合成ベンチマークによる MapReduce 処理系SSS の性能評価

小川宏高、中田秀基、工藤知宏
独立行政法人産業技術総合研究所

背景

- MapReduce の普及
  - Apache Hadoopの普及による
- MapReduce 処理系
  - 大規模データ処理 - Hadoop
    - 繰り返し実行が低速
  - 共有メモリオンメモリデータ処理 – Phoenix, Metis
    - シングルノード、もしくはSMPが対象
    - メモリサイズが問題を制約
  - SSS [Ogawa, MapReduce11]: 両者の間をねらう
    - クラスタ上で動作
    - ディスクを利用 ― メモリ量の制限なし
    - 高速な実行

研究の目的

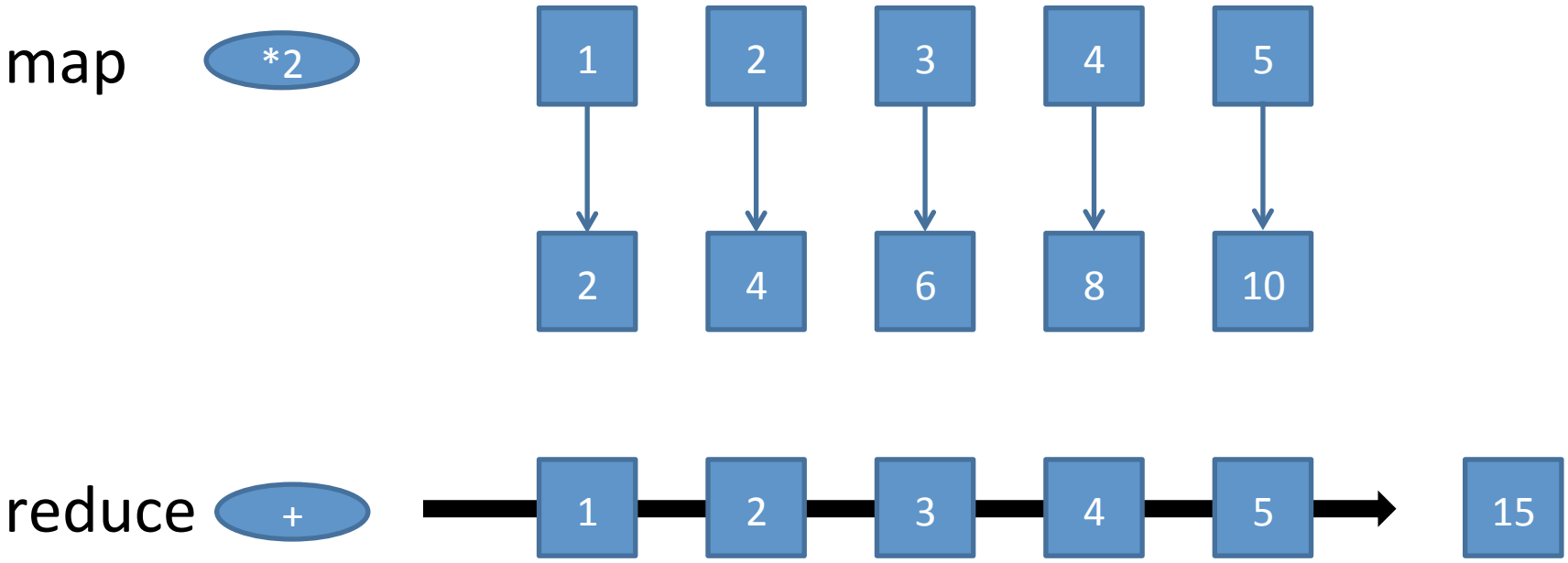
- SSS の性能を計測
  - Hadoopと比較
  - 合成ベンチマーク[小川: HPC129] を利用
    - Read/Write/Shuffle 各フェイズでの動作特性
  - K-means（予稿にはありません）
    - アプリケーションでの動作特性

発表の概要

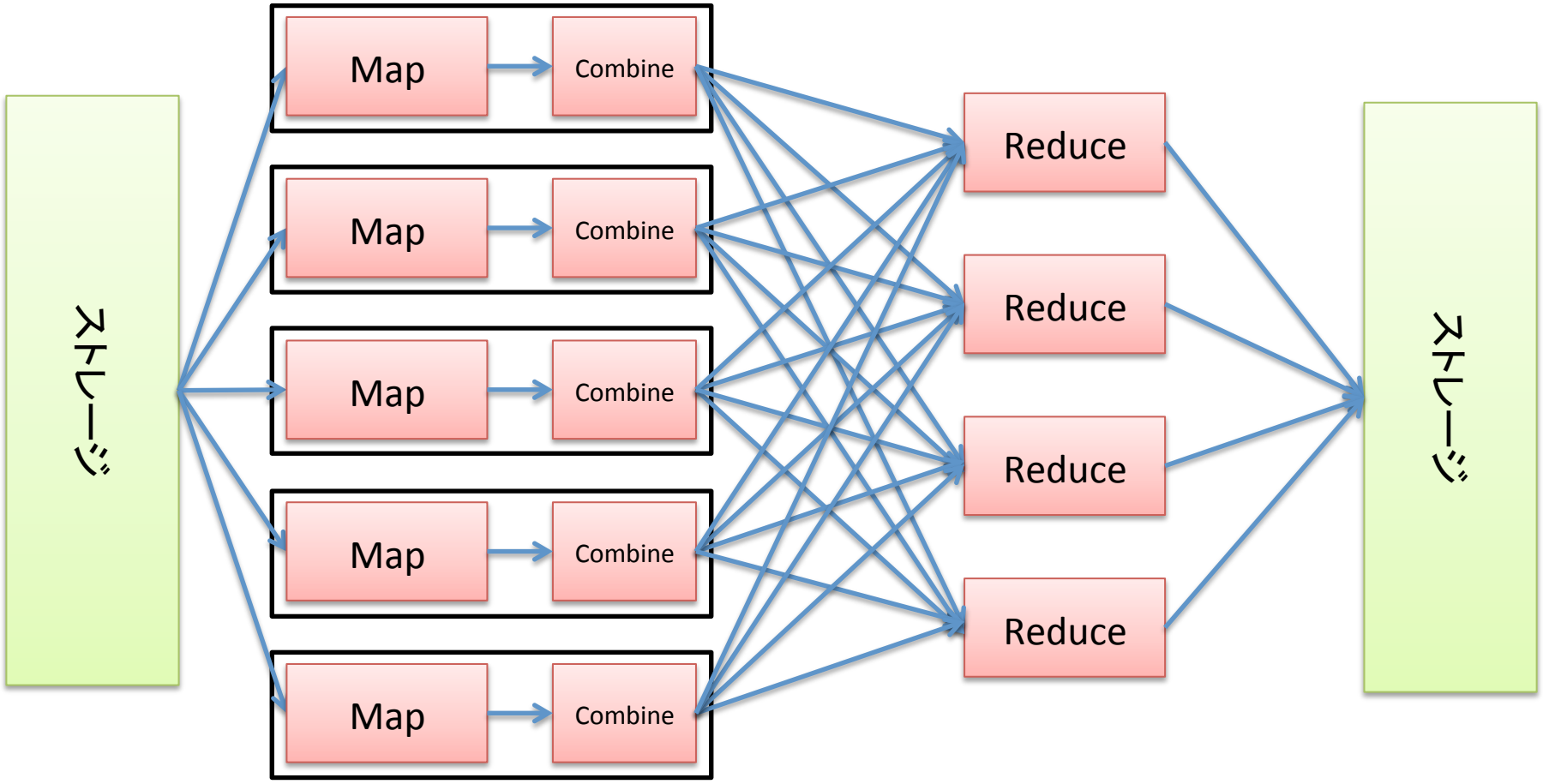
- MapReduce / Hadoop の概要
- SSSの概要
- 合成ベンチマークによる評価
- K-meansによる評価
- まとめと今後の課題

MapReduceとは

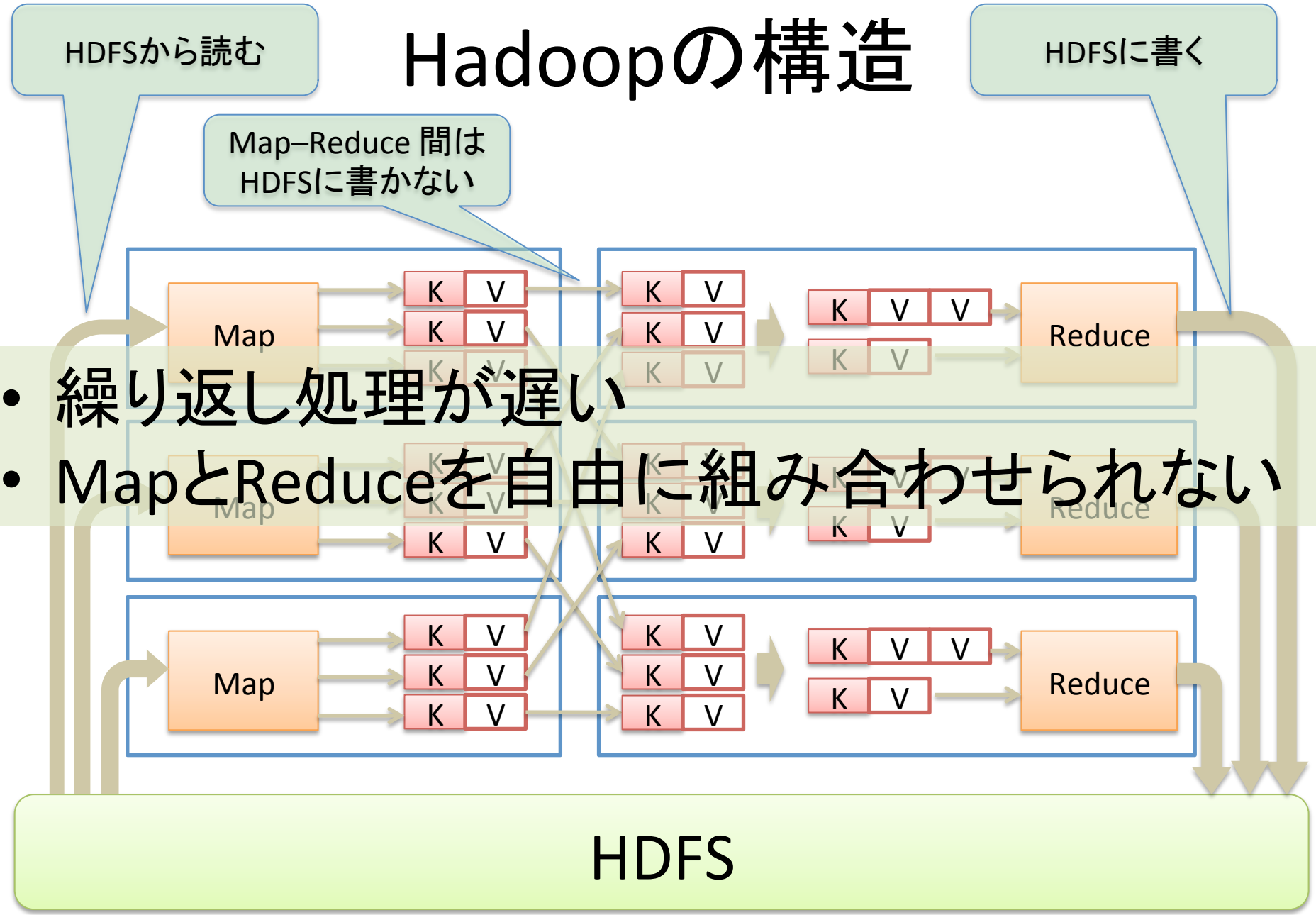
高階関数を持つ言語に一般的なmapとreduce関数にヒント



MapReduceの概要



Hadoopの構造

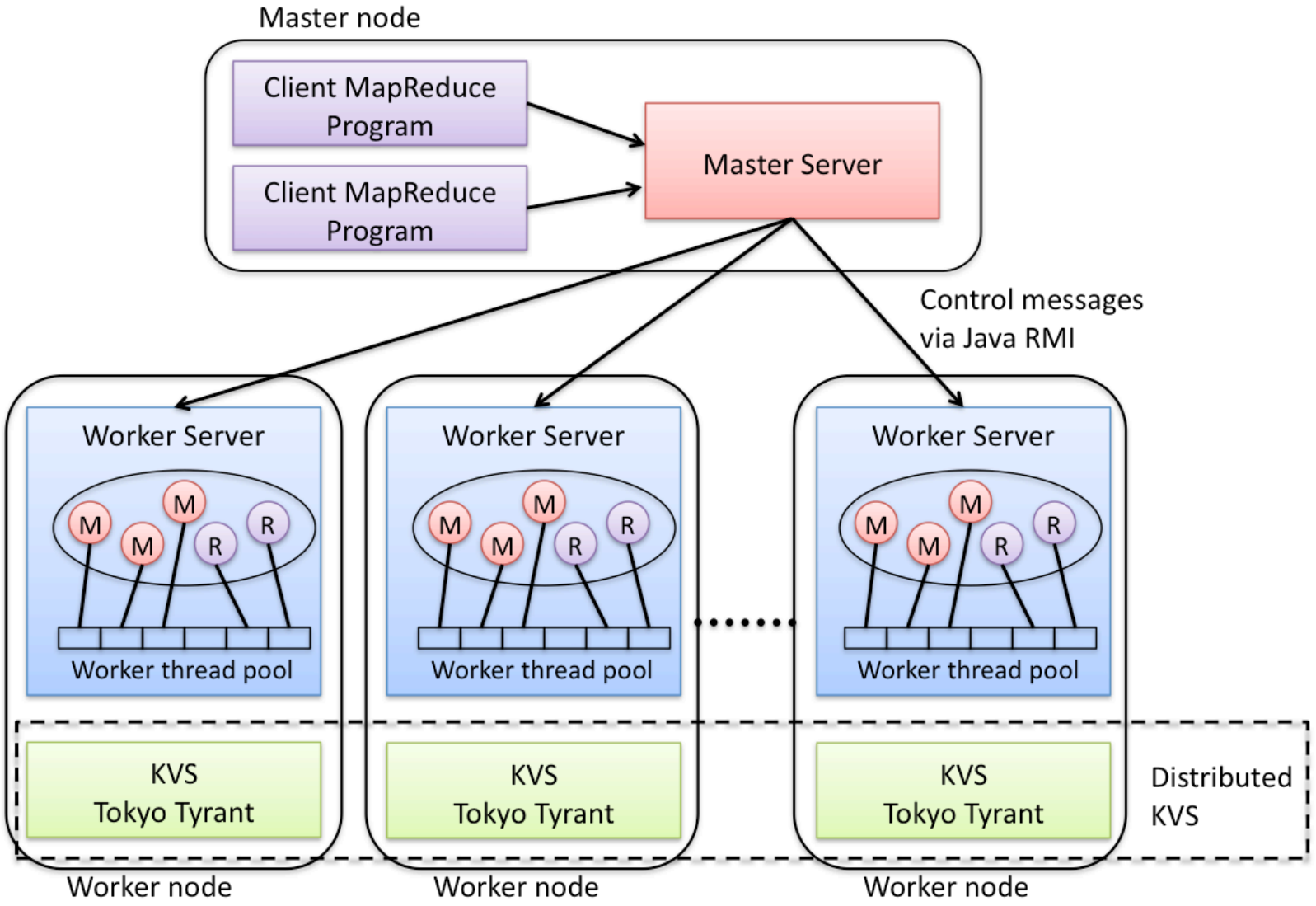


- 繰り返し処理が遅い
- MapとReduceを自由に組み合わせられない

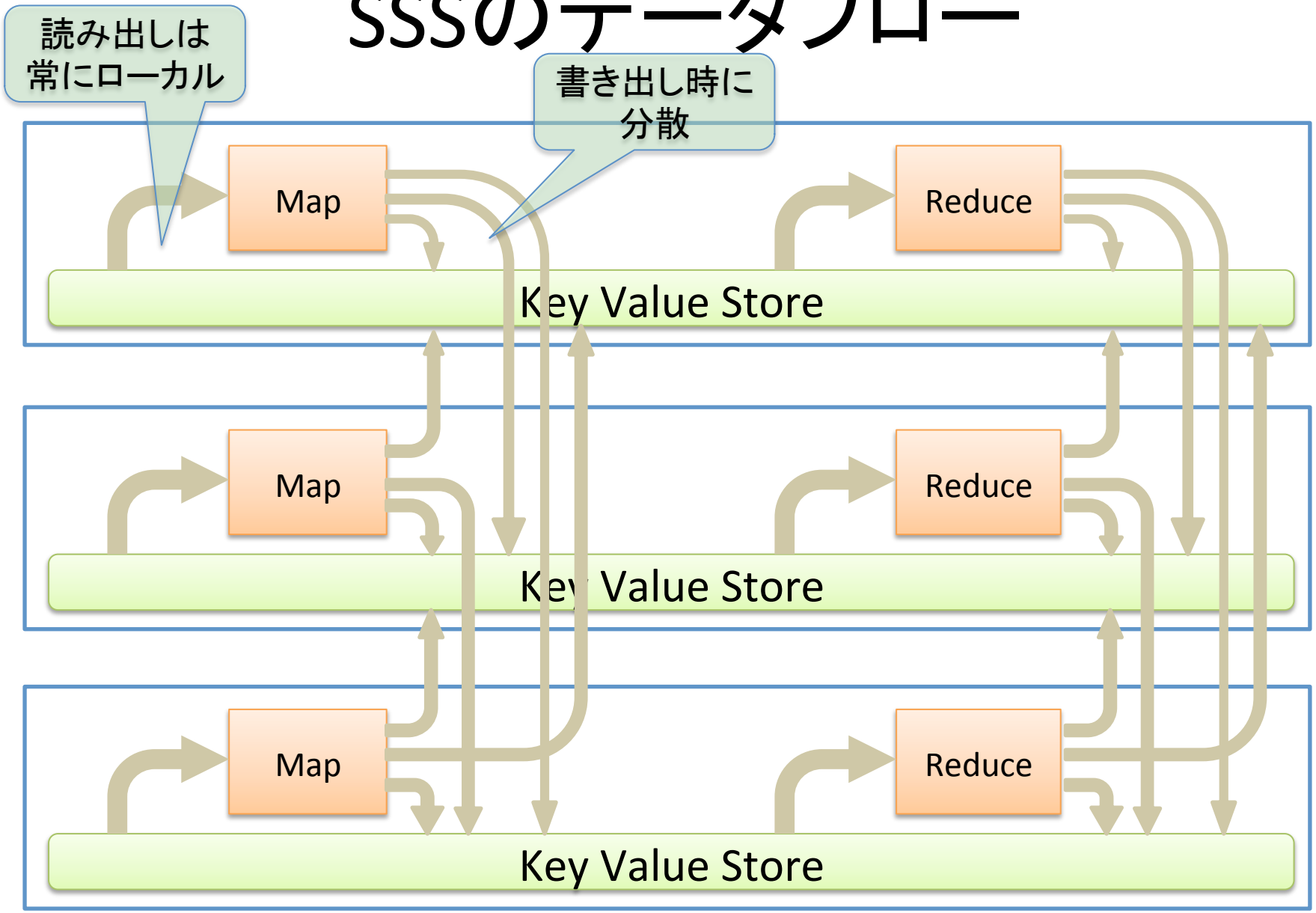
SSS

- 分散ファイルシステムではなく分散KVSをベースに
  - Map後にも書き出すので、MapとReduceを自由に組み合わせることが可能
  - シャッフルのフェイズをハッシング＋KVS内でのB+treeによるソートで代用
- Owner Computes Rule
  - データをハッシュで分散
  - データがある場所でMap / Reduce
  - すべてのノードで一斉に実行するためスケジューリングコストが安価
- 分散KVSには、Tokyo Tyrant をハッシュで分散したものを利用
  - ソート済みデータのバルク読み出し・書き込みに特化してTokyo Cabinetを修正

SSSの構成

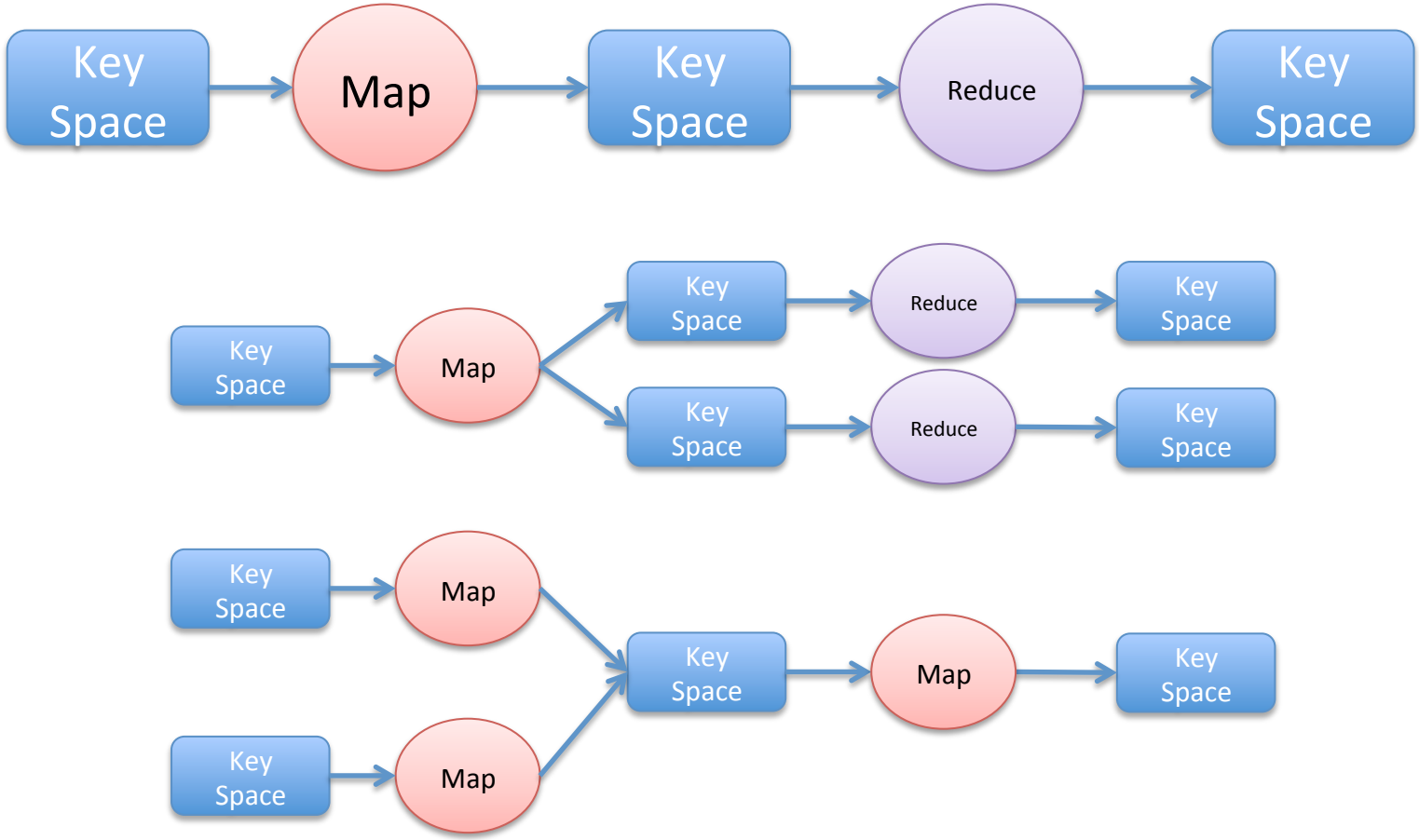

Master node
Client MapReduce Program
Client MapReduce Program
Master Server
Control messages via Java RMI
Worker Server
M
M
M
R
R
Worker thread pool
Worker Server
M
M
M
R
R
Worker thread pool
Worker Server
M
M
M
R
R
Worker thread pool
KVS Tokyo Tyrant
KVS Tokyo Tyrant
KVS Tokyo Tyrant
Distributed KVS
Worker node
Worker node
Worker node

SSSのデータフロー


読み出しは
常にローカル
書き出し時に
分散
Map
Reduce
Key Value Store
Map
Reduce
Key Value Store
Map
Reduce
Key Value Store

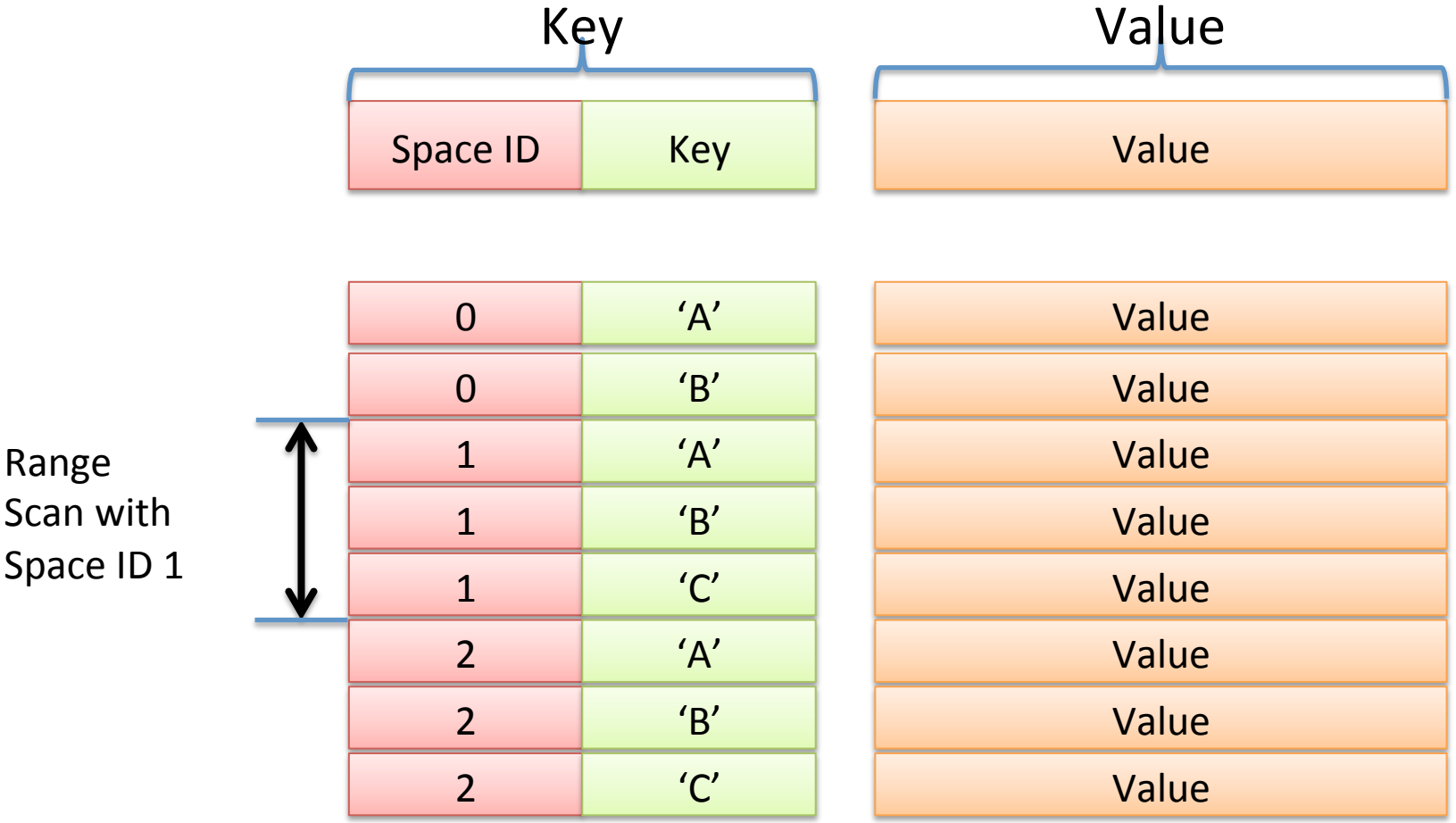
Key Space

- Key Space – 処理対象のデータ集合
- Map と Reduce の処理は殆ど同じ
- 自由に組み合わせることが可能



Key Spaceの実装

- キーのプレフィックスとして Key SpaceのIDを付与
- Range ScanでKey Space をスキャン

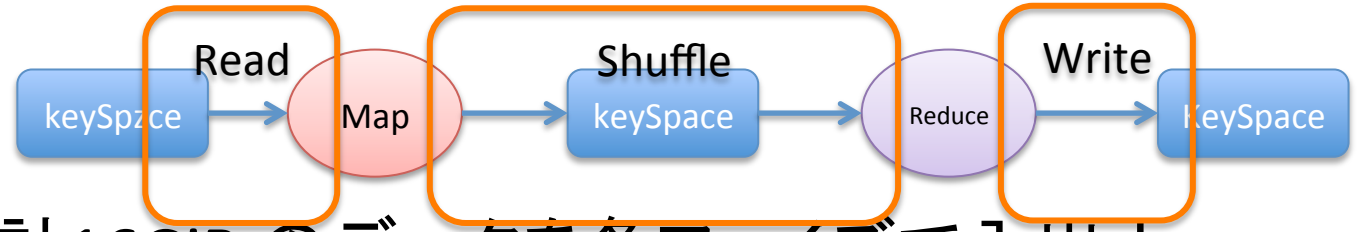


評価

- SSS の性能を計測
  - Hadoopと比較
  - 合成ベンチマーク[小川: HPC129] を利用
    - Read/Write/Shuffle 各フェイズでの動作特性
  - K-means（予稿にはありません）
    - アプリケーションでの動作特性

合成ベンチマークの構成

- 読み込み、書き出し、シャッフルのデータ数、データ量を独立して制御することが可能



- 総計16GiB のデータを各フェイズで入出力
  - 個々のKVペアのサイズを変更、個数で総データ量を調整

サイズ	1GiB	256MiB	64MiB	16MiB	4MiB	1MiB	256KiB
個数	16	64	256	1Ki	4Ki	16Ki	64Ki

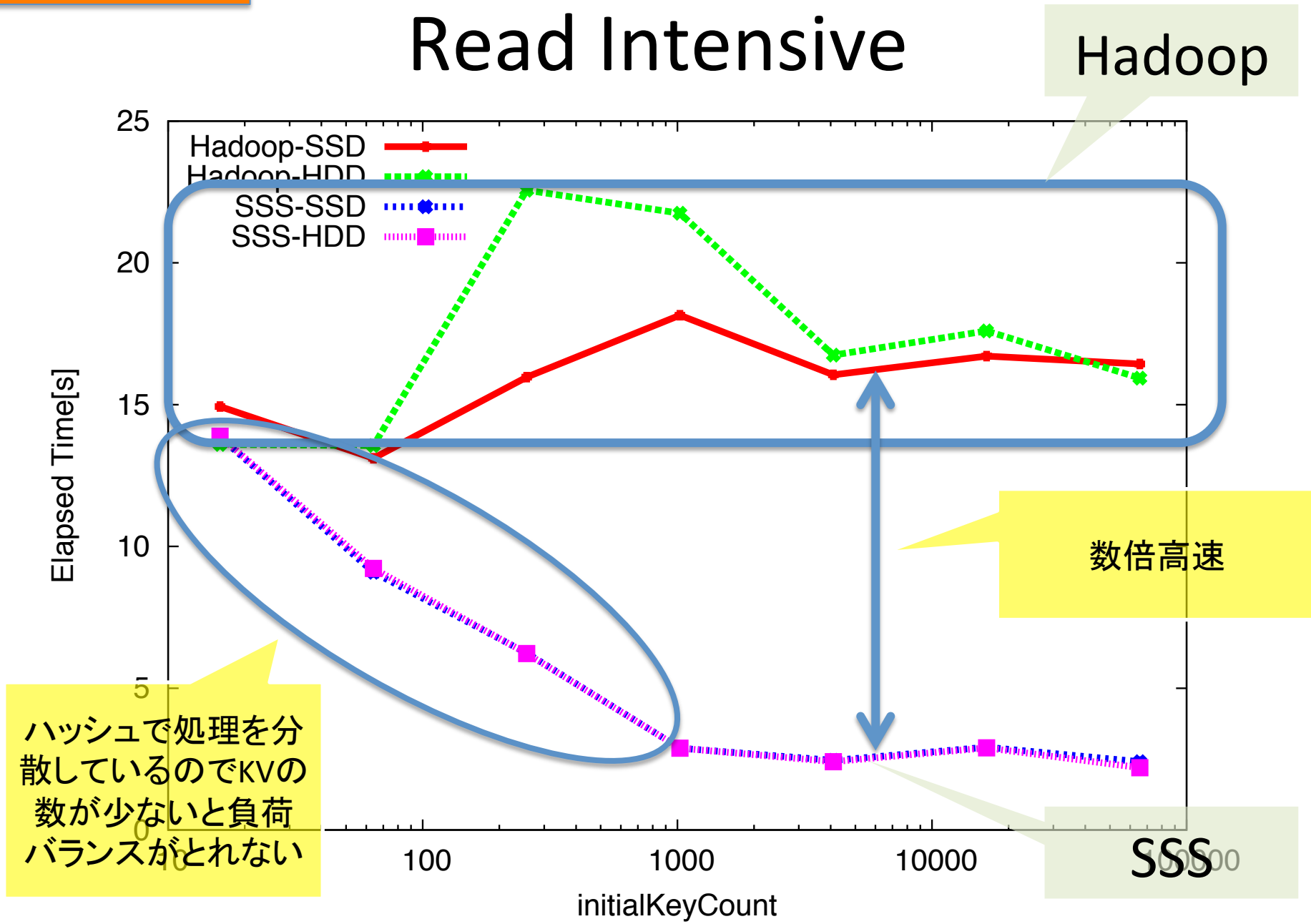
- Hadoop版
  - 1ファイルに64MiB
  - 予稿では1KV＝1ファイルとしたが、性能低下が著しかったため修正

評価環境

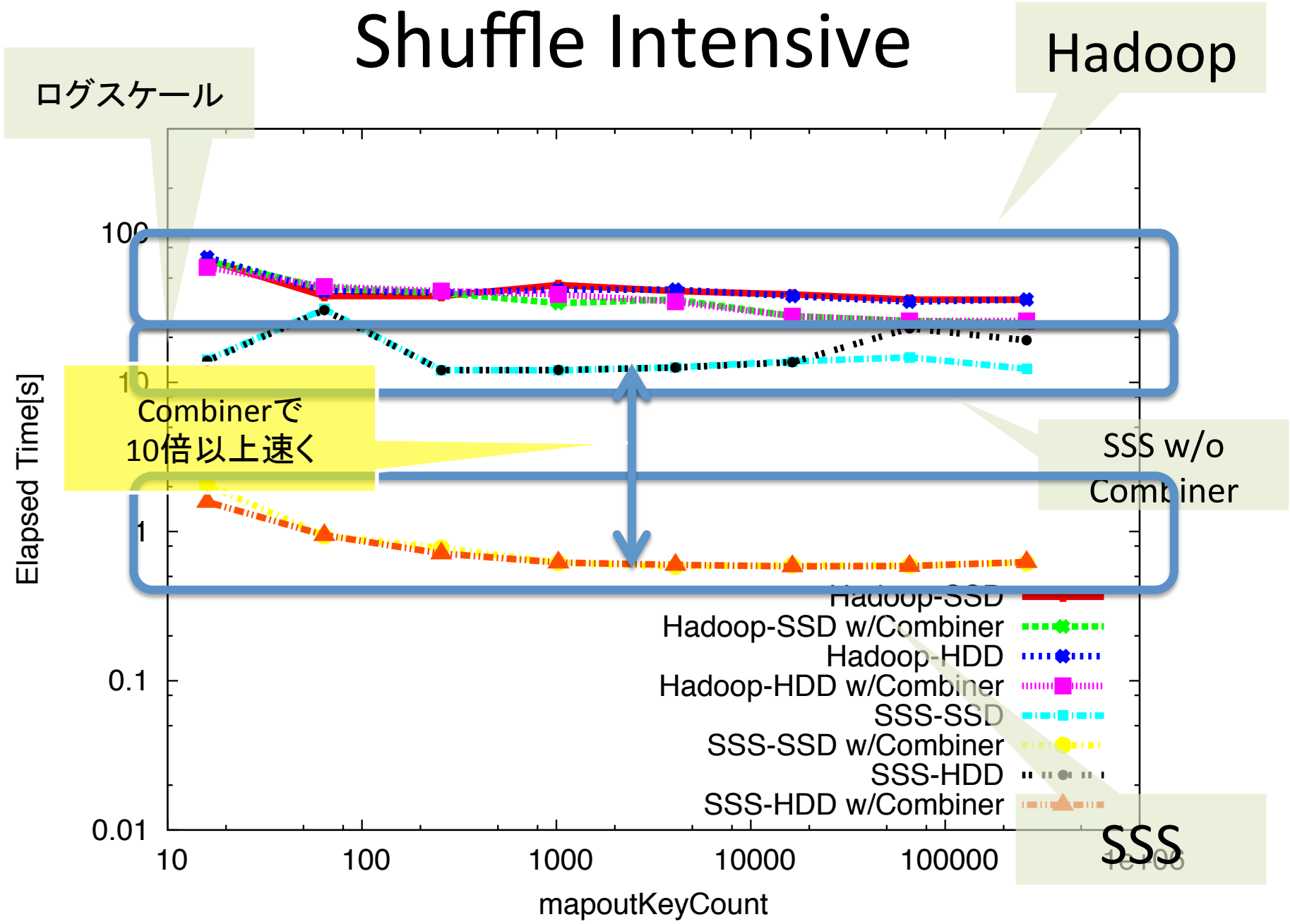
- クラスタを使用
  - Number of nodes: 16 + 1 (master)
  - CPUs per node: Intel Xeon W5590 3.33GHz x 2
  - Memory per node: 48GB
  - OS: CentOS 5.5 x86_64
  - Storage: Fusion-io ioDrive Duo 320GB
  - NIC: Mellanox ConnectX-II 10G
- ソフトウェア
  - SSS
  - Hadoop 0.20.2
    - HDFSレプリカ数を1に設定
    - Nodeあたりのmapper数7

予稿と違います

Read Intensive



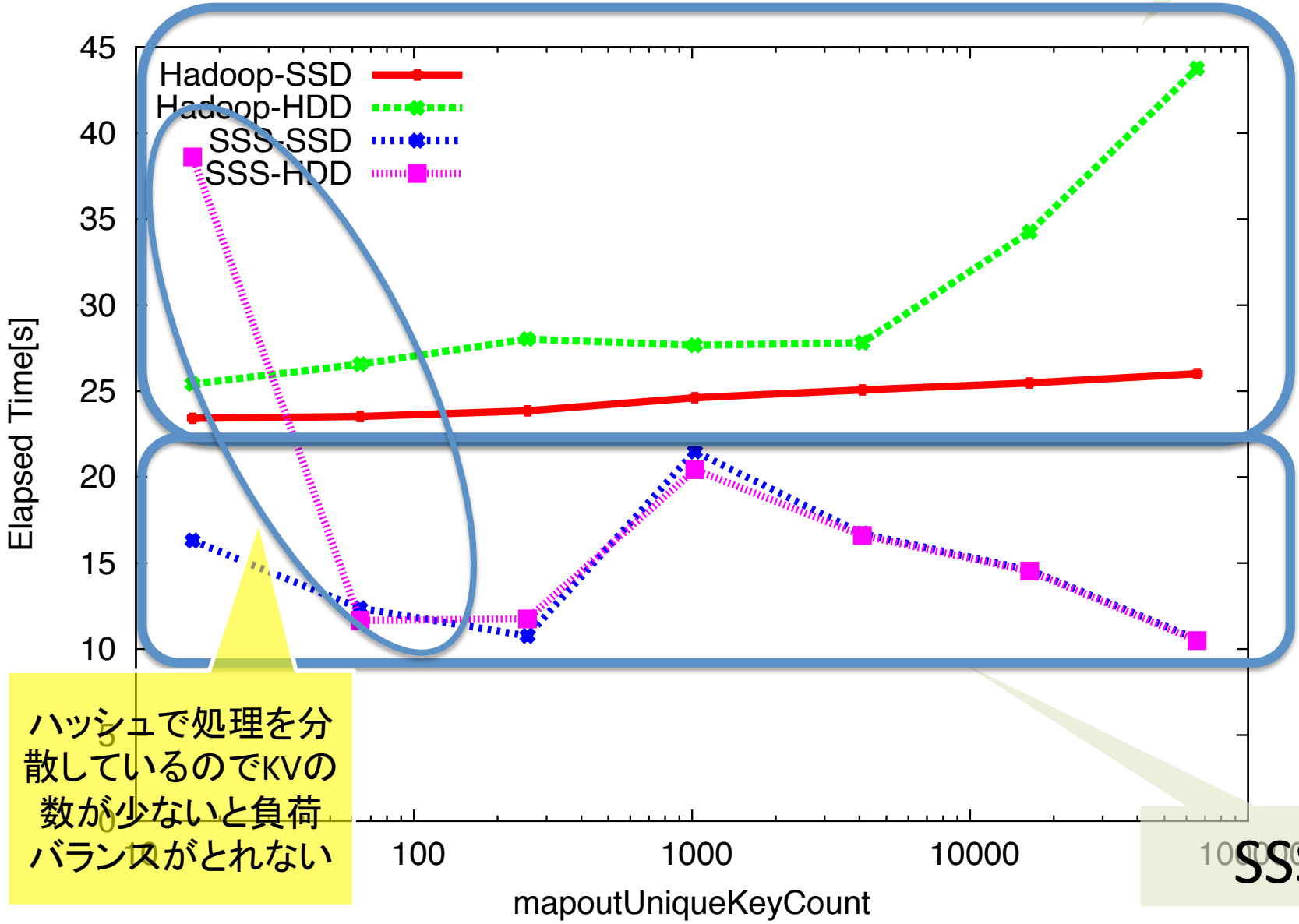
Shuffle Intensive



予稿と違います

Write Intensive

Hadoop



ハッシュで処理を分散しているのでKVの数が少ないと負荷バランスがとれない

SSS

議論

- 殆ど常にSSSのほうが高速
- キーの数が少ない場合には、データがうまくハッシュで分散されないためSSSが低速になる場合がある
- Shuffle Intensive では相違が顕著
  - 特にSSSでのcombinerの有効性が顕著

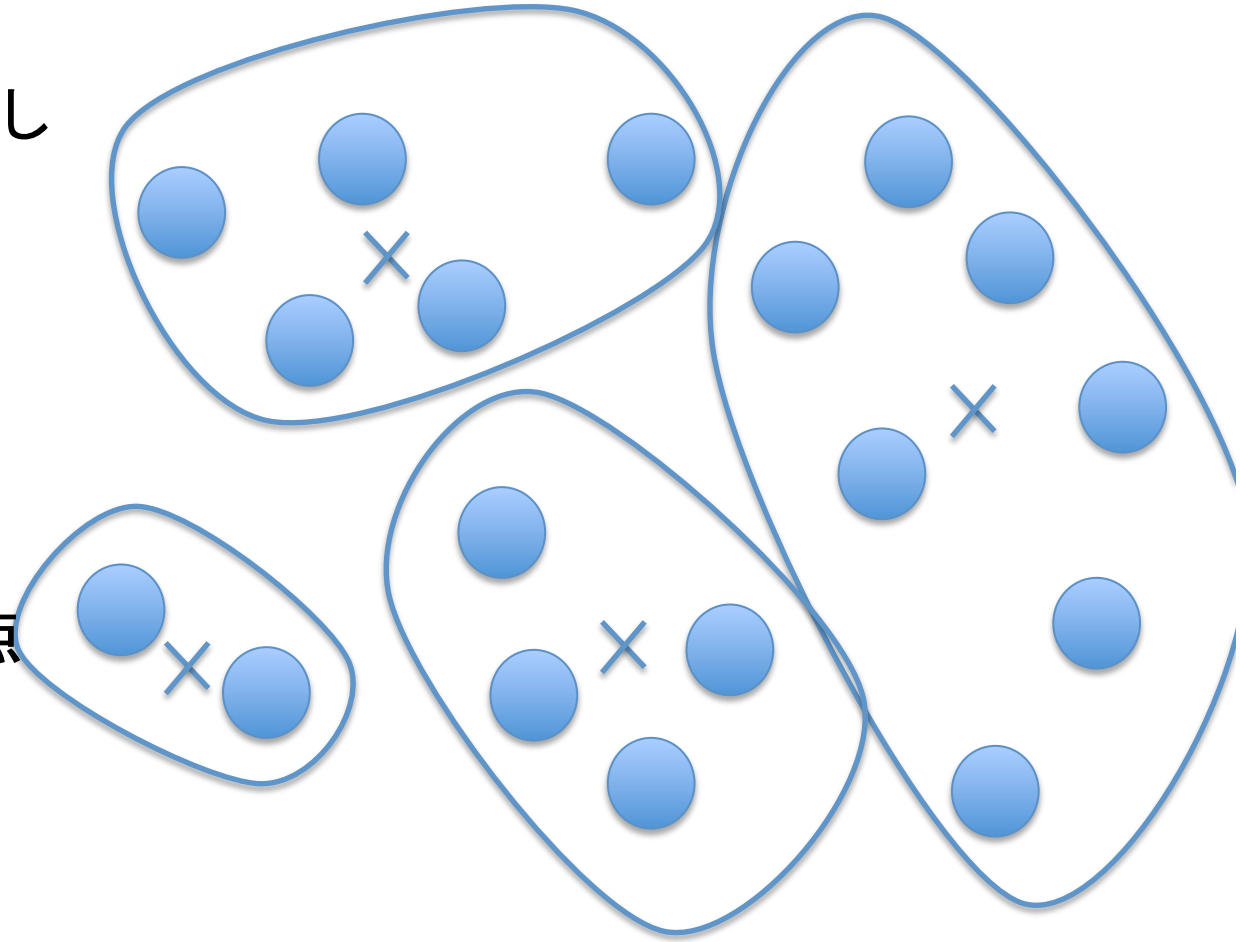
予稿にありません

K-meansによるクラスタリング

- 大容量データを繰り返しスキャン
- 重心を繰り返し更新
- 収束するまで実行

- Read Intensiveに近い

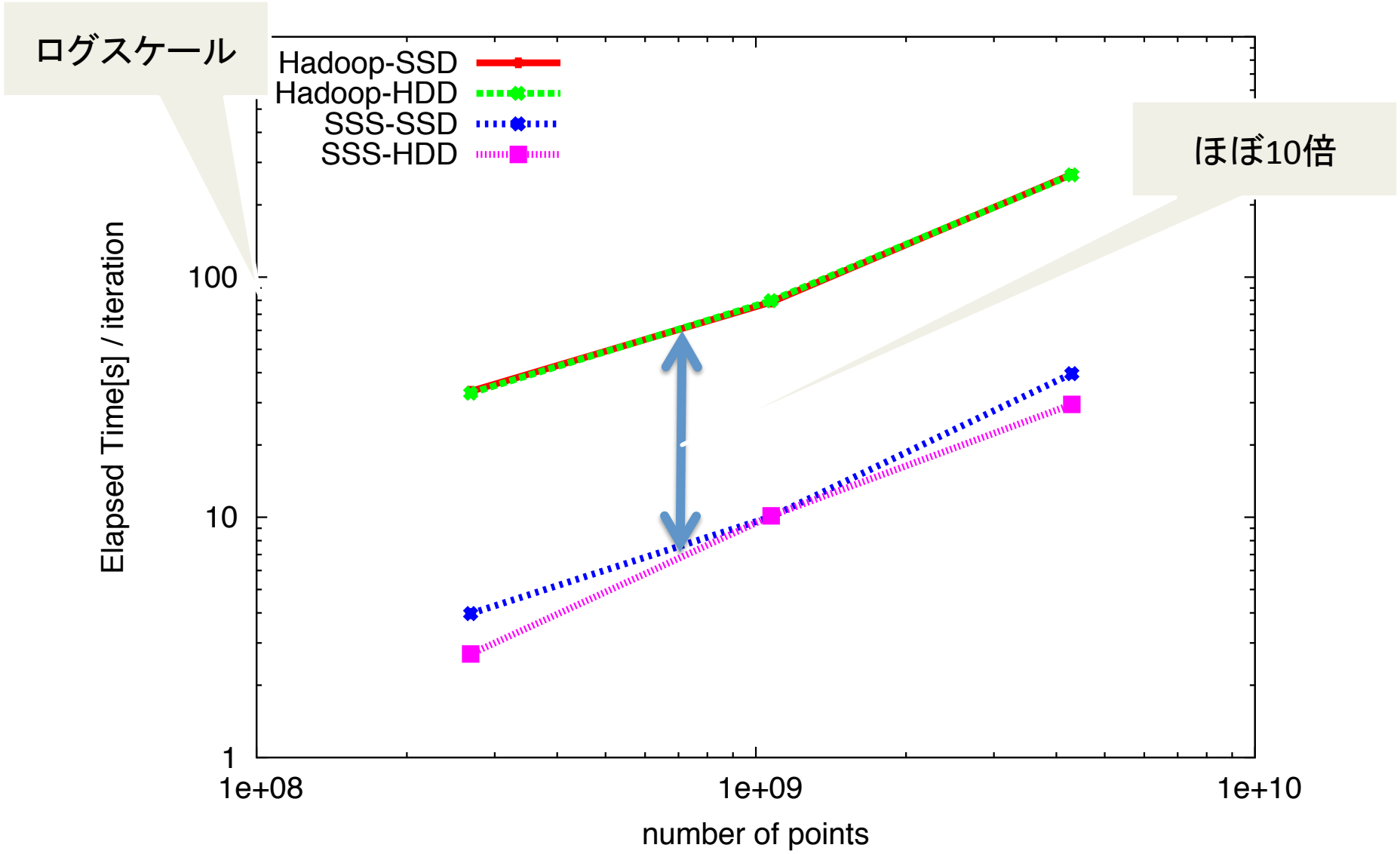
- 256Mi点、1Gi点、4Gi点を処理
- データ総量は 1GiB, 16GiB, 64GiB



K-means Clustering

予稿にありません

K-meansの結果 (iteration あたり)



まとめ

- MapReduce 処理系SSSに対して
  - 合成ベンチマークによる評価を示した
  - K-meansによる評価を示した
- Hadoopと比較して高速であることを確認
- Shuffle インテンシブジョブに関してCombinerが非常に有効に機能することを確認

今後の課題

- ベンチマーク設定の見直し
  - 1ノードあたり1GiBだとメモリに乗ってしまう
  - データ量の増加
    - 総計16GiB →1TiB 程度
- 実アプリケーションでの評価

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