【VLDB2010勉強会】

# Session 2: Parallel and Distributed Databases

担当：山室健（NTT）

# 紹介する3本の論文

- 1. Carlo C.(MIT) et al.: Schism: a Workload-Driven Approach to Database Replication and Partitioning
- 2. Lu Q.(Hong Kong univ.) et al.: Ten Thousand SQLs: Parallel Keyword Queries
- 3. Alexander T.(Yale univ.) et al.: The Case for Determinism in Database Systems

# Schism: a Workload-Driven Approach to Database Replication and Partitioning

- 論文の概要
  - 分散DB上の各マシンにデータを分割/複製する場合に、**誤ったデータ配置戦略を適用すると分散Tr.が大量に発生**し、性能劣化が発生
  - 入力ワークロードとスキーマ情報を用いて、**アクセス関係を表すグラフを構築し、最小カットを適用**することで、マシン間にまたがる分散Tr.を最小化
  - プロトタイプを作成し、TPC-C/Eと複雑なモデル(ソーシャルネットワークのデータ)に本手法を適用し、分散Tr.の割合を評価
- 分散Tr.のコスト
  - Tr.に必要なデータが複数マシンにまたがっている場合に分散Tr.(Two-phase commit)が発生
  - 処理が単体で完了する場合と比較して、**約2倍の性能差**
    - MySQLの分散Tr.実装を利用

約2倍の性能差

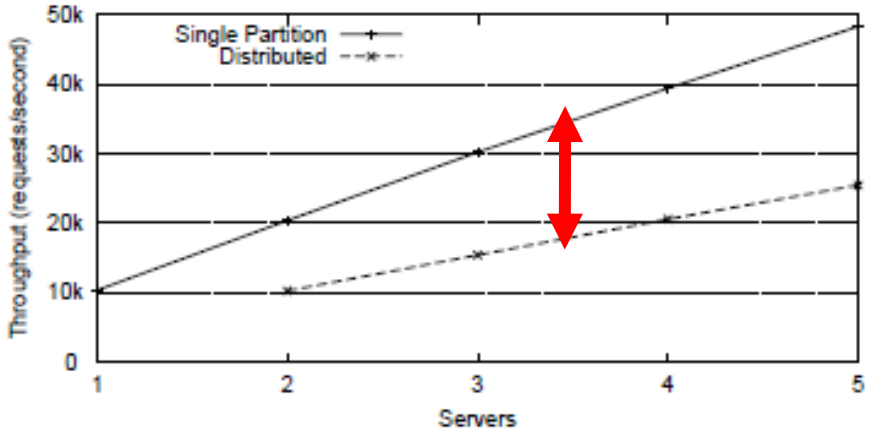



Figure 1: Throughput of distributed transactions

# Schism: a Workload-Driven Approach to Database Replication and Partitioning

- ワークロードからタプル間アクセス関係をグラフ構造化して、分散Tr.のコストが最も低くなる分割点を最小カット問題を解くことで決定
  - 1. タプルをNODE、同一のTr.によってアクセスされるタプル間にEDGEを構成（Fig.2）
  - 2. EDGEの重みに、アクセス回数を設定（Fig.2）
  - 3. 各NODE（タプル）の更新を考慮するために、単一NODEを複数NODEに展開（Fig.3の黄色い領域）
    - EDGEの重みを更新回数にして、更新の少ないタプルを複製対象にしやすくする

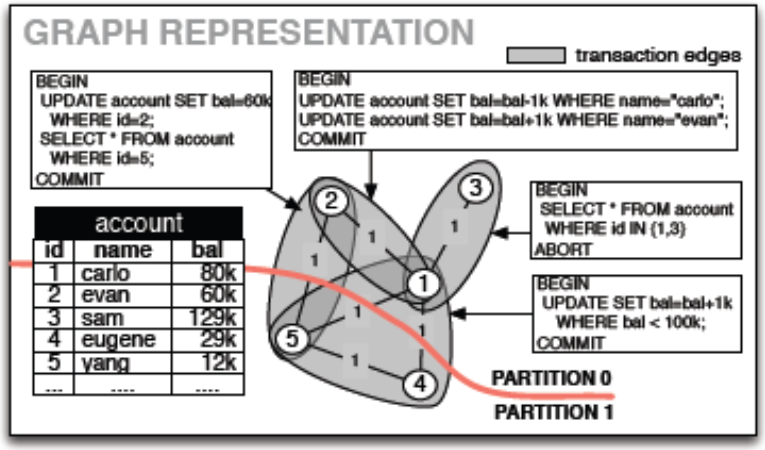



Figure 2: The graph representation

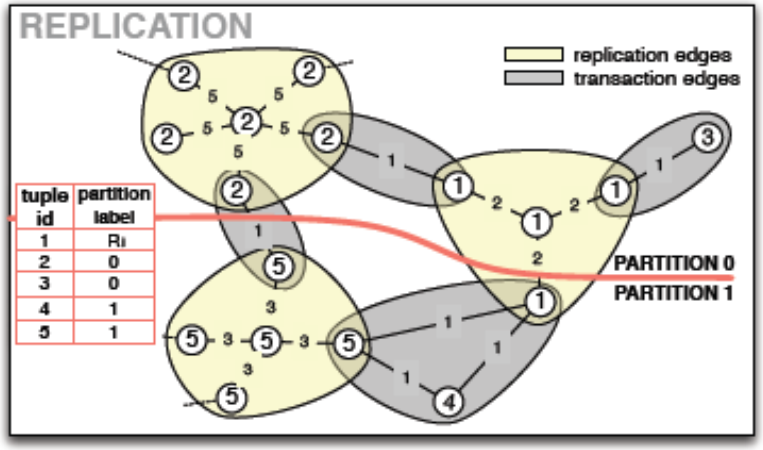
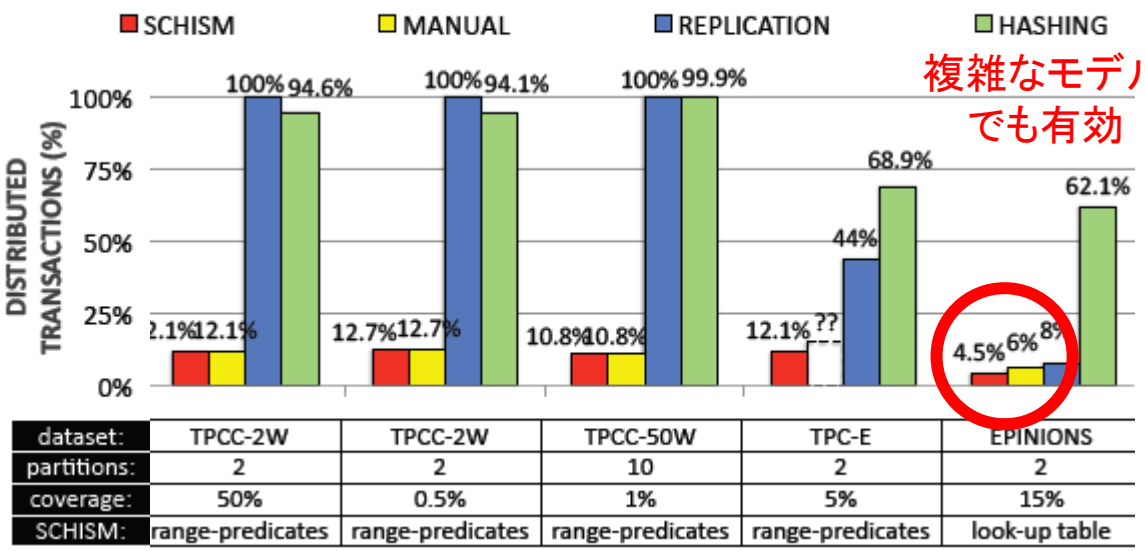



Figure 3: Graph with replication

# Schism: a Workload-Driven Approach to Database Replication and Partitioning

- 様々なワークロードにおける、提案手法と従来手法を分散Tr.割合で評価（Fig.4）
  - SCHISM: 提案手法
  - MANUAL: 最も効果的になるように、（MITの学生が）手動で分割したもの（目標値）
  - REPLICATION: 全てのデータを複製
  - HASHING: 全てのデータをHASHを用いて分割
- パーティションサイズとワークロードの複雑さに対する分割決定時間の評価（Fig.5）




| dataset: | TPCC-2W | TPCC-2W | TPCC-50W | TPC-E | EPINIONS |
|---|---|---|---|---|---|
| partitions: | 2 | 2 | 10 | 2 | 2 |
| coverage: | 50% | 0.5% | 1% | 5% | 15% |
| SCHISM: | range-predicates | range-predicates | range-predicates | range-predicates | look-up table |

Figure 4: Schism database partitioning performance.

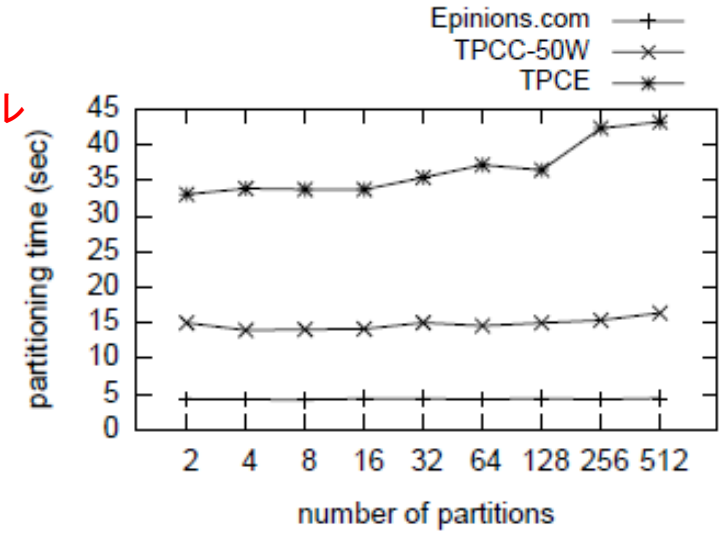



Figure 5: METIS graph partitioning scalability for growing number of partitions and graph size.

# Ten Thousand SQLs: Parallel Keyword Queries Computing

- 論文の概要
  - RDBMSに探索キーワードの集合を与えて、結果として外部キーで相互接続されるタプル結果の集合を返す問題
    - ex）DBLP上のデータに対して、著者名の集合を探索キーワードとして与えた場合に、引用(Cite)関係で相互接続される結果タプルの集合を返却
  - 結果を作成するにあたり大量の処理(SQL)が生成されるため、マルチコア環境を前提に各コアに処理を均一化させながら、冗長な処理を最小化したい(Multi-query optimization問題)
  - SQL単位、Operators単位で各コアに処理を振り分ける手法に対して、著者らが提案するData単位での処理の振り分けが最も負荷の偏りに強いことが判明
  - DBLPデータを用いて評価を行い、提案手法が16コアまでほぼ線形にスケールすることが判明

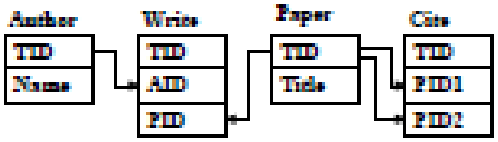


Figure 1: *DBLP* Database Schema

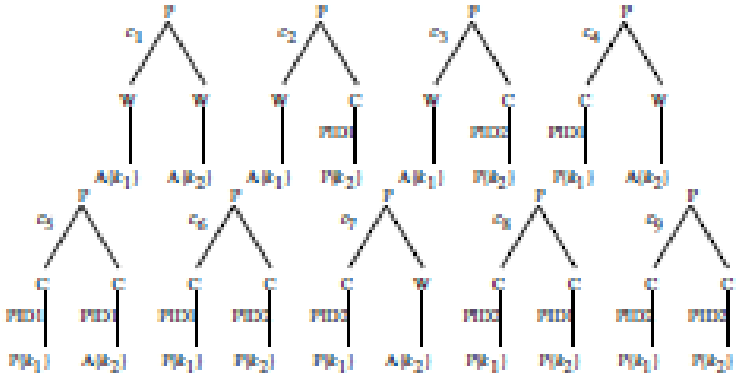


Figure 2: Nine *CNs*

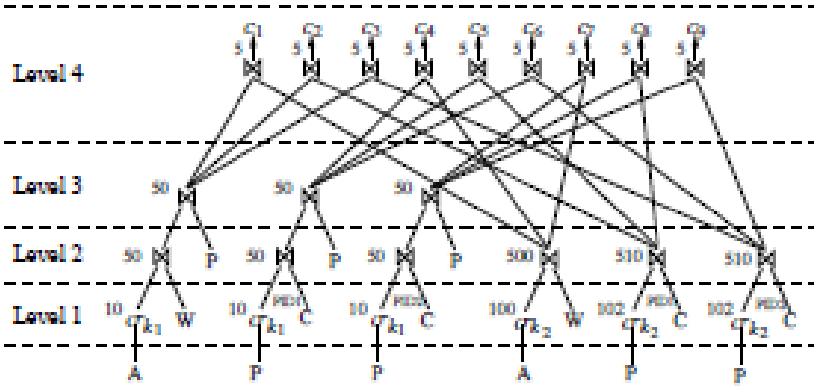


Figure 3: A Partial Execution Graph for Query $\{k_1, k_2\}$

# Ten Thousand SQLs: Parallel Keyword Queries Computing

- 提案手法: Data単位での処理の振り分け(DLP)
  - SQL単位(CLP)、Operators単位(OLP)での処理の振り分け手法において、各処理の負荷に偏りがある場合に、最適な(コアに対してリニアになるような)処理の分割ができない(Section 5.と6.)
  - 上記の従来手法を、より細かい単位であるData単位で処理をコアに振り分けることで負荷の偏りを解消
- Adaptive Schedulingにおける3つの留意点
  - 1. 最もコストの高い処理ノードを複数のコアに分割
  - 2. 結果のMerge処理は最後のPhaseで実施
    - 中間結果大、最終結果小の傾向を利用
  - 3. 処理コストの歪が最小になるデータ分割点の発見

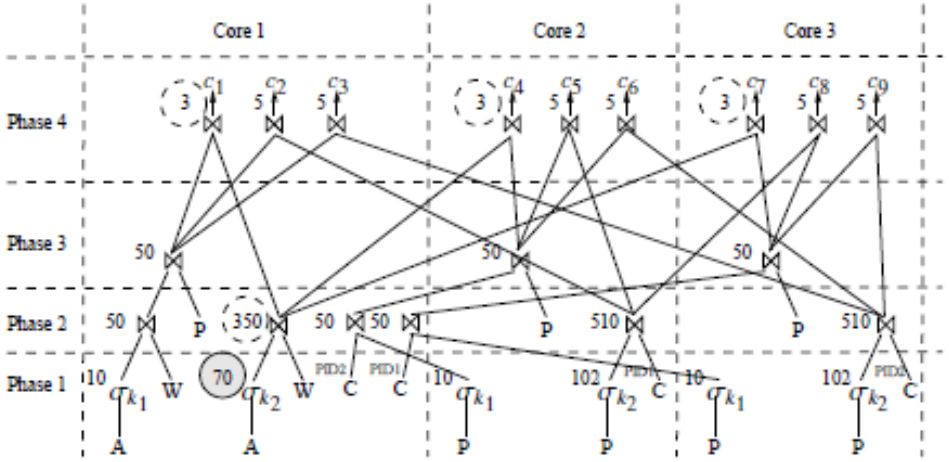


Figure 8: Adaptive Scheduling

# Ten Thousand SQLs: Parallel Keyword Queries Computing

- DBLP上のデータを対象に、従来手法（CLPとOLP）と提案手法（DLP）の性能比較を実施（Knum: keyword数を変化させた場合）
  - DBLP
    - データサイズ: 686MB
    - 4リレーションから構成
    - タプル数は5,662,265
  - コア数を増加させた場合の処理完了時間を比較
- 提案手法は16コアまでほぼリニアに処理がスケール
  - 従来手法はコア間に偏りが発生し、処理時間がコア数増加に対して改善しない

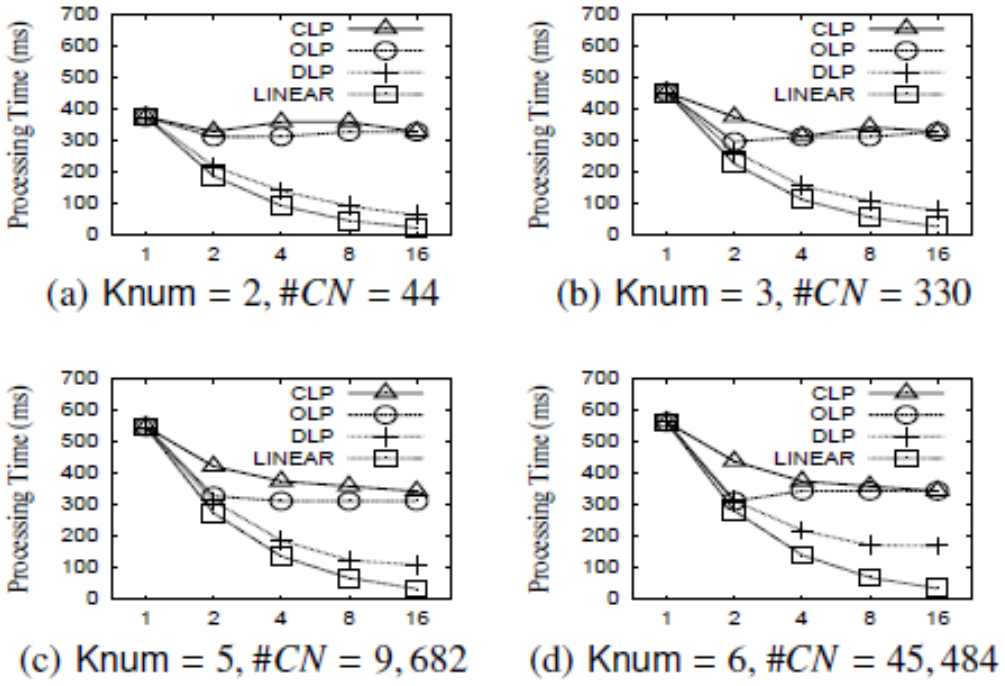



**Figure 16: Vary Knum (*DBLP*)**

# The case for Determinism in Database System

- 論文の概要
  - 分散DB環境下のレプリカに対する処理（OLTPを想定）のDeadlock-freeで、効率的な平行順序制御手法を提案
  - 全レプリカに対して同期的に更新するレプリケーション構成を前提に、“全てのTr.の順序を事前決定”することで、分散DB上でのボトルネックを排除
  - 結果として、分散DB環境でよく利用されるtwo-phase commitと比較し、大幅な性能向上を実現
- 従来のレプリケーション構成の3分類
  - 1. Post-write replication
    - いわゆるMaster/Slave構成
    - SlaveはMasterの更新情報をある程度遅延して反映
  - 2. Active replication with synchronized locking（今回の対象はコレ）
    - 更新処理に対して、システム全体で排他制御を行うことで順序性を維持
  - 3. Replication with lazy synchronization
    - 複数のレプリカで独立に更新処理を行い、最終的に同じ状態に

# The case for Determinism in Database System

- 本提案手法のアプローチ概要
  - 近年のOLTPの傾向（主に処理時間の短縮化）を背景に、従来手法で用いられてきた**Tr.の順序入れ替え最適化処理（Non-deterministicなアプローチ）**を除去し、システム投入時にTr.の順序を固定的に決定
- システム概要（主に3ステップから構成）
  - 1. 非決定的な処理の実行
    - random()や、now()による
  - 2. Tr.順序の決定
    - 永続デバイスへの同期書き込み
  - 3. 各レプリカへTr.処理をbroadcast
    - reliableでtotally-orderedな通信路を想定

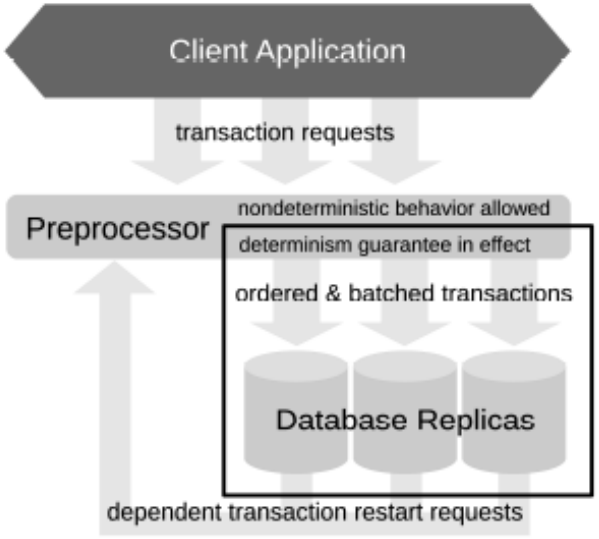



Figure 2: Architecture of a deterministic DBMS.

# The case for Determinism in Database System

- 評価対象の環境
  - データパーティショニングを適用した分散DBに対するTPC-Cで評価
  - 従来手法はノードにまたがる処理がtwo-phase commitで実施、一方提案手法はone-phase commit
  - Tr.の中で複数ノードにまたがる処理の割合で評価
    - マシン台数は2台を想定
    - 1 warehouse per 1 node, or 5 warehouse per 1node
- 実験結果
  - またがる処理が拡大すると、性能劣化が発生
    →two-phase処理の影響

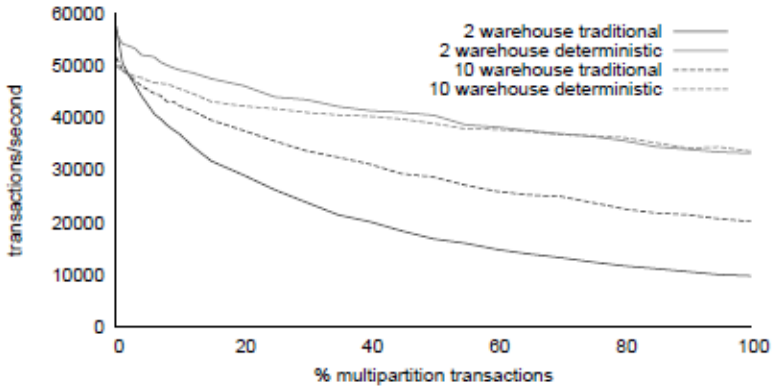



Figure 3: Deterministic vs. traditional throughput of TPC-C (100% New Order) workload, varying frequency of multipartition transactions.