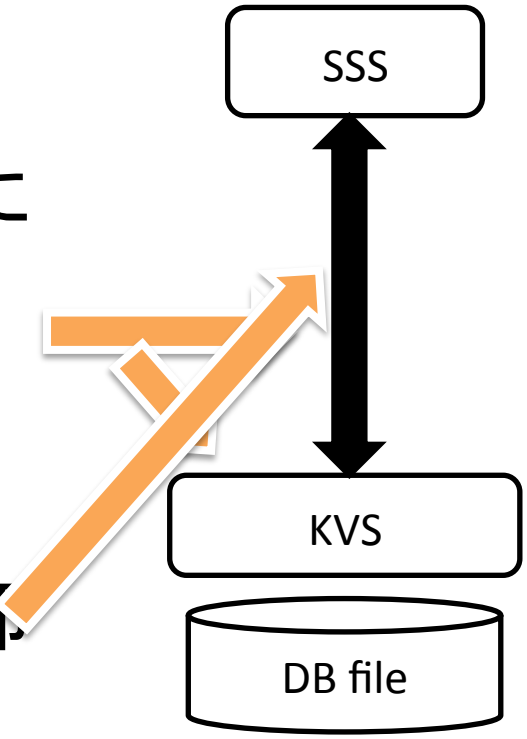
MapReduce処理系SSSにおける Key Value Storeアクセス手法の改良

中田秀基、小川宏高、工藤知宏
独立行政法人産業技術総合研究所

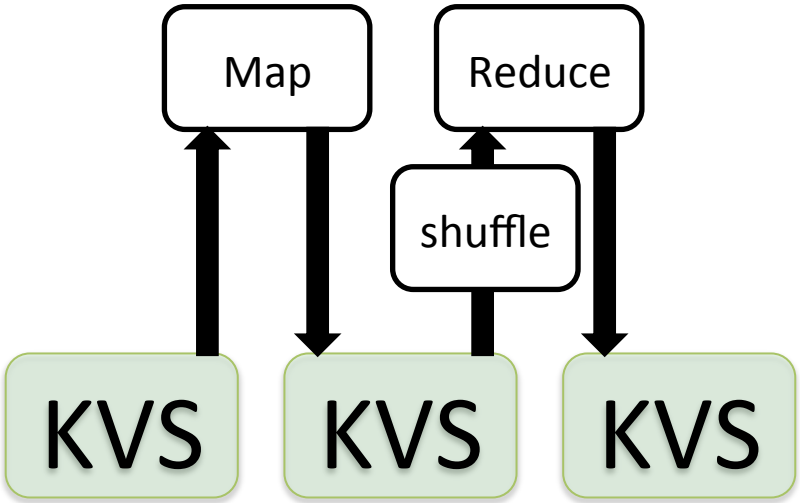
あらまし

- Key-value ストアをベースとしたSSSを開発中
- 開発はほぼ完了しているが、地道に性能改善中
  - [CPSY-2012-4 : MapReduce処理系SSSに向けたKVSの改良]
- 本発表 : KVSとのネットワーク接続部を総取り替えでさらに性能改善を目指す

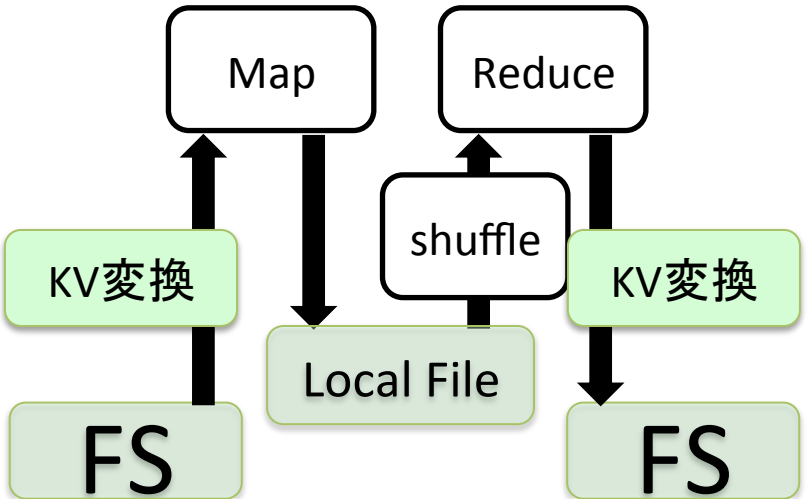


背景（1）

- MapReduce
  - Key-Value ペアに対する演算として並列アルゴリズムを抽象化
  - Apache Hadoopの普及により広く普及

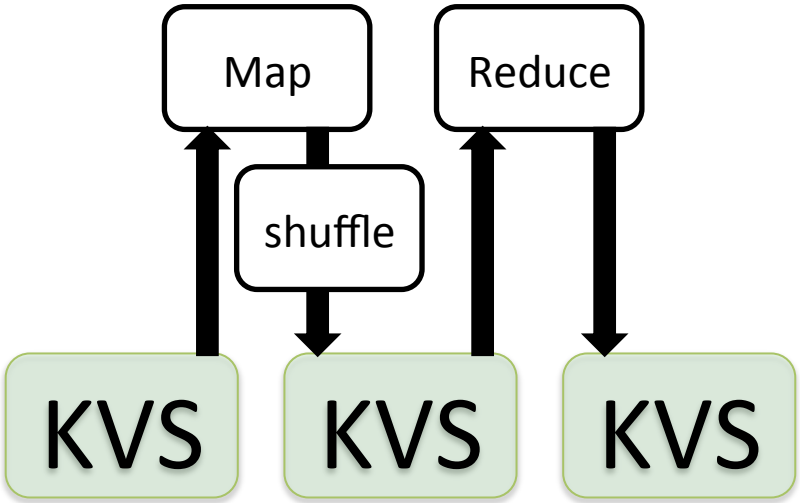


- Hadoop
  - 分散ファイルシステムベースで実装
  - I/Oは高速
  - Mapのみ、Reduceのみの計算ができない



背景（2）

- SSS MapReduce
  - 分散KVSをベースとして用いる
  - 高速なイタレーションと柔軟なワークフロー処理



- 要素KVSとしてTokyoCabinetを改変したものを使用[CPSY2012-4:中田]
  - リモートアクセスにはTokyoTyrantを利用
  - TokyoTyrantがボトルネックに

研究の目的と成果

- 要素KVSへのアクセスプロトコルの改良
  - TokyoTyrantから独自のデータサーバへ
  - データベースファイルの持ち方を変更

- 改変の効果を検証
  - マイクロベンチマーク
  - マクロベンチマーク
  - 削除の高速化

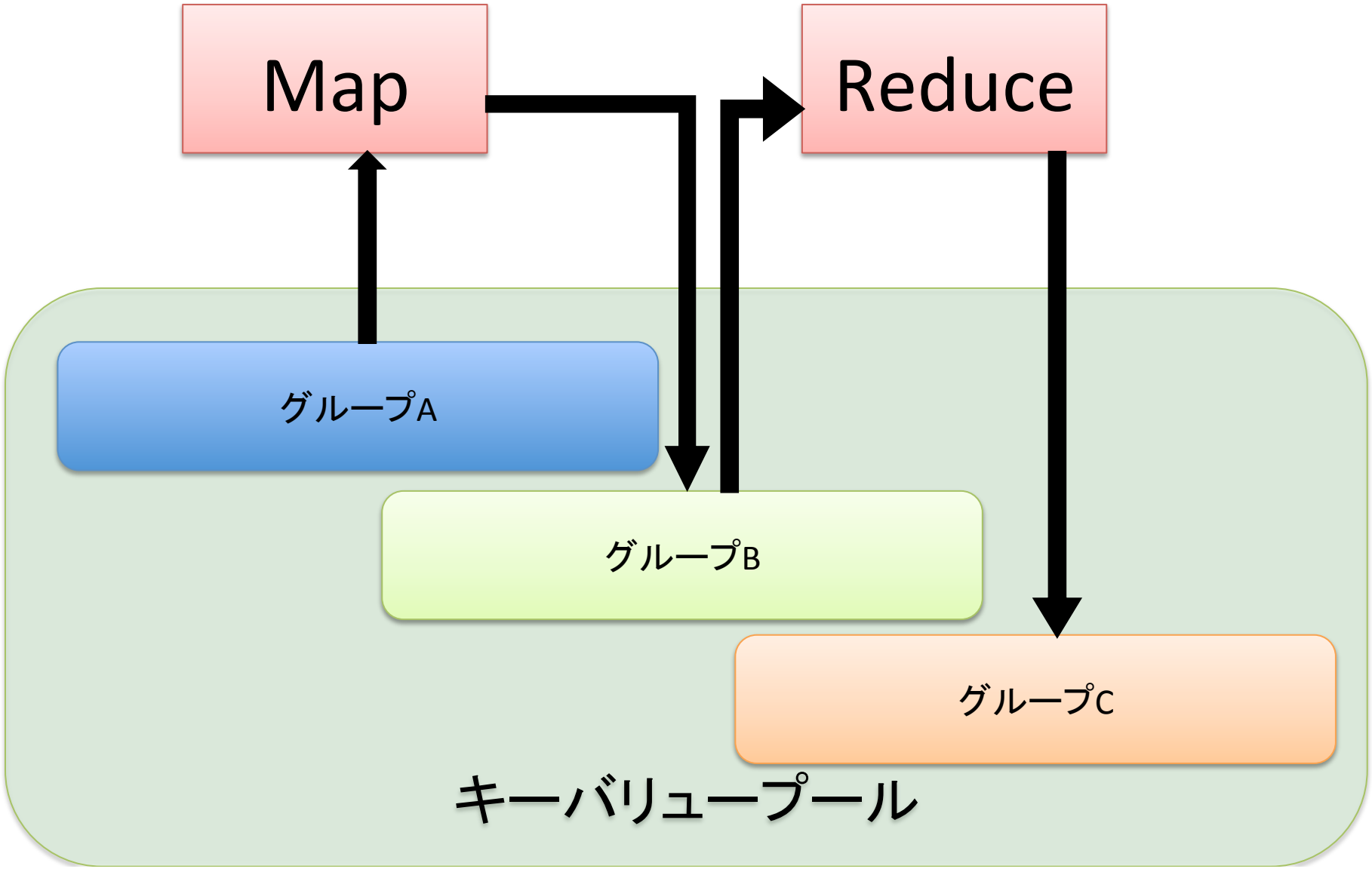
発表の概要

- SSSの概要
  - 実装のポイント
- TokyoTyrantによる既存実装
- 提案手法による実装の詳細

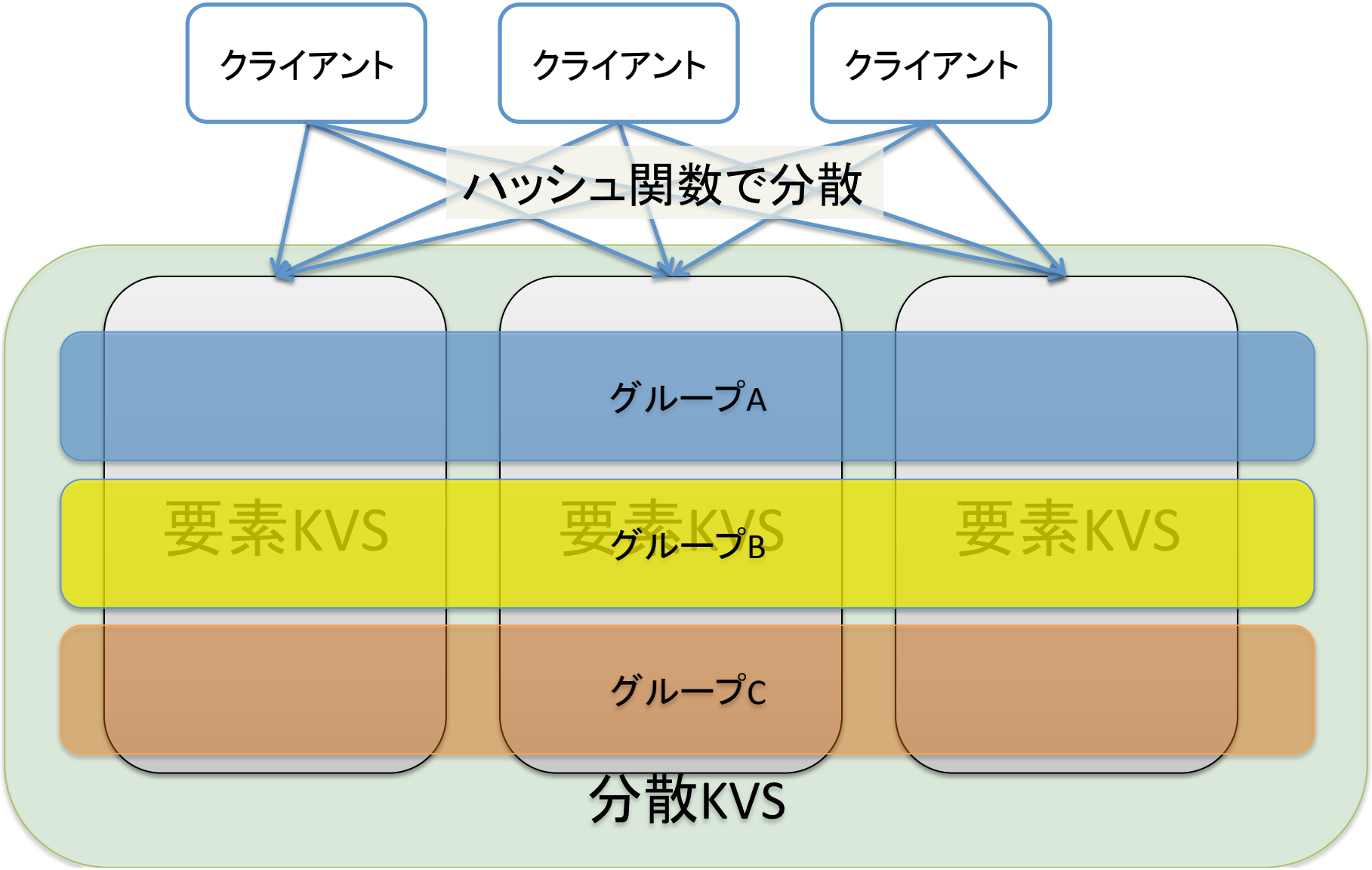
- 評価

- まとめと今後の課題

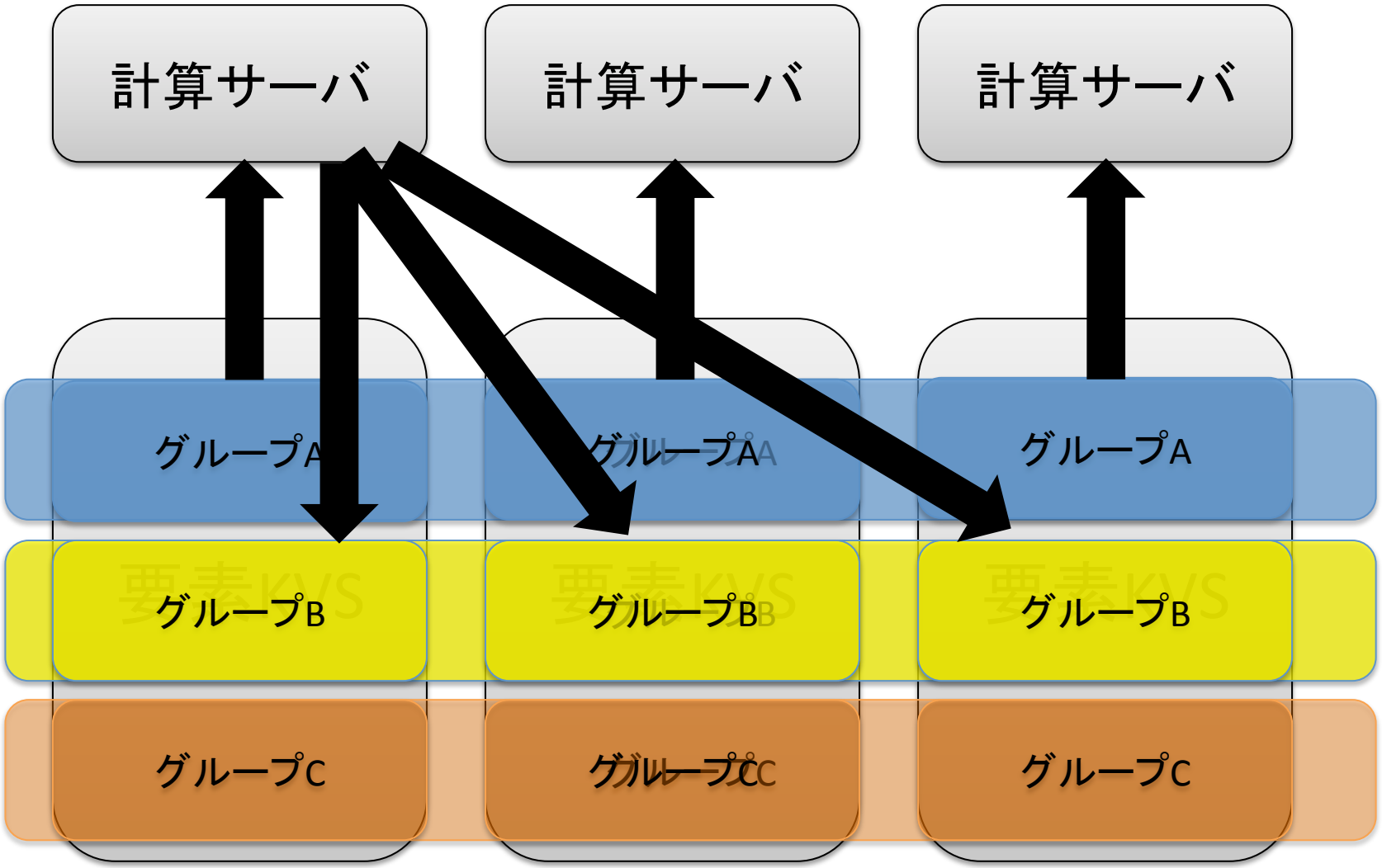
SSSの設計：タプルグループ



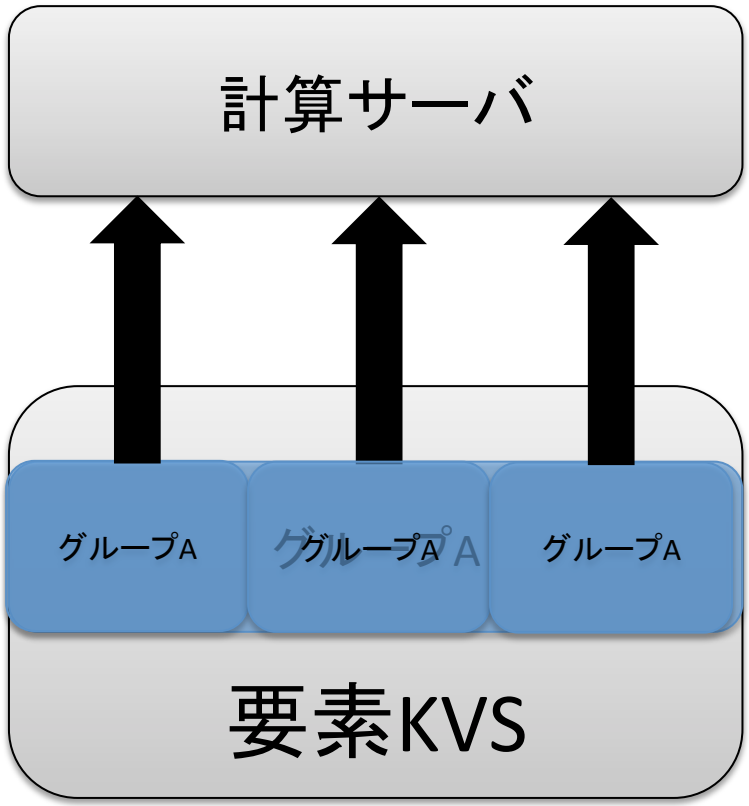
SSSの設計：分散KVS



SSSの設計：Owner Compute



SSSの設計：並列読み出し



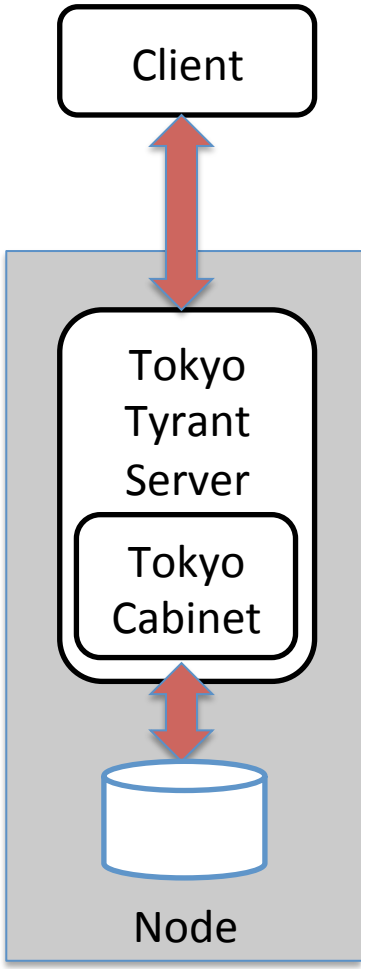
グループをサブグループに分割

並列に読み出すことによって、KVSのレイテンシの隠蔽

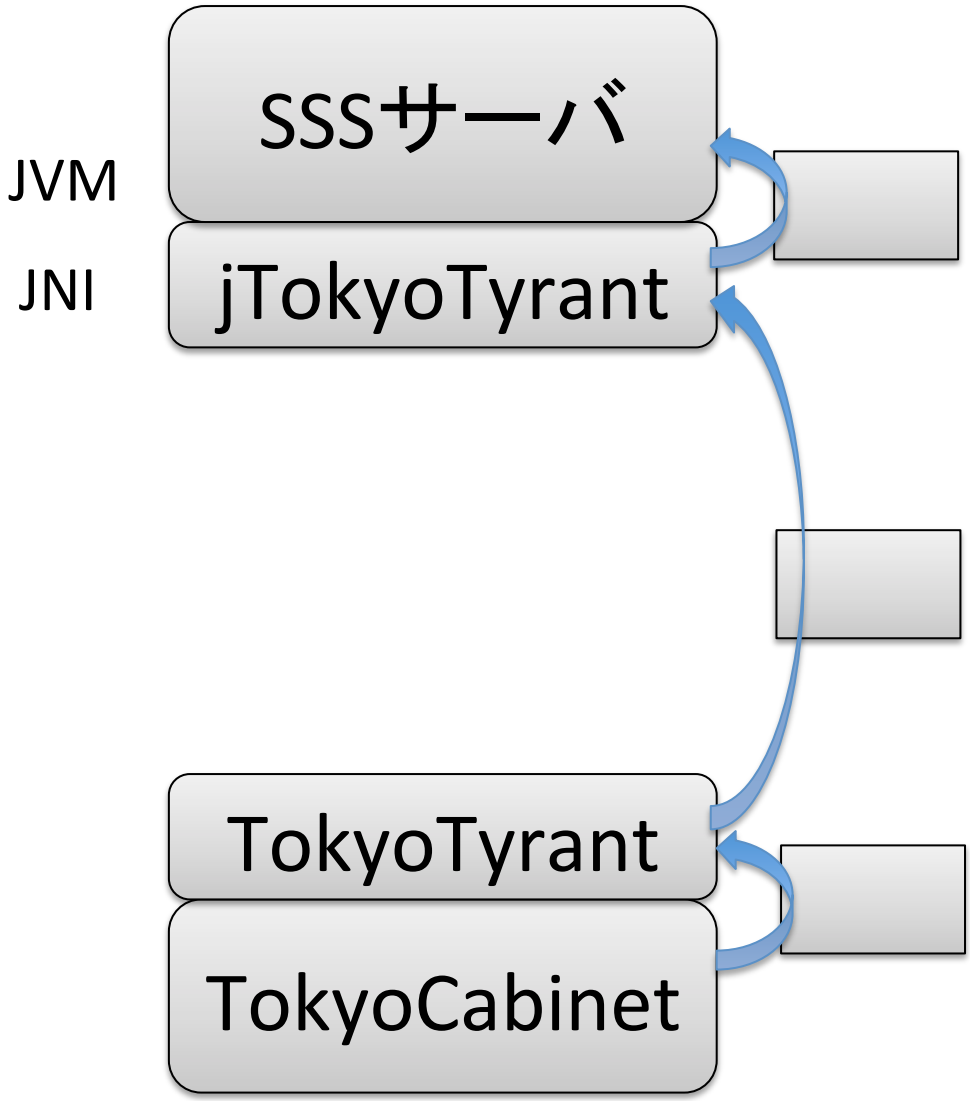
＊ランダムアクセスに強いSSDを前提

既存実装：TokyoCabinet/TokyoTyrantを利用

- Fal Labsの平林幹雄氏が開発したキーバリュー型のデータベースシステム
  - mixiで利用されていることで知られる
- TokyoCabinet
  - データベースライブラリ
  - バイナリプログラムとリンクしてファイル上のDBを操作する
  - **ロックをバルク処理に最適化**
- TokyoTyrant
  - TokyoCabinetのネットワークインターフェイス
  - サーバがTokyoCabinetのライブラリとリンクされる
  - **範囲処理プロトコルを追加**

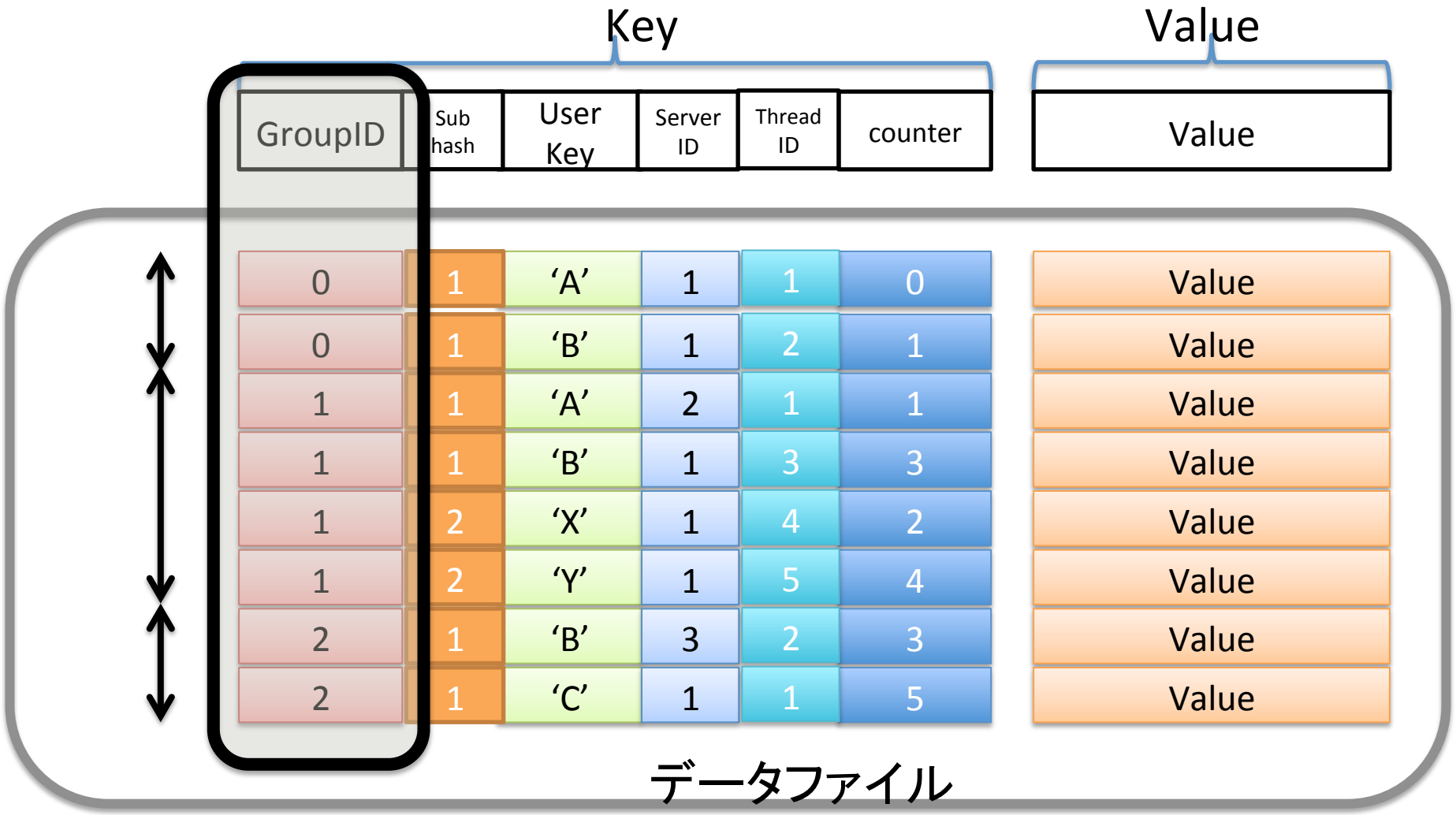


TokyoTyrant / jTokyoTyrant

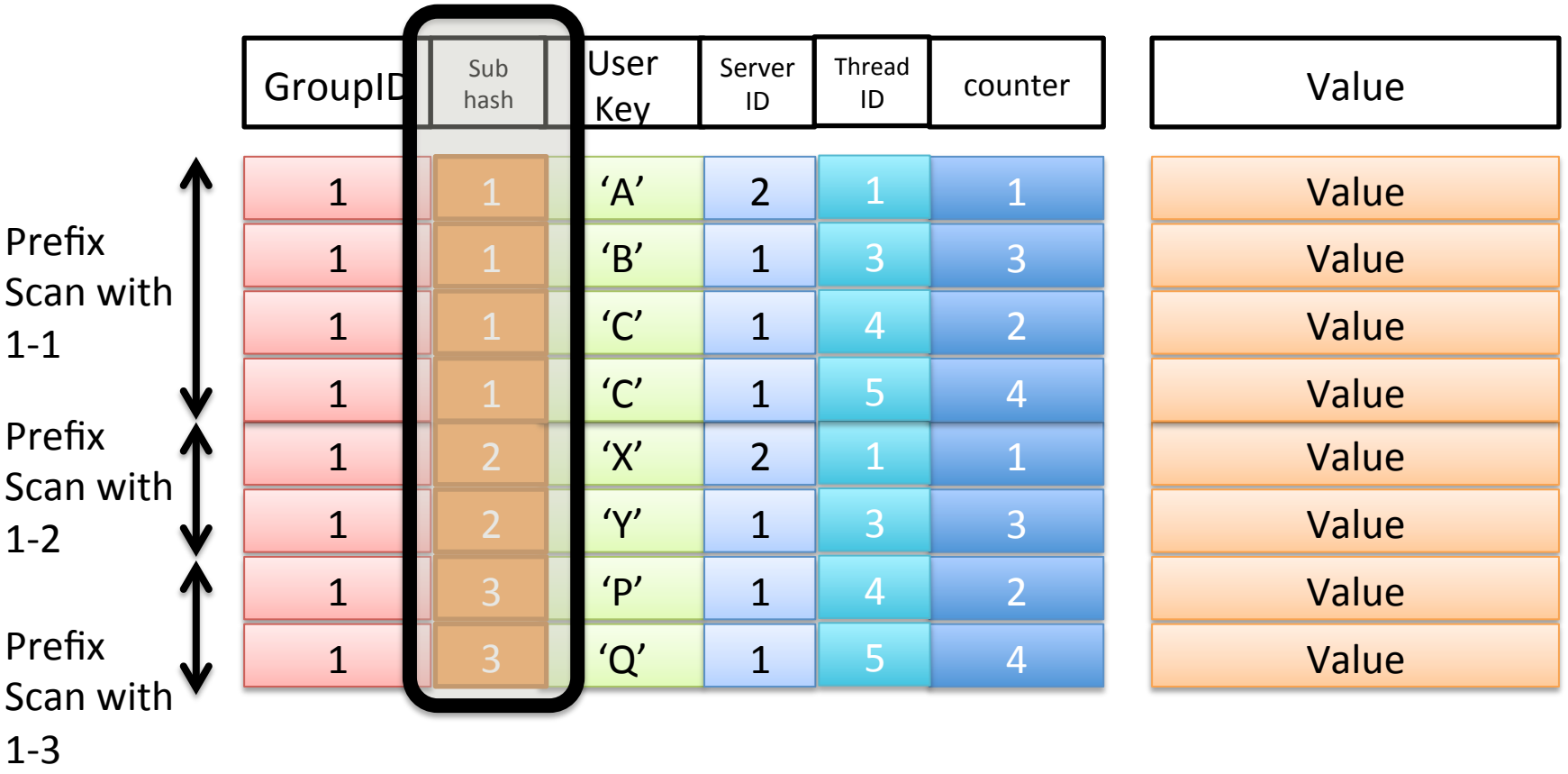


- jTokyoTyrant – JNI によるライブラリ
- 個数を指定して一括転送するプロトコル

既存実装:タプルグループ



既存実装 : 並列読み出しの実装

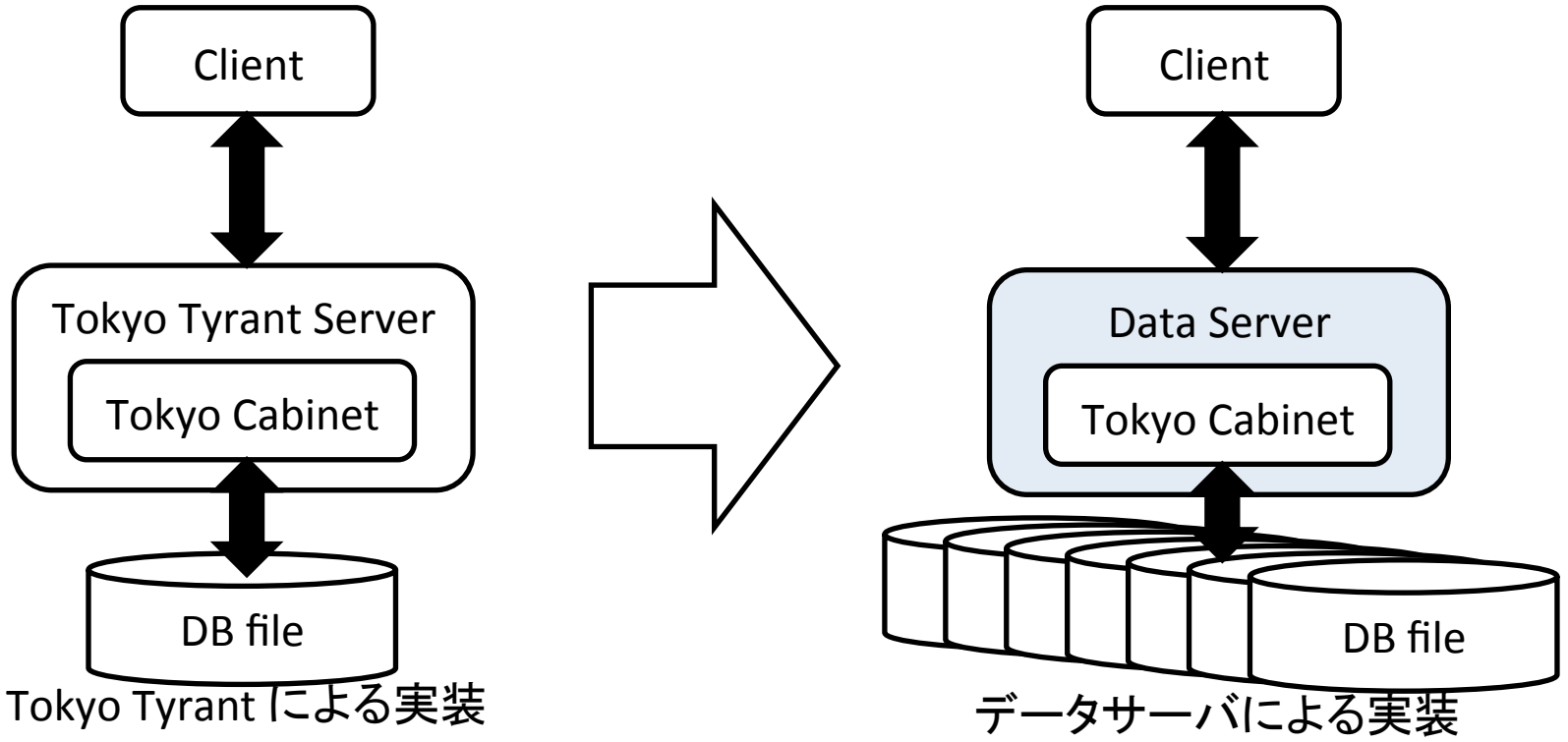


既存実装の問題点

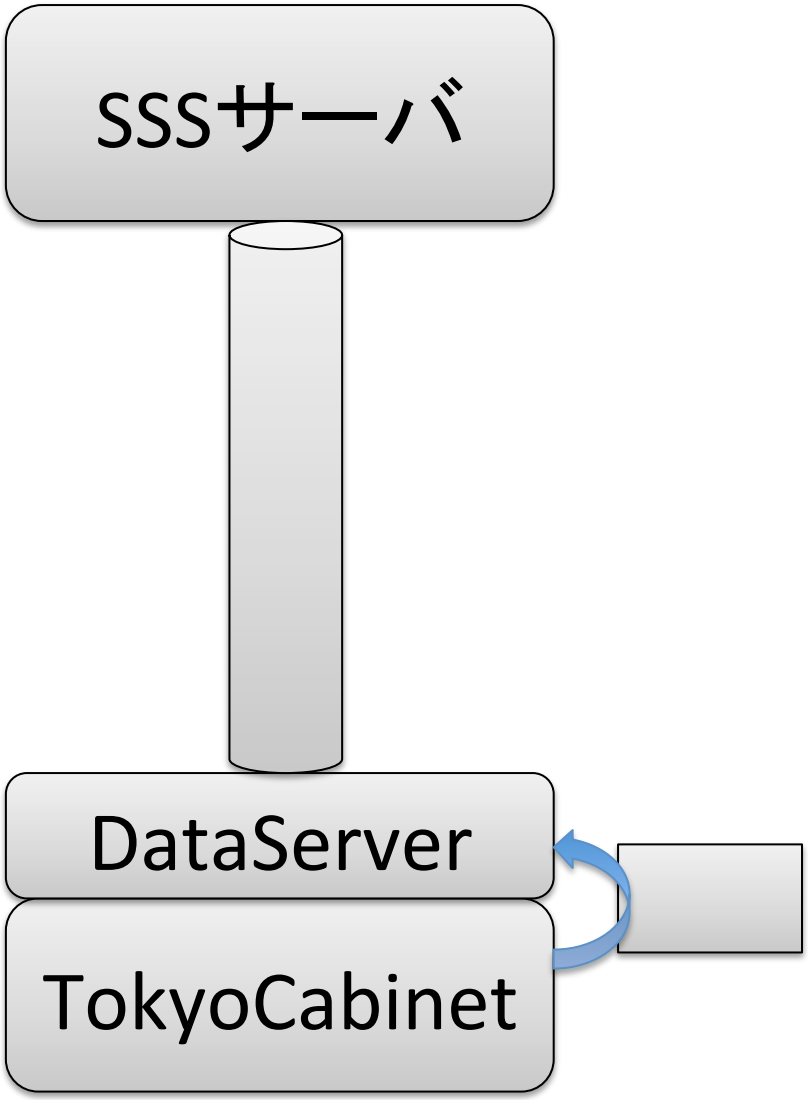
- TokyoTyrantが性能的にボトルネック
  - TokyoCabinetの速度を十分に引き出せない
- タプルグループの削除が低速
  - 繰り返し処理を行う場合に問題

提案手法の概要

- TokyoTirantに変わるデータ転送層を実装
  - データコピー回数の削減
- データベースファイルをサブグループごとに分割



新規サーバ



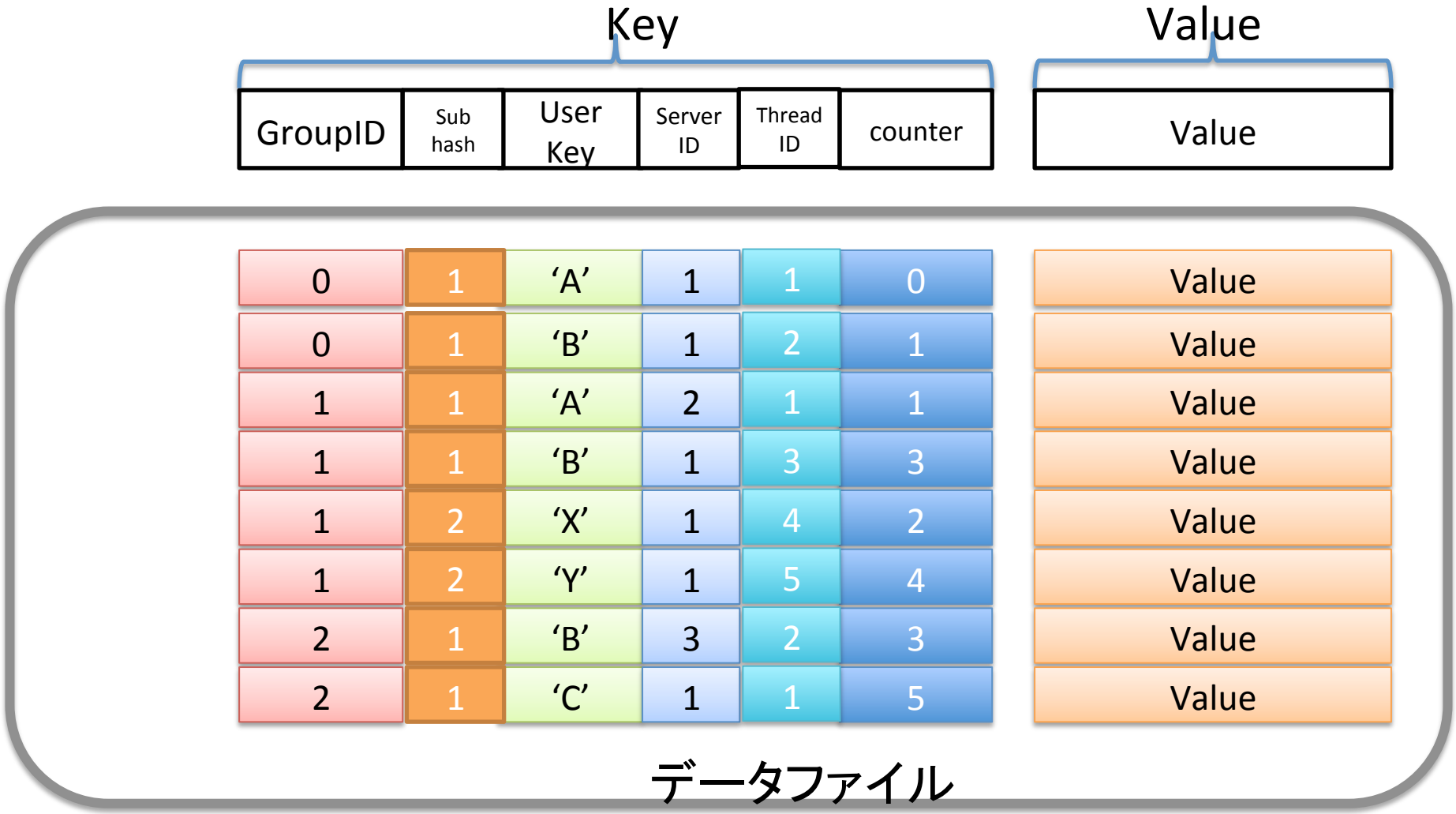
- パイプライン的に転送
  - ブロック分割なし
- Java Nativeのクライアント
  - JNIのオーバヘッドなし

データベースファイルの分割

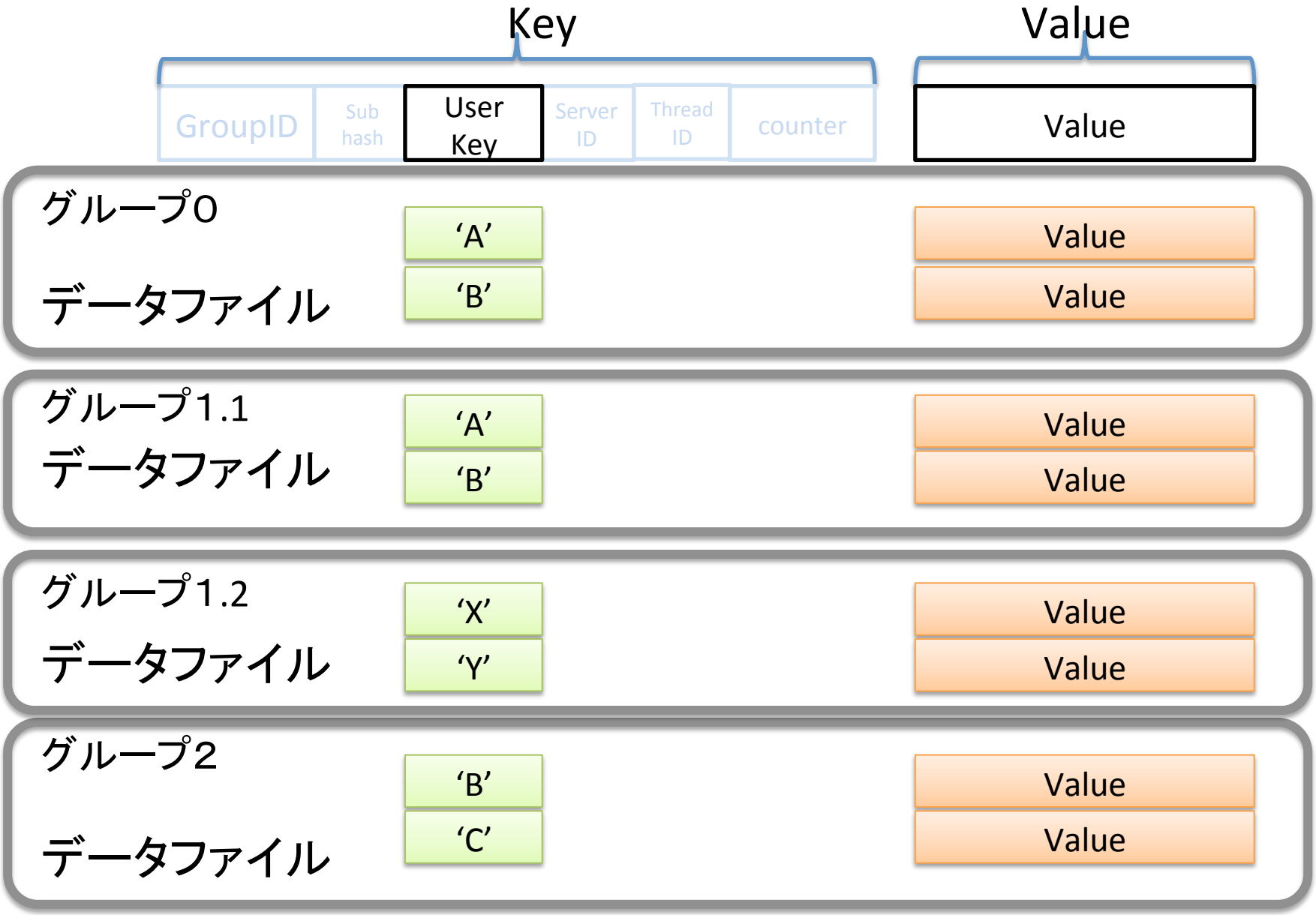
- TokyoTyrantではデータベースファイルはひとつしか持てない
- 新サーバでは複数のデータベースファイルを使い分けることが可能

- サブグループごとに、データベースファイルを割り当てる

既存実装:


Key
Value
GroupID
Sub hash
User Key
Server ID
Thread ID
counter
Value
0 1 'A' 1 1 0 Value
0 1 'B' 1 2 1 Value
1 1 'A' 2 1 1 Value
1 1 'B' 1 3 3 Value
1 2 'X' 1 4 2 Value
1 2 'Y' 1 5 4 Value
2 1 'B' 3 2 3 Value
2 1 'C' 1 1 5 Value
データファイル

提案実装


Key
Value
GroupID
Sub hash
User Key
Server ID
Thread ID
counter
Value
グループ0
データファイル
'A'
'B'
Value
Value
グループ1.1
データファイル
'A'
'B'
Value
Value
グループ1.2
データファイル
'X'
'Y'
Value
Value
グループ2
データファイル
'B'
'C'
Value
Value

評価

- マイクロベンチマークによる評価
  - 専用のクライアントを用いたリード・ライトレベルの評価
- マクロベンチマークによる評価
  - SSSレベルでのリード・ライト・シャッフル
- タプル削除による評価

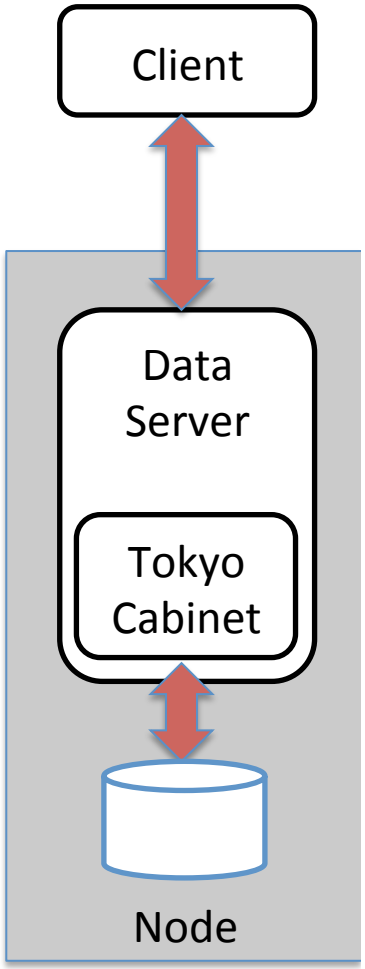
評価環境

- クラスタを使用
  - Number of nodes: 16 + 1 (master)
  - CPUs per node: Intel Xeon W5590 3.33GHz x 2
  - Memory per node: 48GB
  - OS: CentOS 5.5 x86_64
  - Storage: Fusion-io ioDrive Duo 320GB
  - NIC: Mellanox ConnectX-II 10G

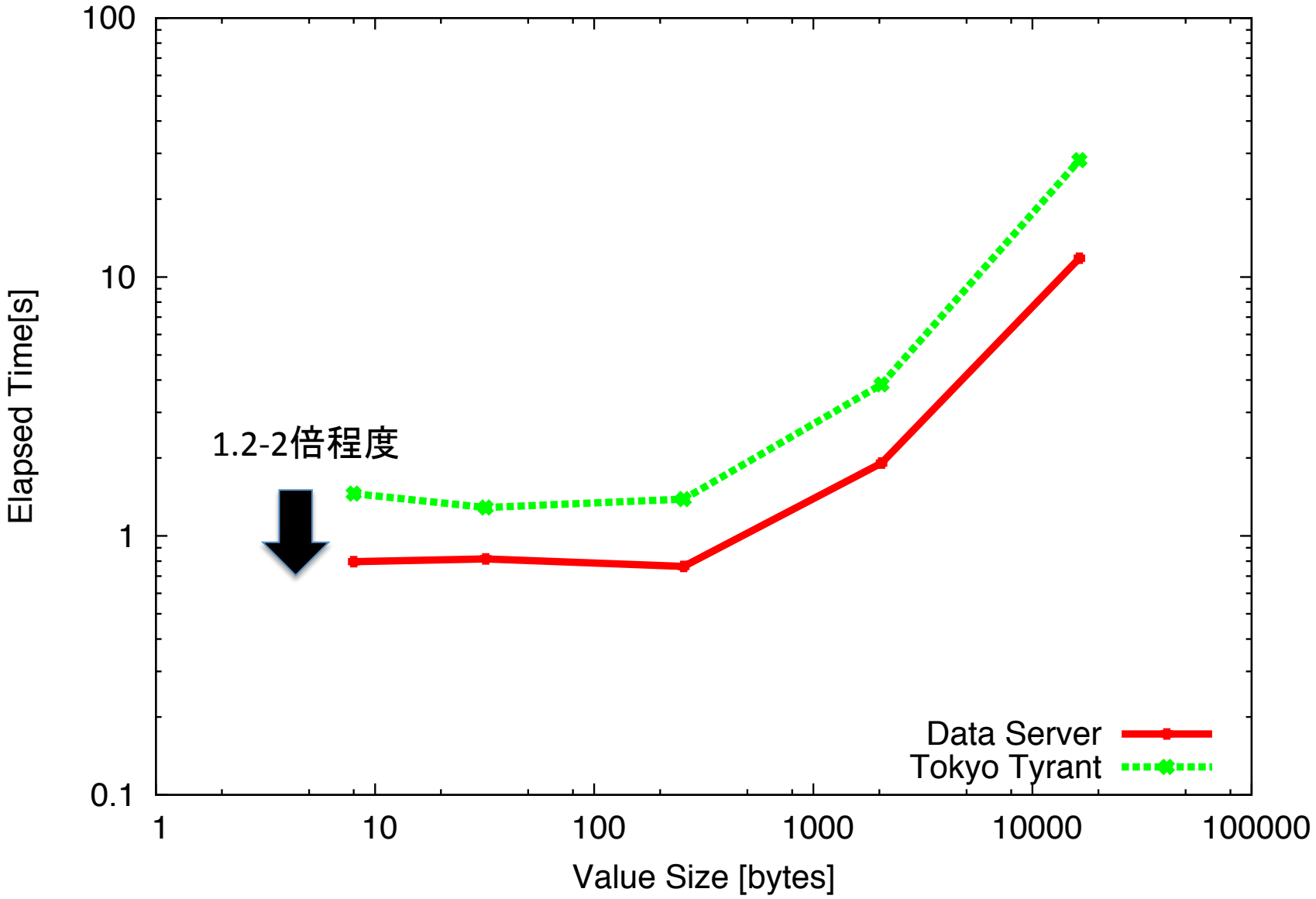
マイクロベンチマーク

- サーバの性能を測定
- 専用クライアントから1Mi個のレコードを16スレッドで読み出し

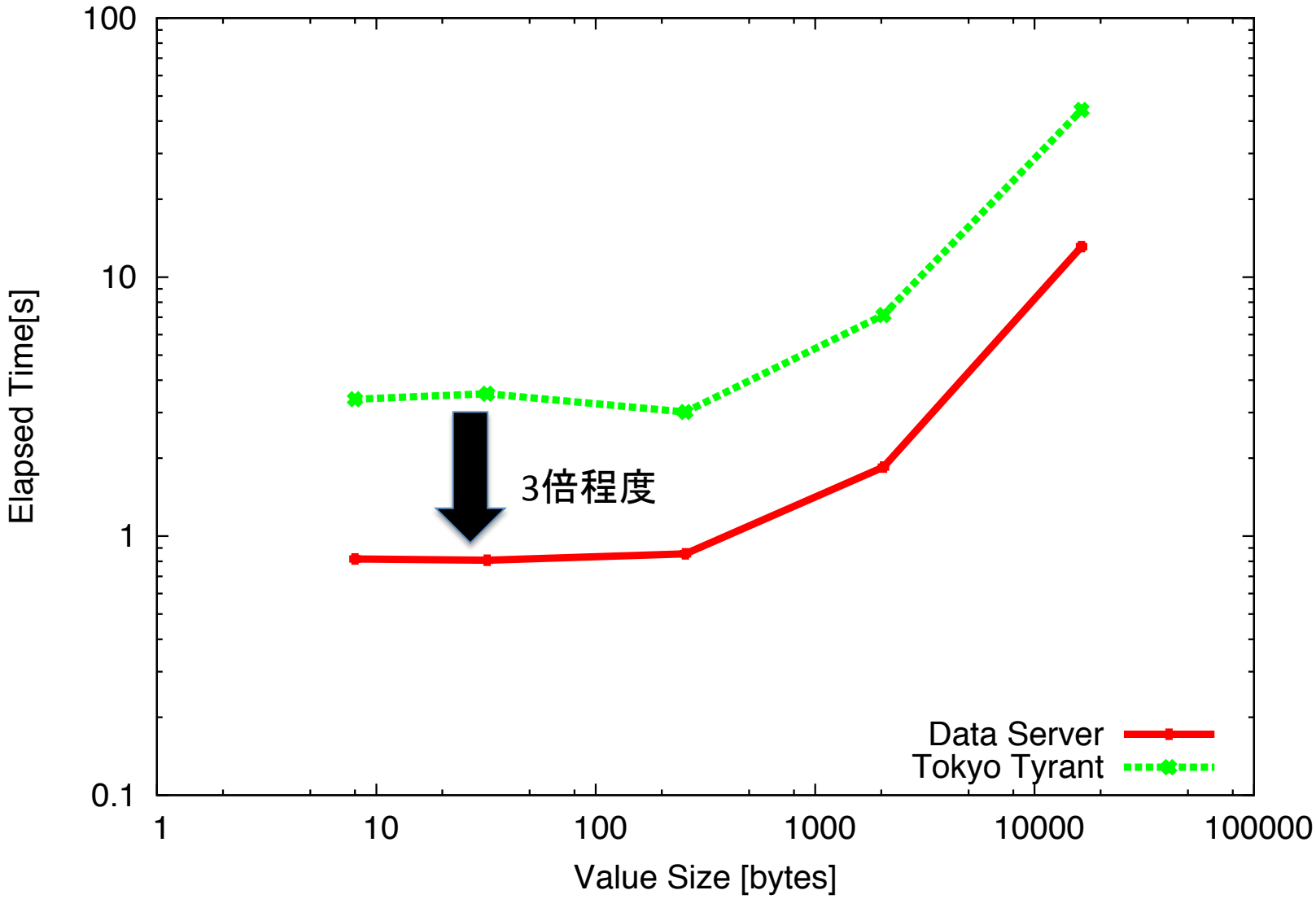
- キー：32byte
- バリュー：8, 32, 25, 2Ki, 16Ki byte



マイクロベンチマーク（リード）



マイクロベンチマーク（ライト）

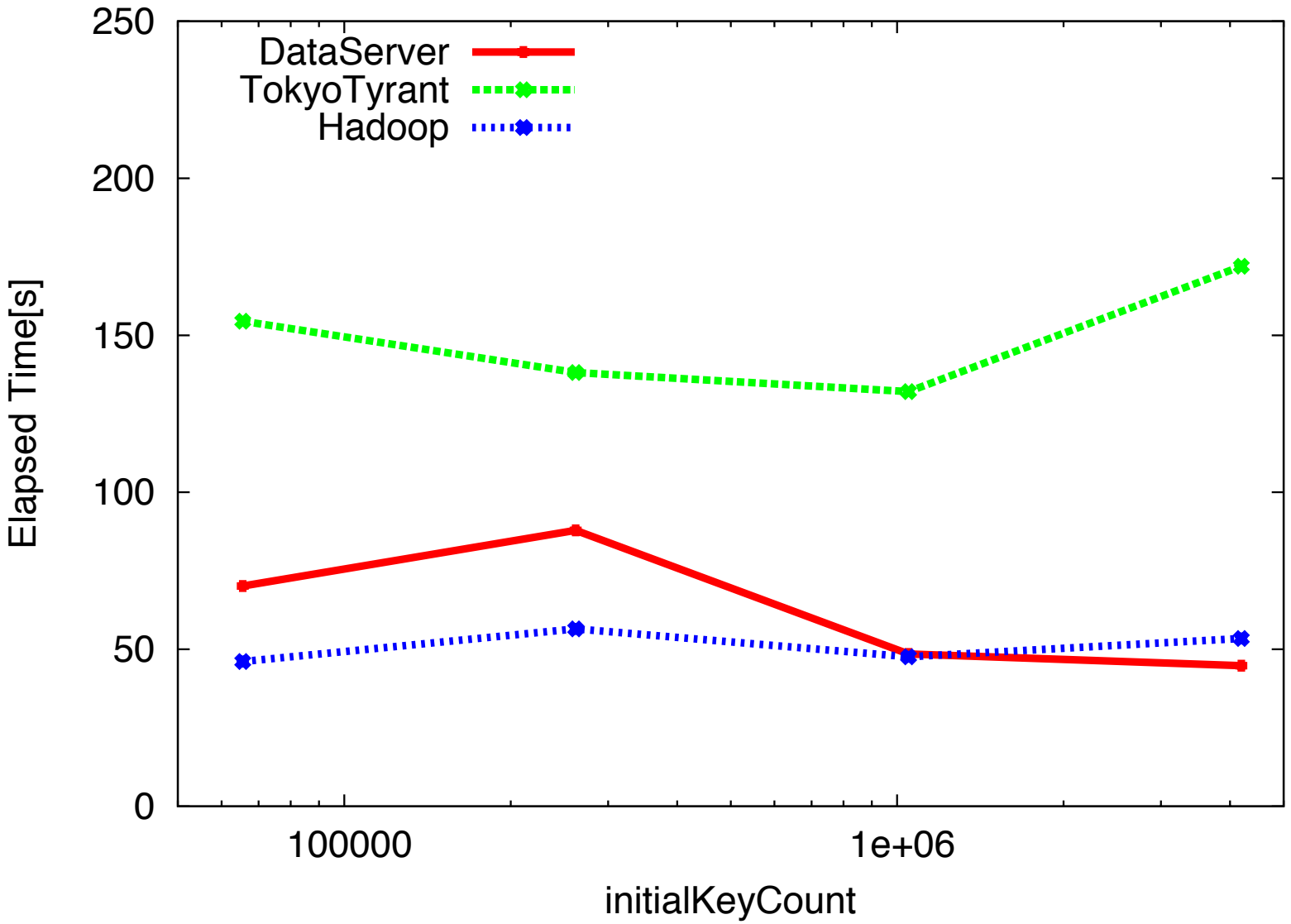


マクロベンチマーク

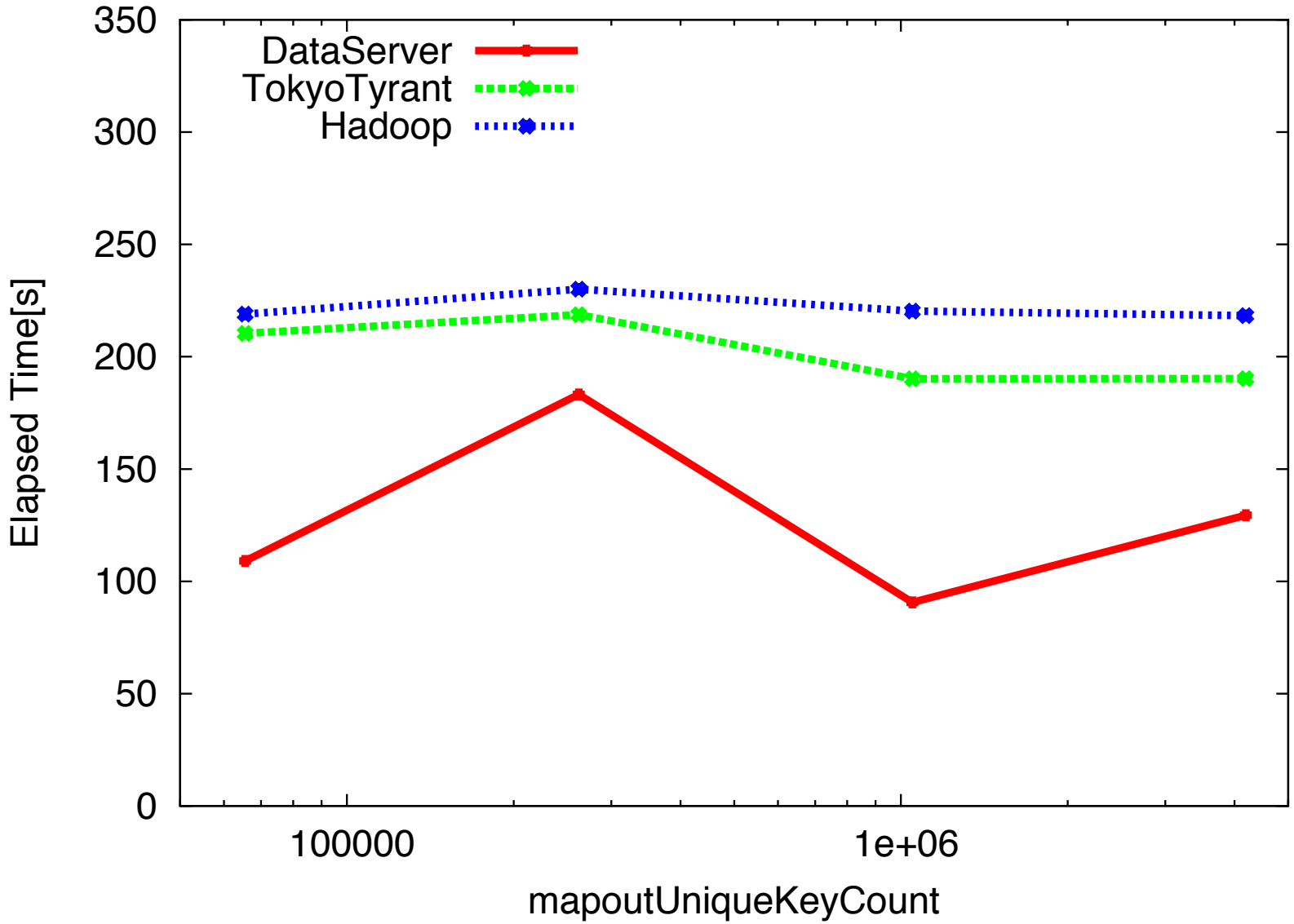
- MapReduceレベルのプログラムで計測
  [2011-HPC-130 小川]
- Read/Write/Shuffle
- Shuffle 時にはユニークなキーの数を変更して調整することが可能
  - キー数 ＝ ペア数
  - キー数 ＝ 1024
- 総計1Tバイトのデータを処理
  - 個々の値サイズと、数の積が一定

データサイズ	16MiB	4MiB	1MiB	256KiB
ペア数	64Ki	256Ki	1Mi	4Mi

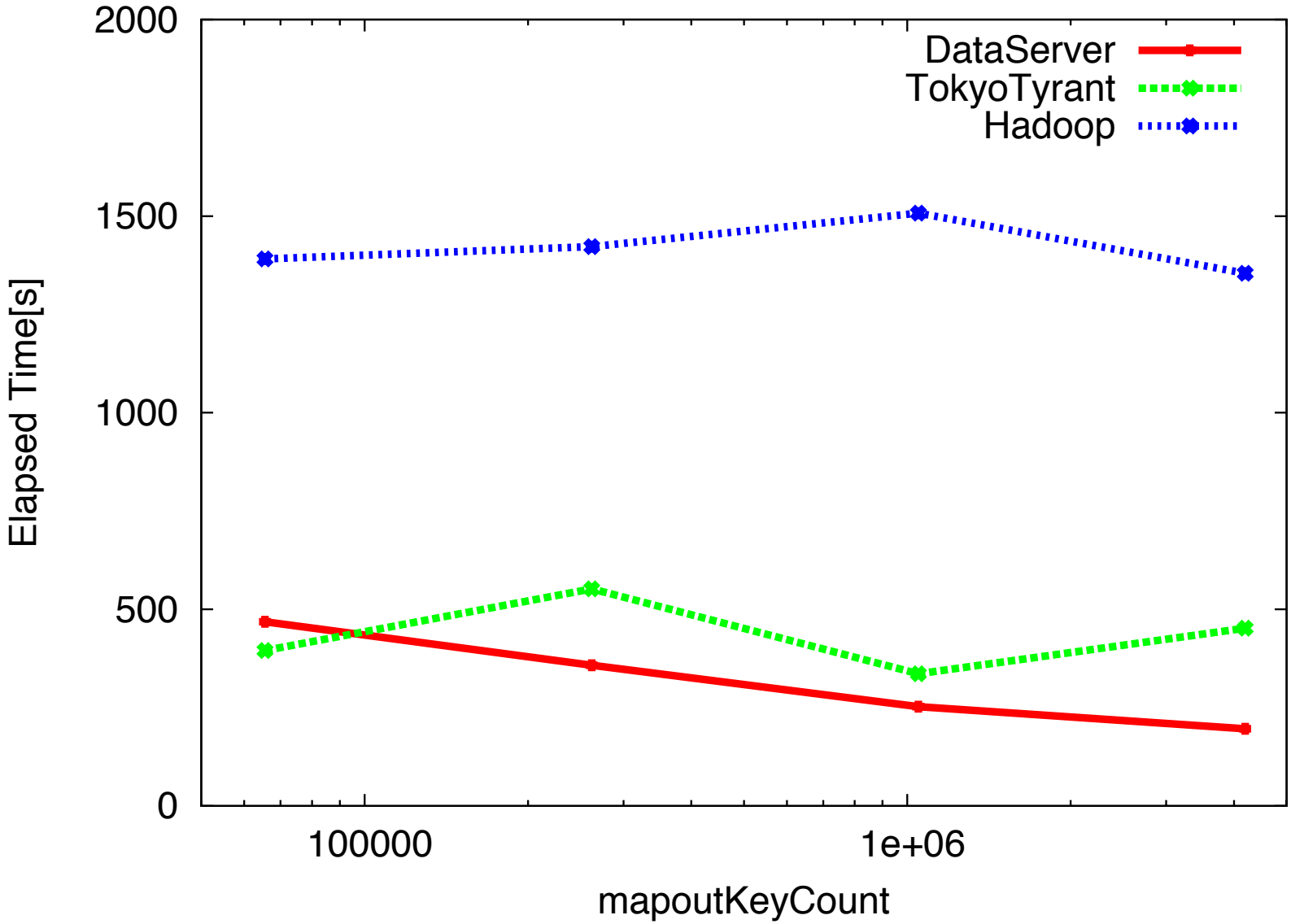
マクロベンチマーク（リード）



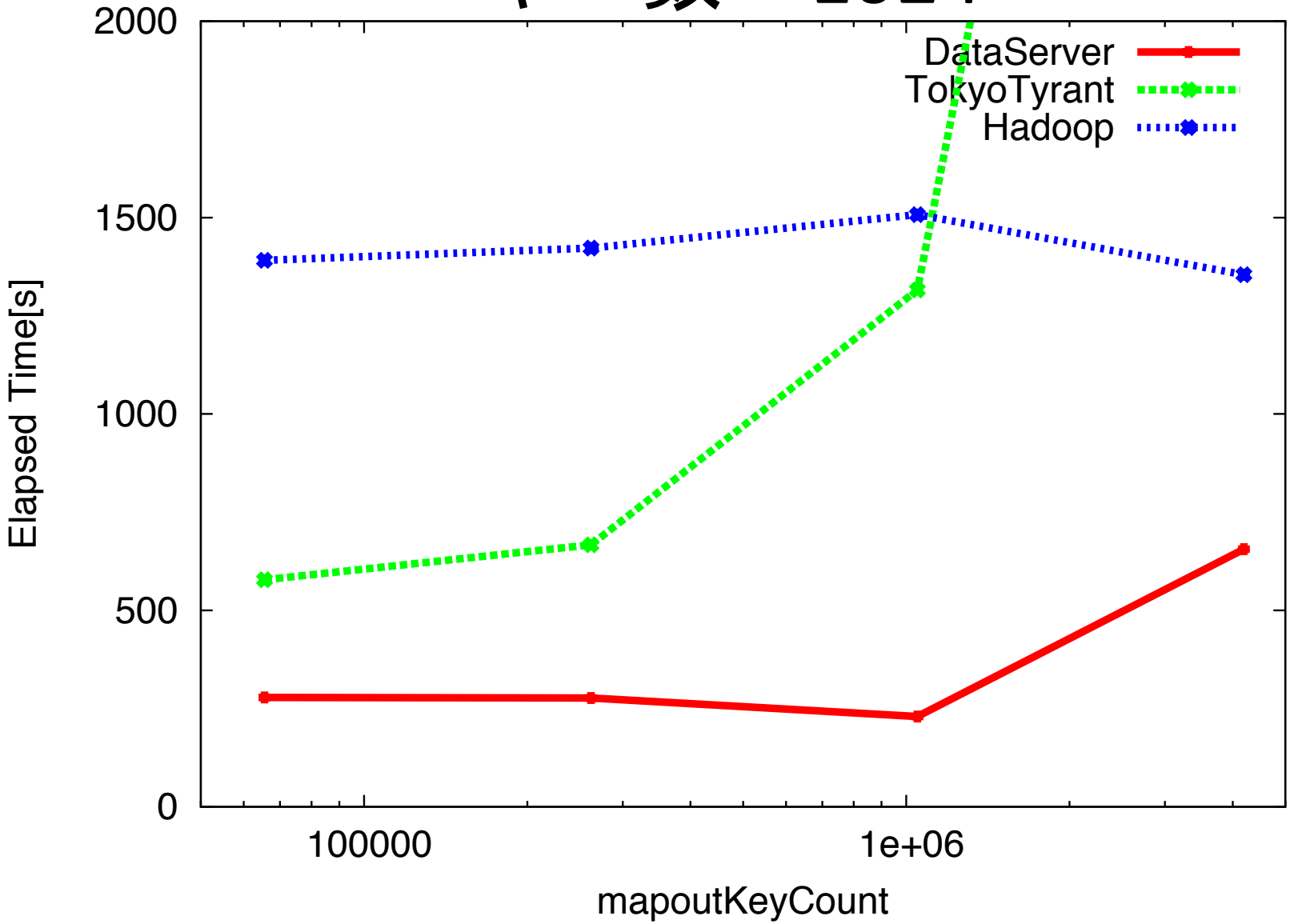
マクロベンチマーク（ライト）



マクロベンチマーク（シャッフル ）
キー数＝ペア数

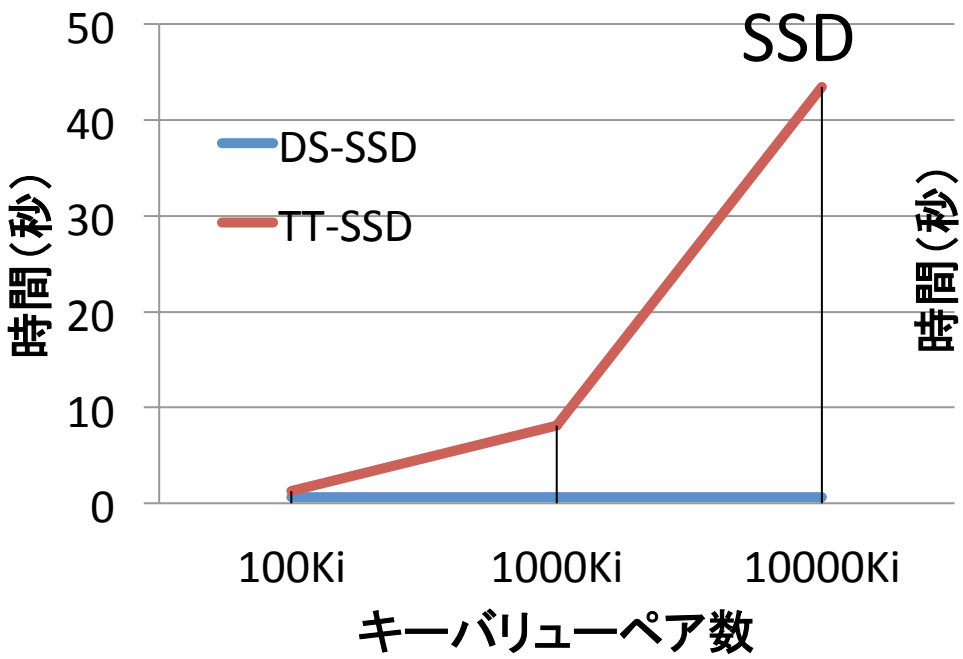


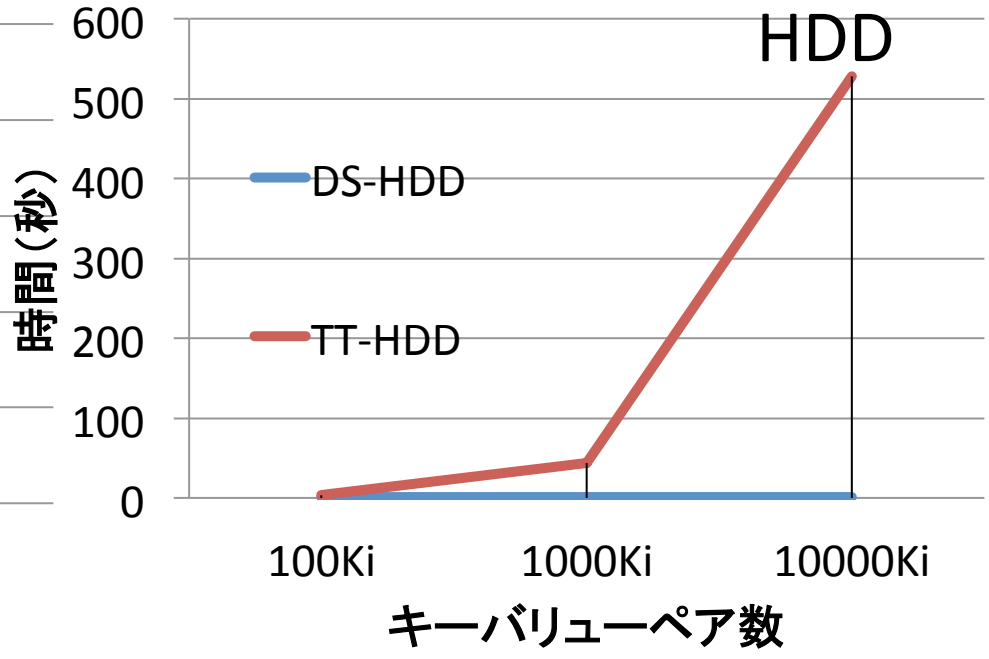
マクロベンチマーク（シャッフル ）
キー数＝1024



タプルグループ削除による評価

- TT版は一つのでデータベースファイルの中からタプルを検索して削除
  - データ量に比例したコスト
- DS版はデータベースファイルごと削除
  - コンスタント





まとめと今後の課題

- 提案
  - SSS用のKVSに接続する専用のネットワークレイヤ
- 評価
  - マイクロ・マクロベンチマークで効果を確認
  - タプルグループ削除時間でも効果を確認

- 今後の課題
  - 実アプリケーションでの評価
  - SSS本体の拡張

謝辞

- 本研究の一部は，独立行政法人新エネルギー・産業技術総合開発機構（NEDO）の委託業務「グリーンネットワーク・システム技術研究開発プロジェクト（グリーンITプロジェクト）」の成果を活用している

ありがとうございました