# VLDB 2015 Research 10
# Novel DB Architectures

Satoshi Hikida@Titech

hikida@de.cs.titech.ac.jp

VLDB2015&SIGIR2015勉強会

R10−1

# Coordination Avoidance in Database Systems

Peter Bailis (UC Berkeley), Alan Fekete (University of Sydney), Michael Franklin (UC Berkeley), Ali Ghodsi (UC Berkeley), Joseph Hellerstein (UC Berkeley), Ion Stoica (UC Berkeley)

※図は全て論文からの引用です.

# 論文概要

- **背景・動機**
  - 常に直列化可能（serializable）なトランザクションはスケーラビリティ，アベイラビリティ，パフォーマンスに難がある
    - coordinationやblockingが発生するため
    - 全トランザクションを直列化可能にしなくても良いアプリもある
  - 並行操作のcoordinationやblockingを最小化すればDBのスケーラビリティ等を最大化出来る
- **貢献**
  - Invariant confluence（形式的フレームワーク）を開発
    - アプリがcoordinationを必要とするかどうかを判定可能
    - 分析によりcoordinationを最小限に抑制出来る
  - 2PLよりもスケーラビリティのあることを実証

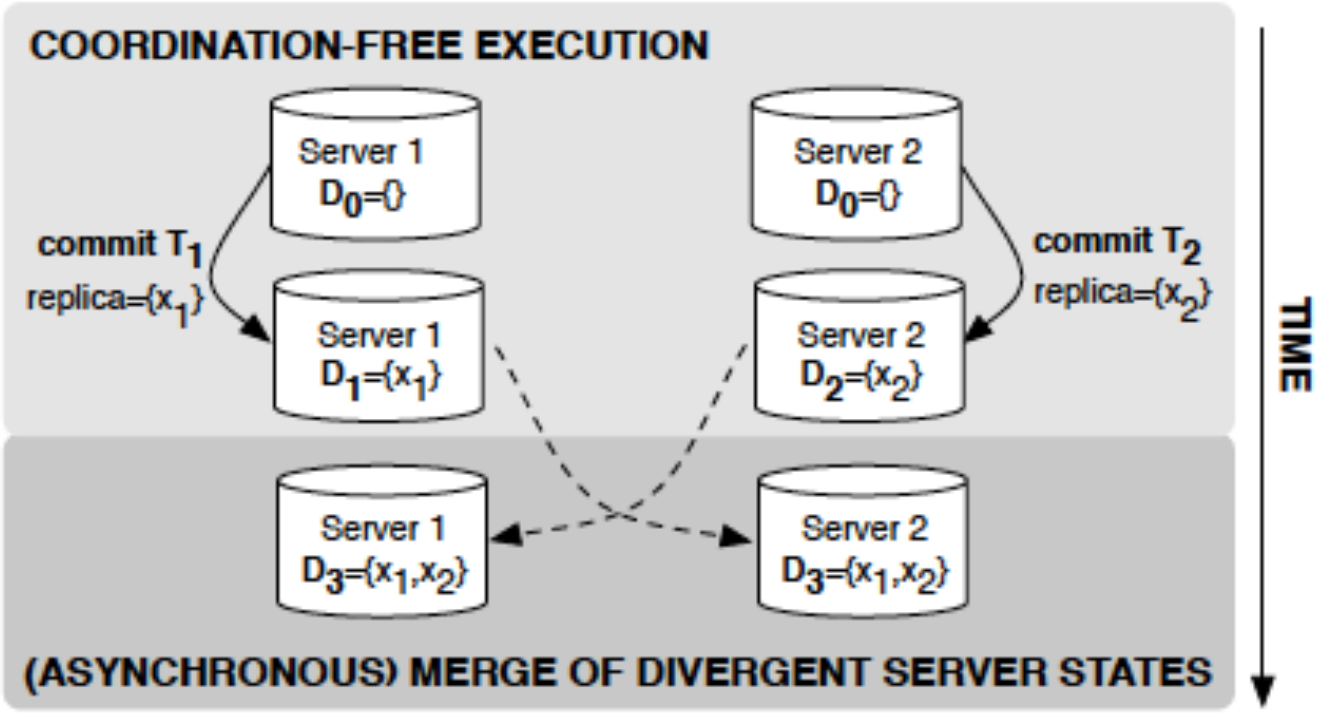
# システムモデル

- **Database**: $D$ (ある時点でのDB状態)
  - データ項目の一意なバージョンの集合
    - 各バージョンは一つ以上のサーバーで保持される
  - $\mathcal{D}$ : 可能なDB状態の集合(バージョン集合の集合)
- **Transactions**: $T : \mathcal{D} \rightarrow \mathcal{D}$
- **Merging**: $\sqcup : \mathcal{D} \times \mathcal{D} \rightarrow \mathcal{D}$ (集合操作のUnion)
  - Transactionのcommitで各replicaはmergeされる
- **Invariants**: $I : \mathcal{D} \rightarrow \{true, false\}$

# システムモデル (続き)

- いくつかの述語を満たす時，DB状態はInvariant I の下で*valid*(I-valid)であるという
  - Definition は1から5まであるが，本資料では割愛

例）トランザクションT1とT2は**coordination-free**で実行可能



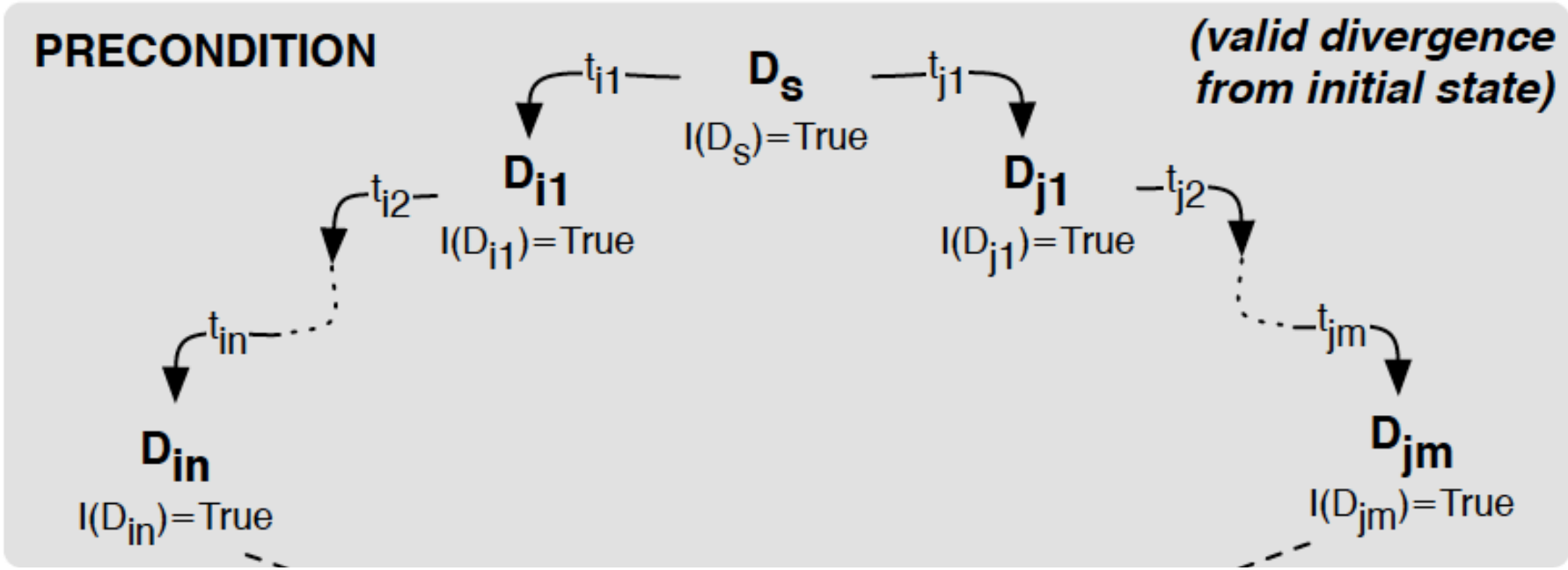
# Invariant Confluence: I-Confluence

状態$S_i$が*I-T reachable*であるとは
あるinvariant $I$とトランザクション集合$T$が与えられた時，あるトランザクション列（半順序）がmerge後に$I$-validな$S_i$を導出する

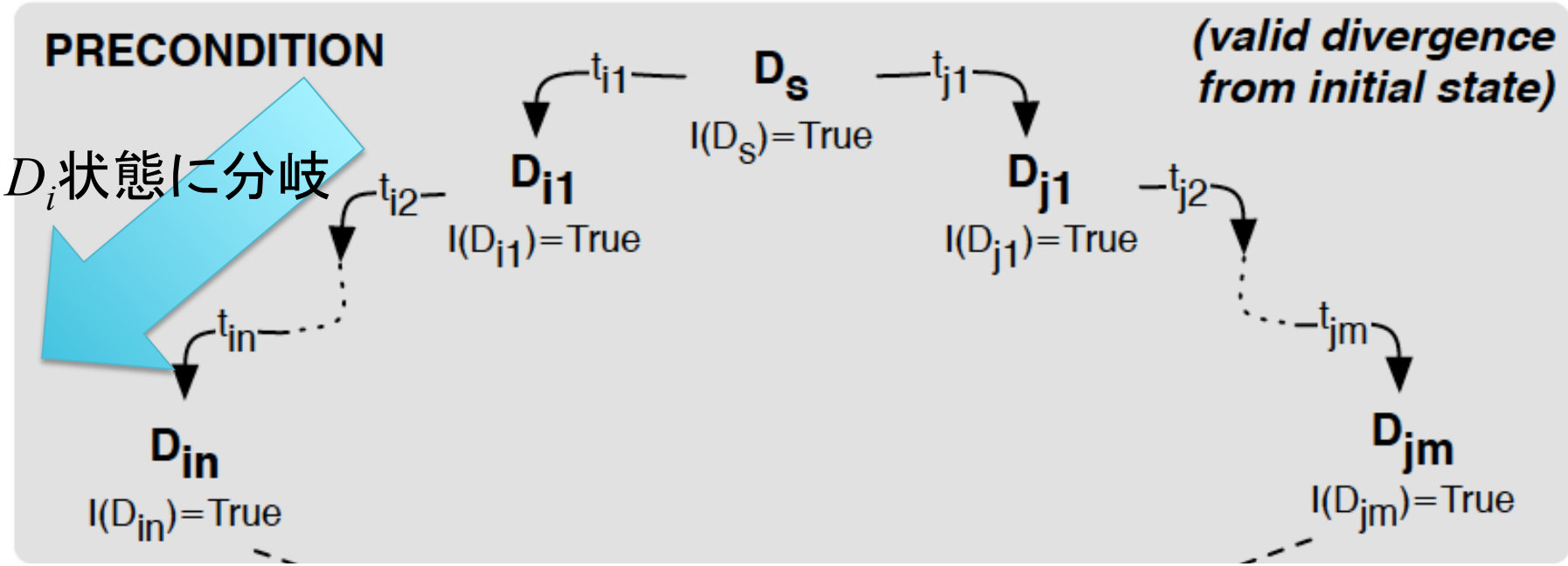
**定義（*I-Confluence*）**
トランザクション$T$の集合が*invariant* $I$ に関して$I$-Confluenceであるとは，全ての*I-T* reachable状態$D_i$, $D_j$が一つの共通状態を祖先に持つとき，$D_i \sqcup D_j$ は$I$-validであるとき

分岐しているI-validなDB状態からI-validなDB状態にmerge可能

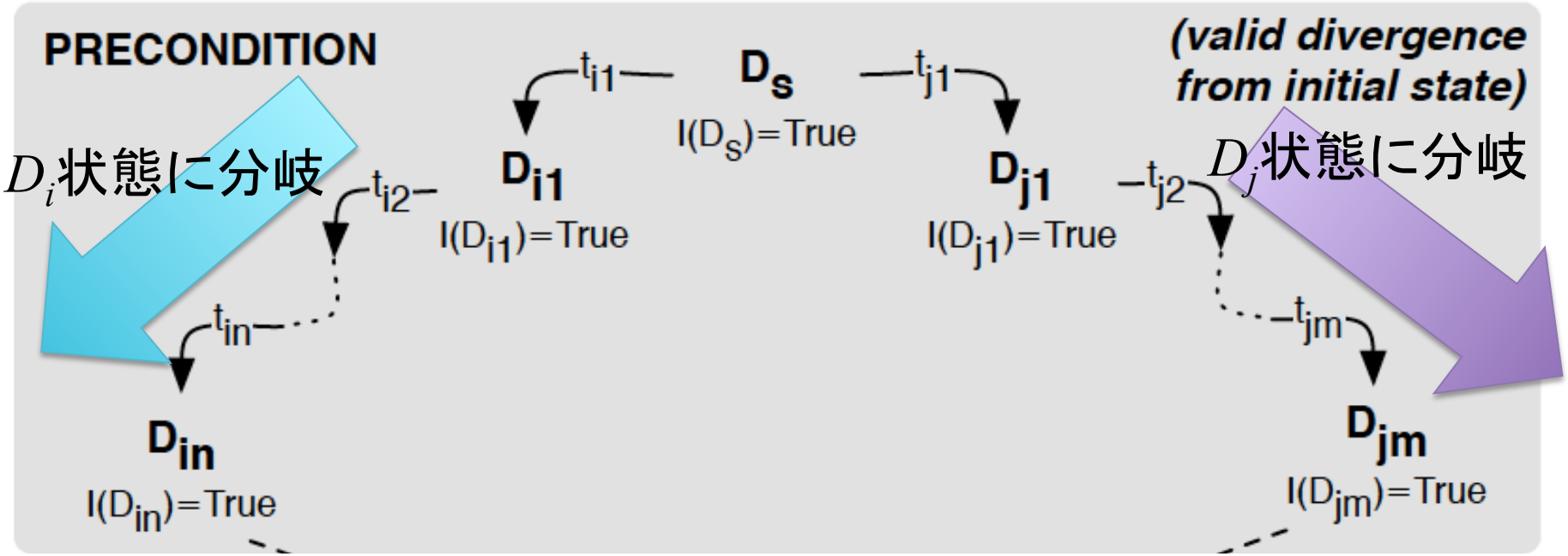
# I-Confluenceなトランザクション実行例



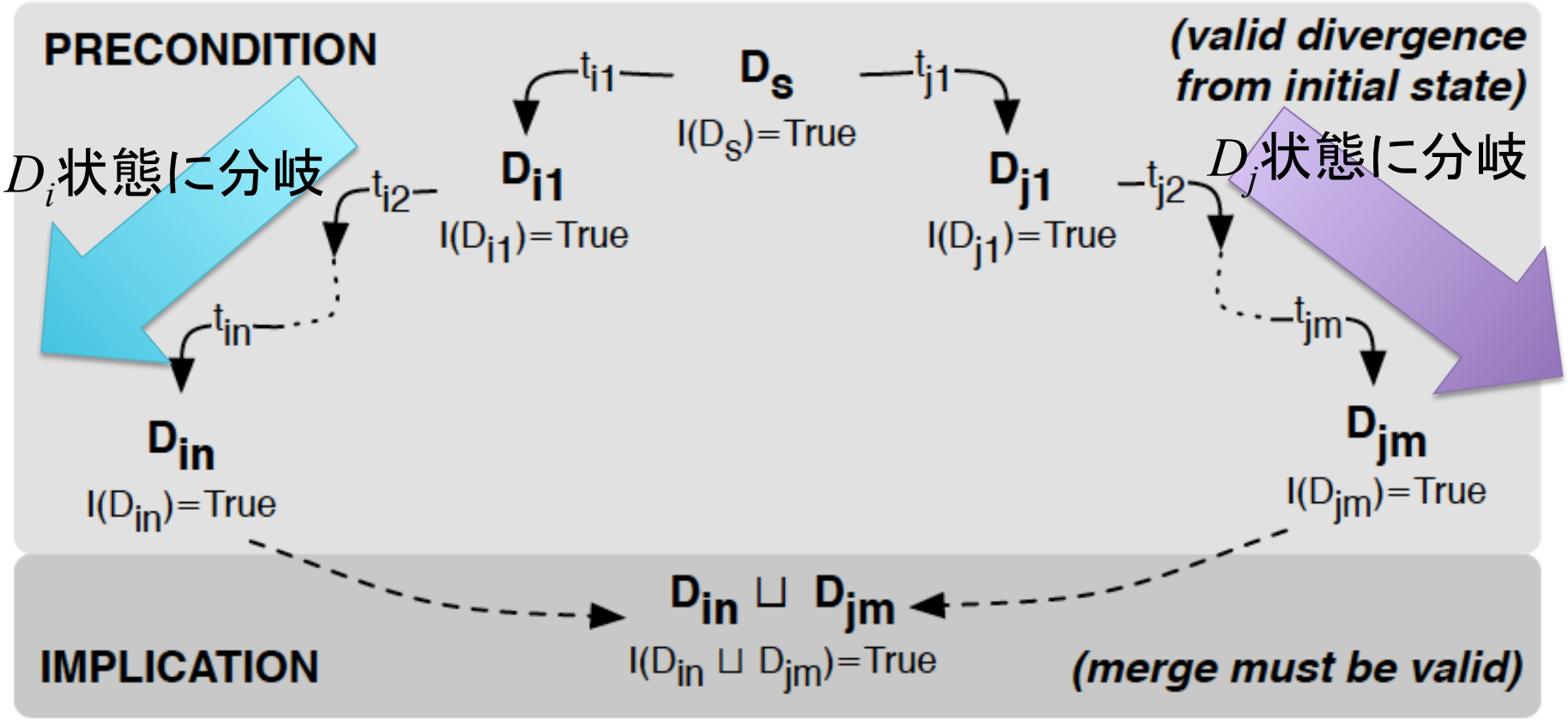
# I-Confluenceなトランザクション実行例



# I-Confluenceなトランザクション実行例



# I-Confluenceなトランザクション実行例




merge後のDB状態もI-validとなる

# I-Confluence analysis

- SQLの各種制約と操作の組についてI-Confluenceを分析
  - 殆どの組み合わせがI-Confluence（証明は論文参照）

| Invariant | Operation | $\mathcal{I}$-C? | Proof # |
|---|---|---|---|
| Attribute Equality | Any | Yes | 1 |
| Attribute Inequality | Any | Yes | 2 |
| Uniqueness | Choose specific value | No | 3 |
| Uniqueness | Choose some value | Yes | 4 |
| AUTO_INCREMENT | Insert | No | 5 |
| Foreign Key | Insert | Yes | 6 |
| Foreign Key | Delete | No | 7 |
| Foreign Key | Cascading Delete | Yes | 8 |
| Secondary Indexing | Update | Yes | 9 |
| Materialized Views | Update | Yes | 10 |
| > | Increment [Counter] | Yes | 11 |
| < | Increment [Counter] | No | 12 |
| > | Decrement [Counter] | No | 13 |
| < | Decrement [Counter] | Yes | 14 |
| [NOT] CONTAINS | Any [Set, List, Map] | Yes | 15, 16 |
| SIZE= | Mutation [Set, List, Map] | No | 17 |

# TPC-CのI-Confluence分析

- TPC-Cのinvariantを定義し，I-Confluence分析を実施

| # | Informal Invariant Description | Type | Txns | $\mathcal{I}$-C |
|---|---|---|---|---|
| 1 | YTD wh sales = sum(YTD district sales) | MV | P | Yes |
| 2 | Per-district order IDs are sequential | $S_{ID}$+FK | N, D | No |
| 3 | New order IDs are sequentially assigned | $S_{ID}$ | N, D | No |
| 4 | Per-district, item order count = roll-up | MV | N | Yes |
| 5 | Order carrier is set iff order is pending | FK | N, D | Yes |
| 6 | Per-order item count = line item roll-up | MV | N | Yes |
| 7 | Delivery date set iff carrier ID set | FK | D | Yes |
| 8 | YTD wh = sum(historical wh) | MV | D | Yes |
| 9 | YTD district = sum(historical district) | MV | P | Yes |
| 10 | Customer balance matches expenditures | MV | P, D | Yes |
| 11 | Orders reference New-Orders table | FK | N | Yes |
| 12 | Per-customer balance = cust. expenditures | MV | P, D | Yes |

Invariant type: **MV**=materialized view, **$S_{ID}$**=sequential ID assignment, **FK**=foreign key

Transactions: **N**=New-Order, **P**: Payment, **D**:Delivery

# TPC-CのI-Confluence分析

- TPC-Cのinvariantを定義し，I-Confluence分析を実施

| # | Informal Invariant Description | Type | Txns | $\mathcal{I}$-C |
|---|---|---|---|---|
| 1 | YTD wh sales = sum(YTD district sales) | MV | P | Yes |
| 2 | Per-district order IDs are sequential | $S_{ID}$+FK | N, D | No |
| 3 | New order IDs are sequentially assigned | $S_{ID}$ | N, D | No |
| 4 | Per-district, item order count = roll-up | MV | N | Yes |
| 5 | Order carrier is set iff order is pending | FK | N, D | Yes |
| 6 | Per-order item count = line item roll-up | MV | N | Yes |
| 7 | Delivery date set iff carrier ID set | FK | D | Yes |
| 8 | YTD wh = sum(historical wh) | MV | P, D | Yes |
| 9 | YTD district = sum(hi | | | |
| 10 | Customer balance matches expenditures | MV | P, D | Yes |
| 11 | Orders reference New-Orders table | FK | N | Yes |
| 12 | Per-customer balance = cust. expenditures | MV | P, D | Yes |

I-Confluenceで無い⇨Coordinationが必要

Invariant type: **MV**=materialized view, **$S_{ID}$**=sequential ID assignment, **FK**=foreign key

Transactions: **N**=New-Order, **P**: Payment, **D**:Delivery

# 評価: Serializableとの比較

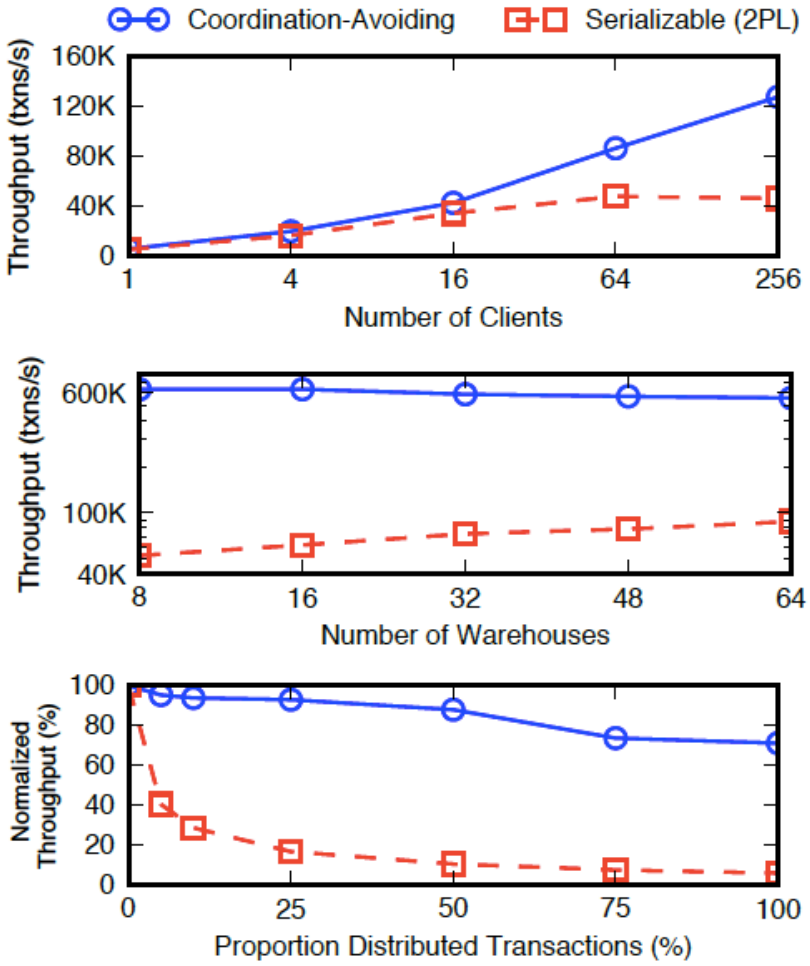



**Figure 5: TPC-C New-Order throughput across eight servers.**

- JVM-basedなトランザクション処理システムのプロトタイプで評価
- TPC-C:New-Orderで評価
  - 10%のTxnsはcoordinationが必要
- **Coordination-Avoiding**
  - I-confluence → coordination-free
- **Serializable (2PL)**
  - 全てのTxnはserializable

Coordination-Avoidingは
Serializableよりもスケーラビリティに優れていることを実証

# 評価: 提案手法のスケーラビリティ

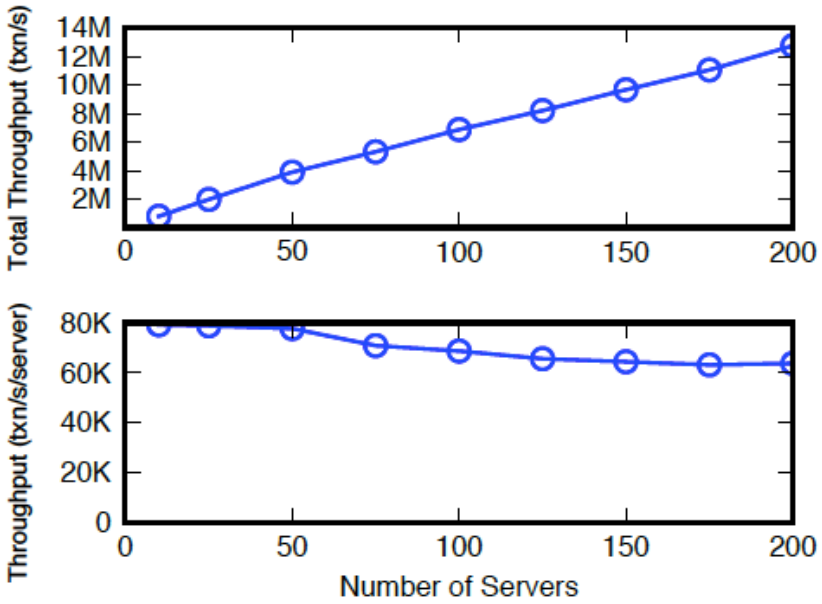



**Figure 6: Coordination-avoiding New-Order scalability.**

サーバー台数に応じてスループットが（ほぼ）線形に増加

100台以上ではサーバー当たりのスループット劣化は横ばい

| Code Path | Cycles |
|---|---|
| Storage Manager (Insert, Update, Read) | 45.3% |
| Stored Procedure Execution | 14.4% |
| RPC and Networking | 13.2% |
| Serialization | 12.6% |
| ID Assignment Synchronization (spinlock contention) | 0.19% |
| Other | 14.3% |

消費するCPUサイクルでcoordinationの割合は0.2%以下→
**Coordinationがボトルネックでは無い**