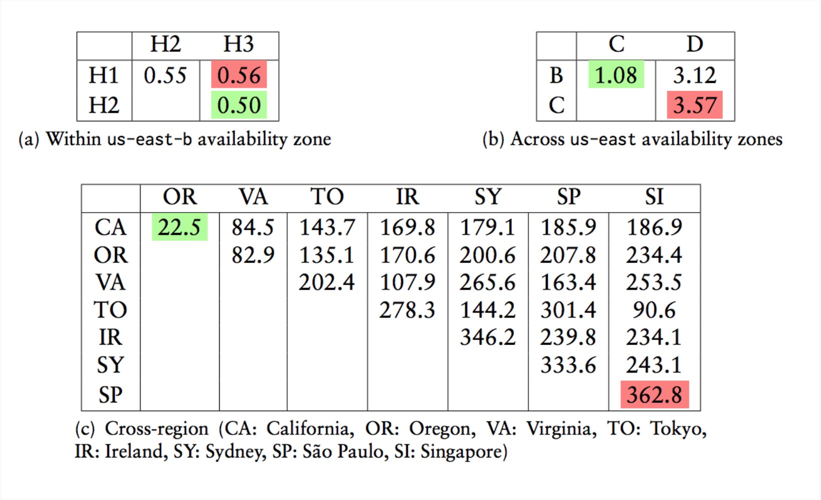
Coordination: Concepts and Costs

Coordination is expensive

1. Low latency
2. Throughput
3. Scalability
4. Availability

Coordination is expensive: Latency

Messages travel is slower that the speed of light due to routing, congestion, and computational overheads.



	H2	H3
H1	0.55	0.56
H2		0.50

(a) Within us-east-b availability zone

	C	D
B	1.08	3.12
C		3.57

(b) Across us-east availability zones

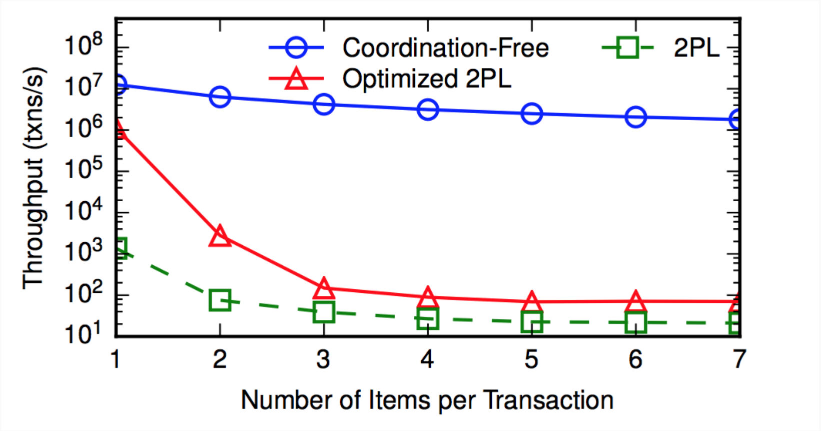
	OR	VA	TO	IR	SY	SP	SI
CA	22.5	84.5	143.7	169.8	179.1	185.9	186.9
OR		82.9	135.1	170.6	200.6	207.8	234.4
VA			202.4	107.9	265.6	163.4	253.5
TO				278.3	144.2	301.4	90.6
IR					346.2	239.8	234.1
SY						333.6	243.1
SP							362.8

(c) Cross-region (CA: California, OR: Oregon, VA: Virginia, TO: Tokyo, IR: Ireland, SY: Sydney, SP: São Paulo, SI: Singapore)

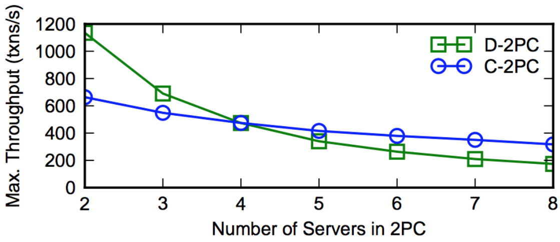
Coordination is expensive: Throughput

Three algorithms were implemented:

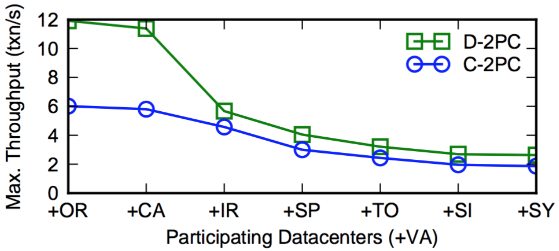
1. Traditional two-phase locking;
2. Optimized two-phase locking;
3. Coordination-free.



Coordination is expensive: Scalability



a.) Maximum serializable transaction throughput over local-area network in [230]



b.) Maximum serializable transaction throughput over wide-area network in [32] with transactions originating from a coordinator in Virginia (VA; OR: Oregon, CA: California, IR: Ireland, SP: São Paulo, TO: Tokyo, SI: Singapore, SY: Sydney)

Coordination is expensive: Availability

Availability: in the presence of communication failures between servers, each client's operations may still proceed, providing "always on" functionality.

A 2011 study of several Microsoft datacenters observed:

- over 13300 network failures with end-user impact;
- 59000 packets lost per failure;
- mean of 40.8 network link failures per day;
- median time to repair 5 minutes (up to 1 week).

System model: database, transaction, replicas

- **Databases**: a set D of unique versions of data items located on an arbitrary set of database servers;
- Users submit request in the form of **transactions** - ordered group of operations on data items.
- Each transaction operates on a **replica** - a set of versions of the items mentioned in the transaction.
- Transaction is a transformation on a replica: $T : \mathcal{D} \rightarrow \mathcal{D}$
- Upon completion, each transaction can commit or abort.
- Upon commit, the replica state is subsequently **merged** into the local database.

Each transaction can modify its replica state without modifying any other concurrently executing transactions' replica state. Replicas provide transactions with partial "snapshot" views of the global database.

System model

System model: invariants

- **Invariants** are needed to determine where a database is valid according to application criteria. Invariants are predicated over databases: $I : \mathcal{D} \to \{true, false\}$ (ex.: unique key, nonzero values).
- A database state is **valid** under an invariant I (or I-valid) if it satisfies the predicate: *A replica state $R \in \mathcal{D}$ is I-valid, iff $I(R) = true$.*
- A system is **globally I-valid** if all replicas always contain I-valid state.

System model: availability, convergence

- **Availability** means that whenever a client executing a transaction T can access servers containing each items in T, the T eventually commits unless: 1) there is an explicit abort operation in T; 2) T violates a declared invariant.
- A system is **convergent** iff for each pair of servers in the absence of new writes or indefinite communication delays, the servers eventually contain the same versions for any items they both store (via merge operator).

System model: coordination-free execution

A system provides **coordination-free execution** for a set of transactions T iff the progress of executing each $t \in T$ in only dependent on t' replica's state (i.e. the versions of the items t reads).

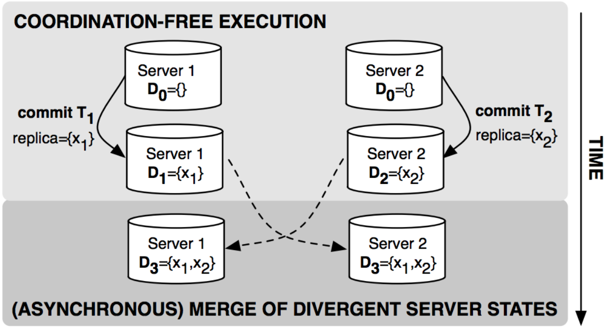
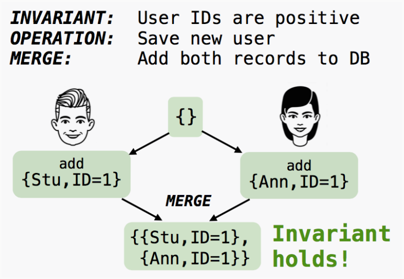


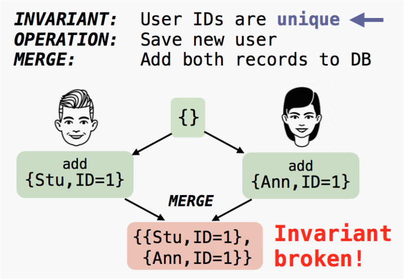
Figure 2.4: An example coordination-free execution of two transactions, T_1 and T_2, on two servers. Each transaction writes to its local replica, then, after commit, the servers asynchronously exchange state and converge to a common state (D_3).

Invariant Confluence: Criteria Defined

KEY QUESTION: Can invariants can be violated by merging independent operations? ANSWER: Invariant confluence test (ICT).

Invariant confluence is a property, that ensured that divergent but *I*-valid database states can be merged into a valid database state.





Invariant Confluence and Coordination

Invariant Confluence: I-T-reachable state

We say that a database D_i is a **I-T-reachable state** if, given an invariant I and a set of transactions T, there exist a partially ordered set of transactions and merge functions that yields D_i, and each intermediate state is I-valid.

All previous states are called **ancestor states**.

Invariant Confluence: Definition

A set of transactions T is **invariant confluent** with respect to invariant I if, for all I-T-reachable states D_i, D_j with a common ancestor state, $D_i \sqcup D_j$ is valid.

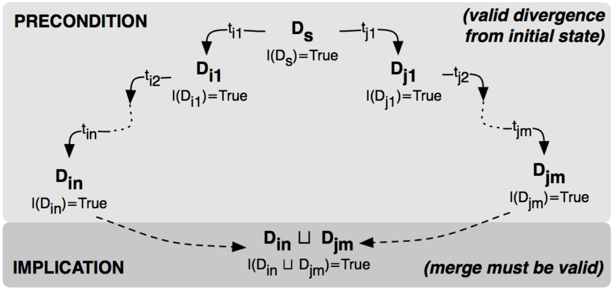


Figure 3.1: A invariant confluent execution illustrated via a diamond diagram. If a set of transactions T is invariant confluent, then all database states reachable by executing and merging transactions in T starting with a common ancestor (D_s) must be mergeable ($\sqcup$) into an I-valid database state.

The main theorem

Theorem 1. A globally I-valid system can execute a set of transactions T with coordination-freedom, transactional availability, and convergence, iff T is invariant confluent with respect to I.

Invariant confluence is a necessary and sufficient condition for invariant-preserving, coordination-free execution.

How to prove. Backward direction: if invariant confluence holds, each replica can check each transaction's modifications locally and replicas can merge independent modifications to guarantee convergence to a valid state. Forward direction: by contradiction: construct a scenario where a system cannot determine whether a non-invariant confluent transaction should commit without violating one of our desired properties (validity, availability, convergent).

Lemma

Lemma 1 Given a set of transactions T and invariants I, a globally I-valid, coordination-free, transactionally available, and convergent system is able to produce any I-T-reachable state S_i.

The proof of the Theorem 1:

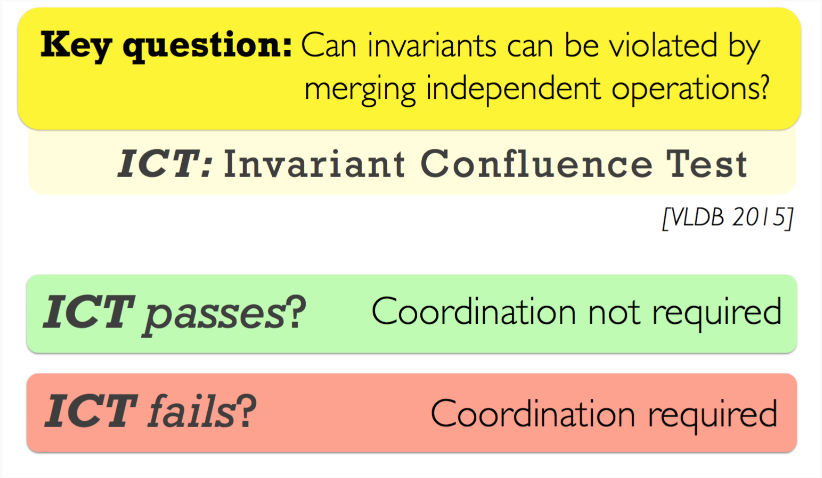
($\Leftarrow$): 1) Each server executes the transactions against a replica of its current state. 2) Checks whether the results are I-valid. 3) If the result is I-valid, the replica commits the transaction, and aborts it otherwise. 4) Servers exchange copies of their local states and merge them. The merge of two I-valid states is valid, because T is invariant confluent.

($\Rightarrow$): *on the blackboard*

Summary

- The difficulties of distributed systems design are relegated to a database tier;
- Coordination is costly;
- It is possible to create a database with coordination-free execution, which implies availability, low latency and scalability;
- Invariant confluence helps to ensure that divergent but valid database states can be merged into a valid database state;
- If all local commit decisions are globally valid then coordination can be avoided.

Home taken message



Key question: Can invariants can be violated by merging independent operations?

ICT: **Invariant Confluence Test**

[VLDB 2015]

ICT passes? Coordination not required

ICT fails? Coordination required