MapReduce処理系SSSの実アプリケーションによる評価

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

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

研究の目的と成果

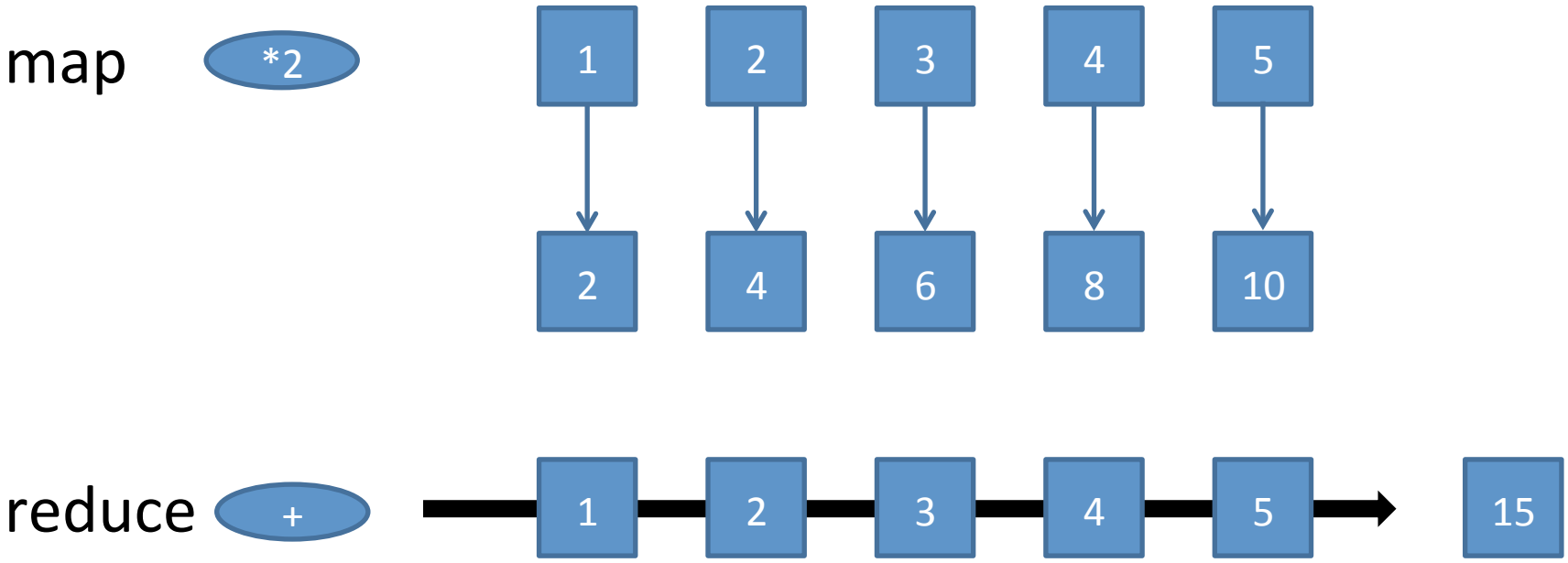
- SSS の性能を実アプリケーションで評価
  - 大量のソースコードからの系列パターン抽出
  - 大阪大学井上研究室、萩原研究室で開発されたPrefixSpan法のMapReduce実装を利用
  - Hadoopと比較

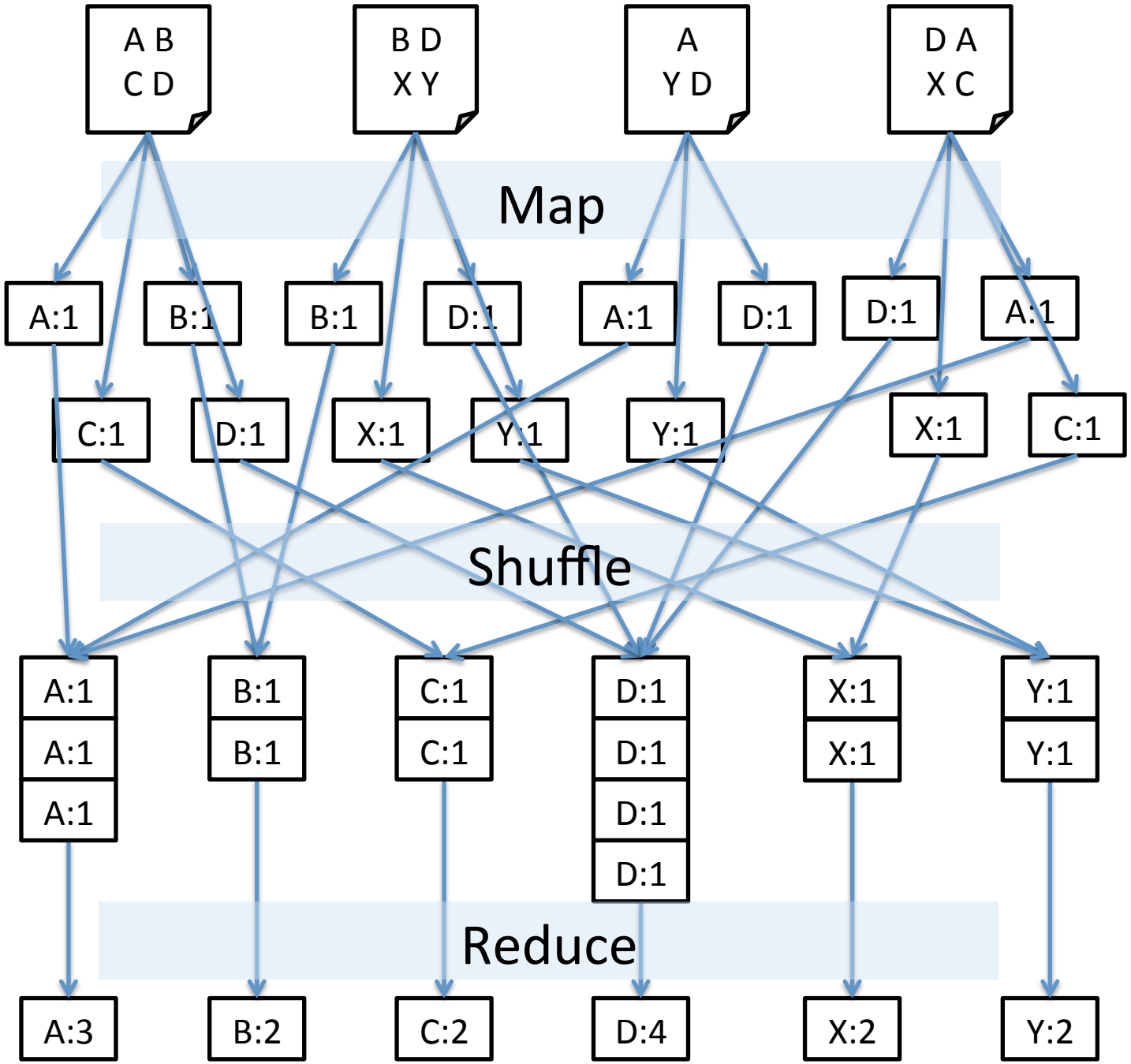
発表の概要

- MapReduce/SSSの概要
- 系列パターン抽出によるソースコード解析
- PrefixSpan 法による系列パターン抽出
  - 実装
- 評価結果
- まとめと今後の課題

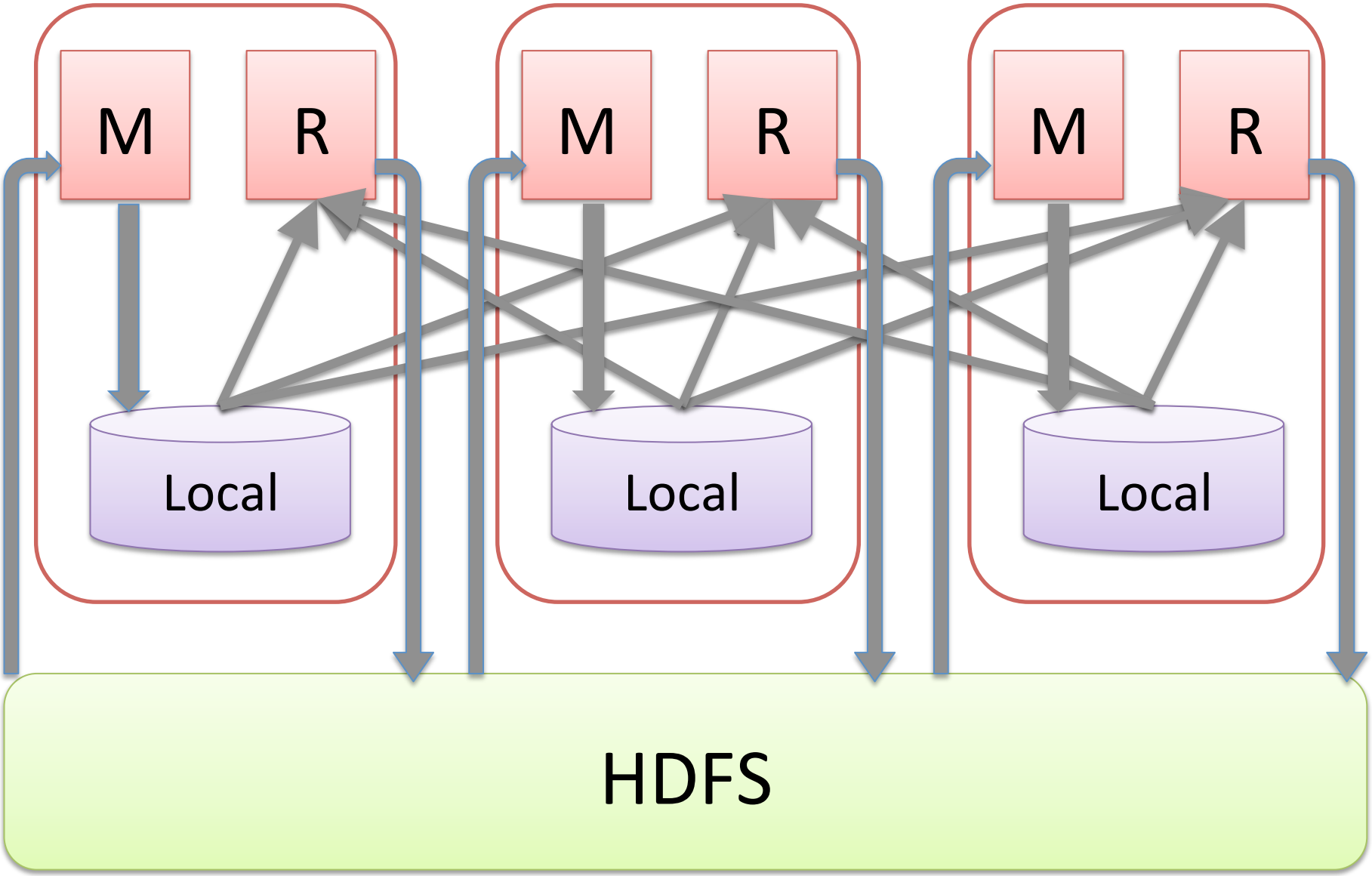
MapReduceとは

高階関数を持つ言語に一般的なmapとreduce関数にヒント




A B
C D
B D
X Y
A
Y D
D A
X C
Map
A:1
B:1
B:1
D:1
A:1
D:1
D:1
A:1
C:1
D:1
X:1
Y:1
Y:1
X:1
C:1
Shuffle
A:1
A:1
A:1
B:1
B:1
C:1
C:1
D:1
D:1
D:1
D:1
X:1
X:1
Y:1
Y:1
Reduce
A:3
B:2
C:2
D:4
X:2
Y:2

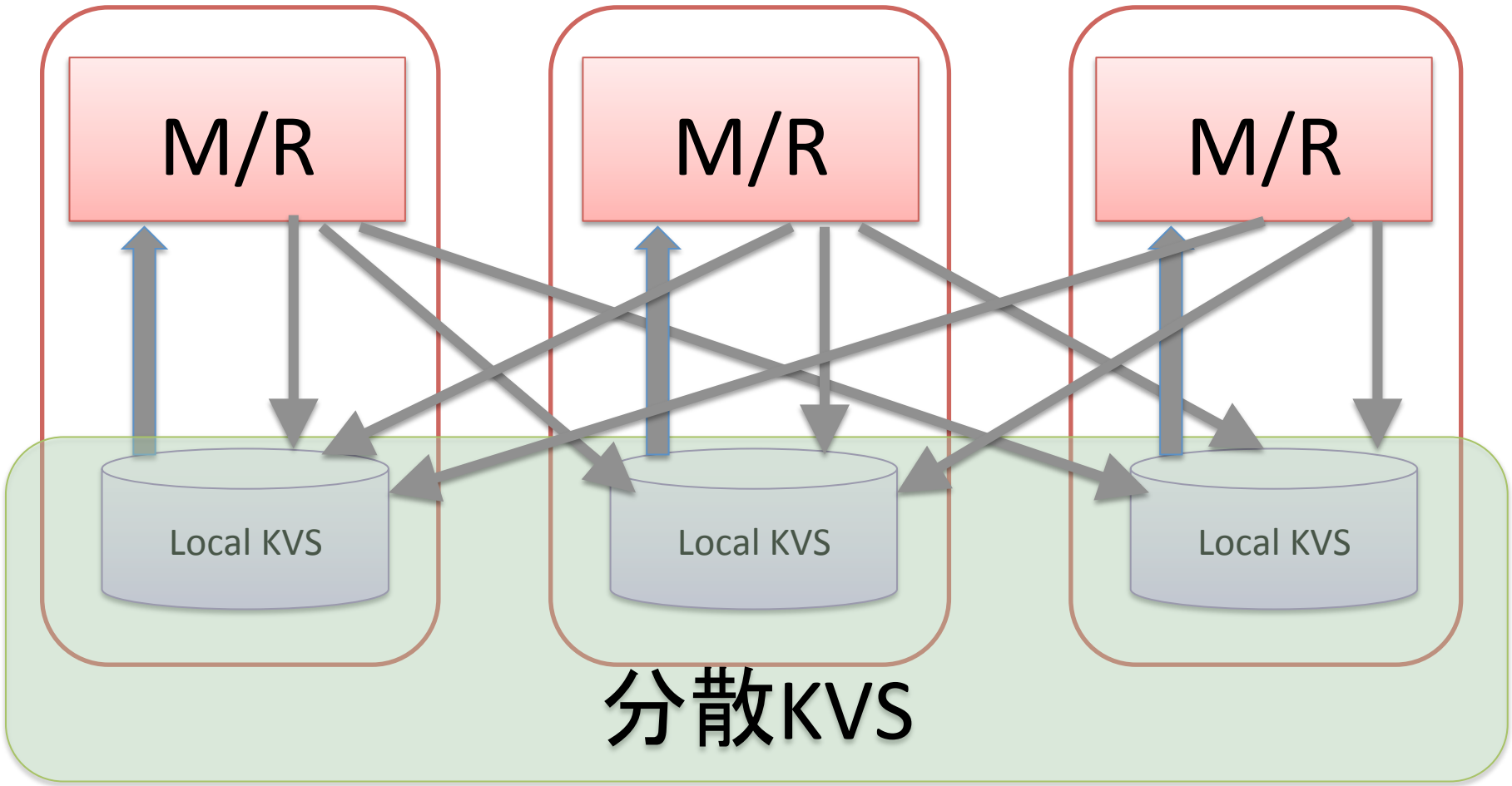
Hadoopの概要



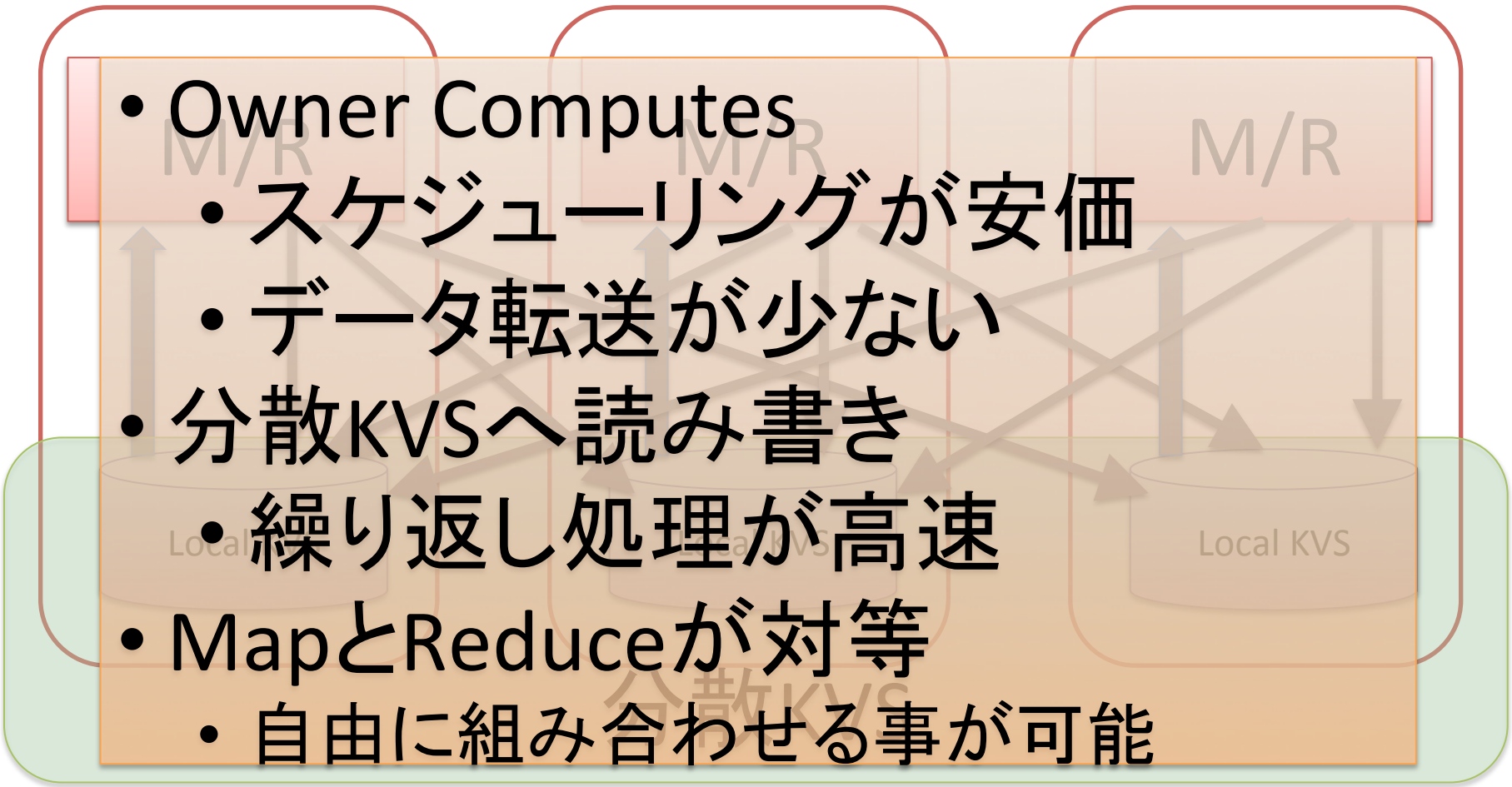
Hadoopの概要

- HDFSから読んでHDFSに書き込み
  - 繰り返し処理が遅い
- MapとReduceのデータフローが異なる
  - MapとReduceを対で使わなければならない

SSSの概要



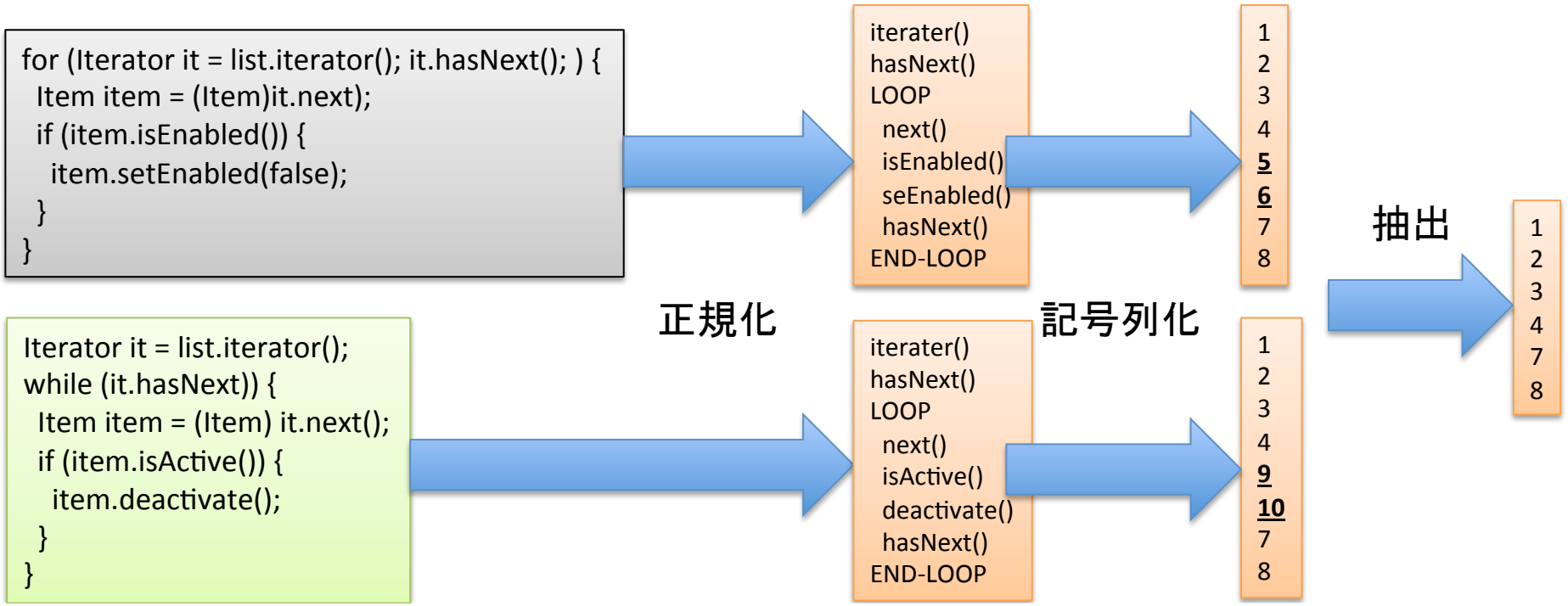
SSSの概要



- Owner Computes
  - スケジューリングが安価
  - データ転送が少ない
- 分散KVSへ読み書き
  - 繰り返し処理が高速
- MapとReduceが対等
  - 自由に組み合わせる事が可能

実アプリケーション：コーディングパターンの抽出

- プログラム中にイディオムとして出現するパターンを抽出 [伊達ら:09]
- プログラムを正規化、記号列化してPrefixSpan法で解析

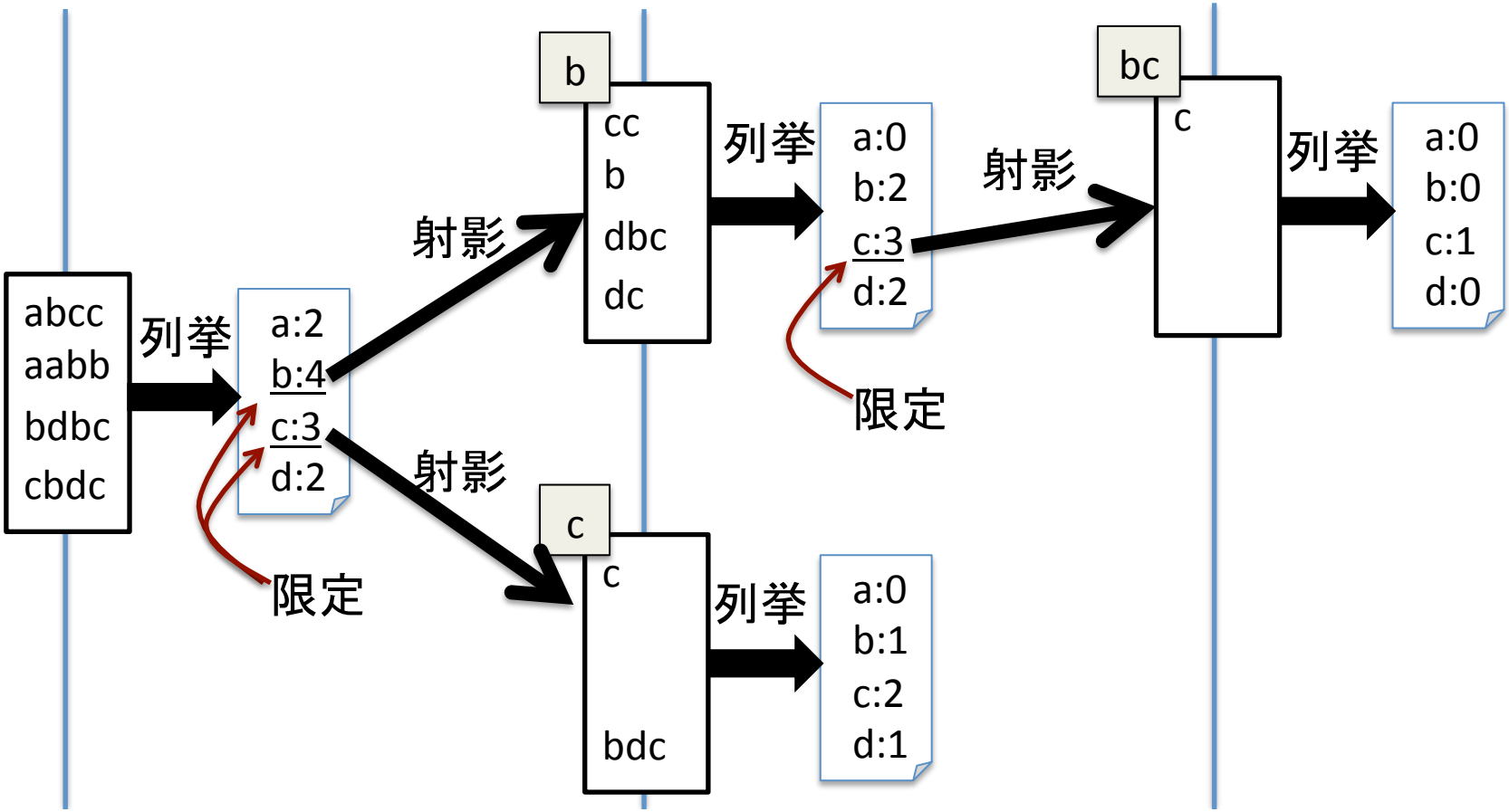


PrefixSpan

- 系列データに存在するすべてのパターンをスキャンし、ある一定数以上含まれるすべてのパターンを検出する
  - 順序のみを考慮し、間に別のデータが挟まっていてもひとつのパターンとして認識する
  - 例： abcde, azdfe には系列パターン’ade’が含まれる

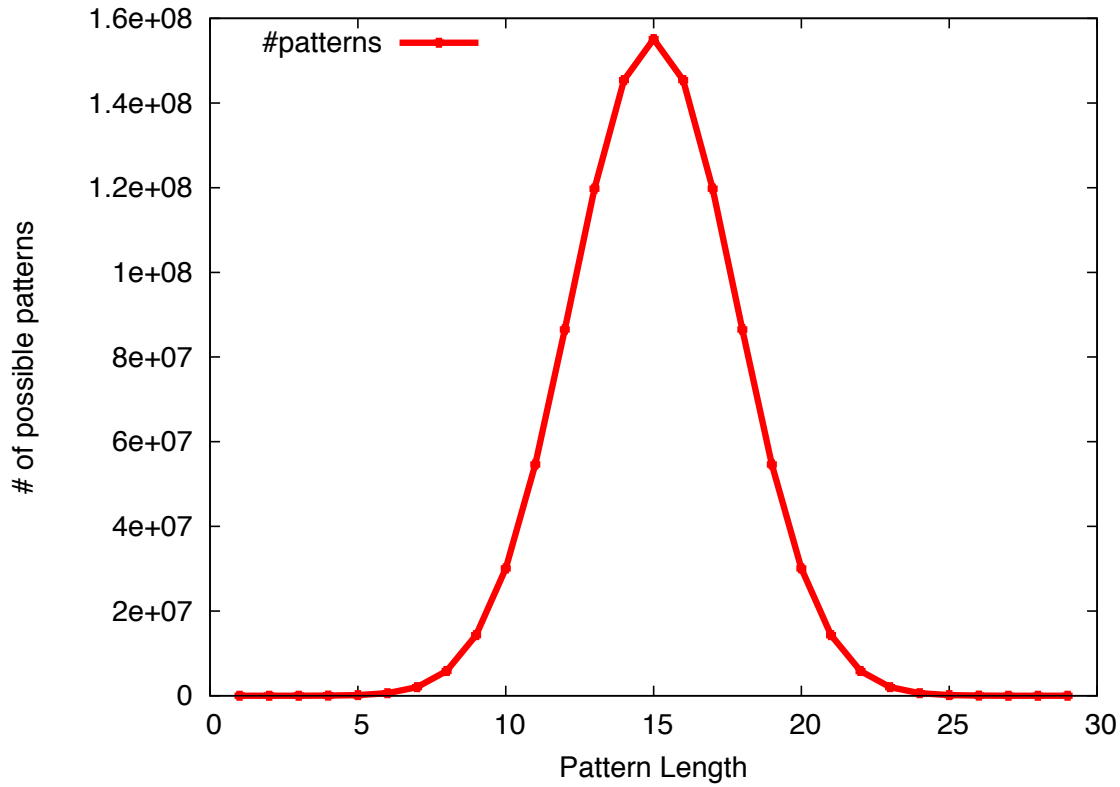
PrefixSpan法の動作

- 列挙：特定の要素に注目し個数を数える
- 限定：一定数以下の出現数のものを排除
- 射影：注目した要素の後続列を生成



PrefixSpanの挙動

- 1つの系列中に登場する、特定の長さのパターンの個数
- ${}_{L}C_{m}$
  - L=系列長
  - m=パターン長
- 系列全長30の場合
  長さ15のパターンが
  最多で10^{8}に達する



MapReduceによるPrefixSpanの実装

- [井上ら, swopp11]の方法に従う
  - s-EB, p-BE, s-BE の三つの手法を提案
  - 最速と思われるs-EBを採用
- s-EB
  - 列挙、射影をMapで実行
  - 限定をReduceで実行
  - メリット：1段の計算を1回のMapReduceで処理
  - デメリット：限定を後に回しているため不要なデータのI/Oが発生する

評価

- PrefixSpan法でのSSS の性能を計測
  - 阪大萩原研の実装をロジックをそのままに移植
- Hadoopと比較
  - 阪大萩原研の実装をそのまま利用

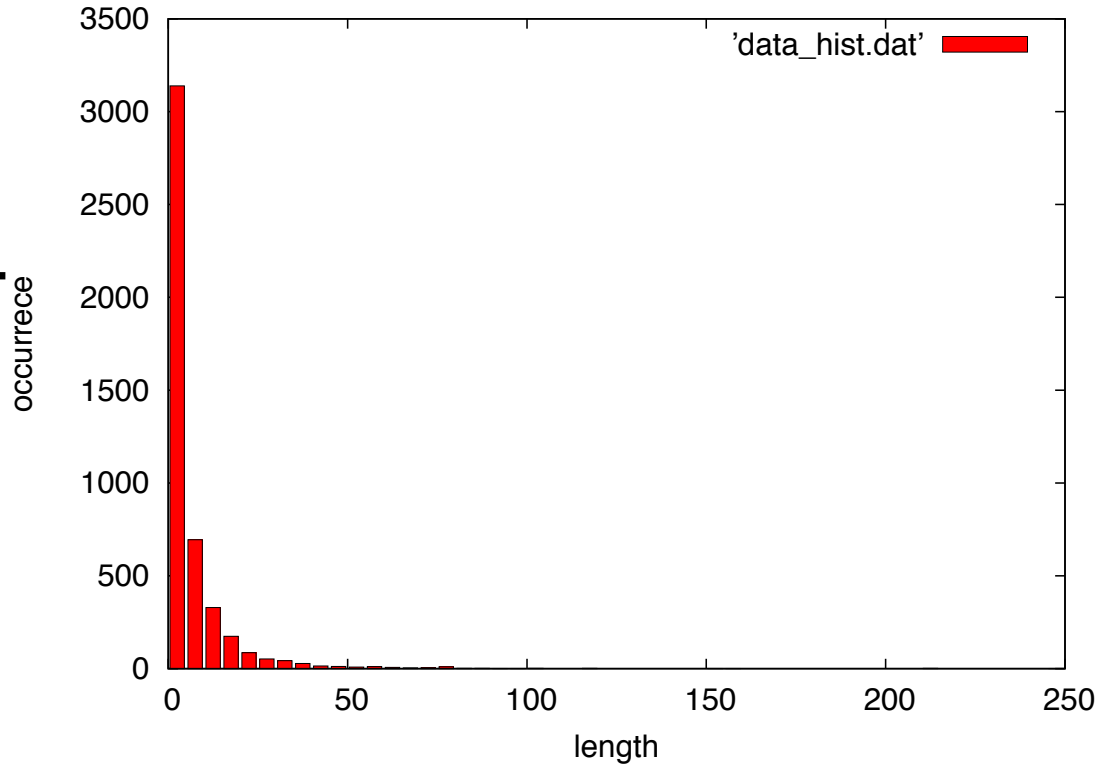
論文と結果が若干異なる。これは論文記述後に発見したプログラム中の余分なファイル転送を削除したため。Hadoop版、SSS版ともに若干高速化している。

評価環境

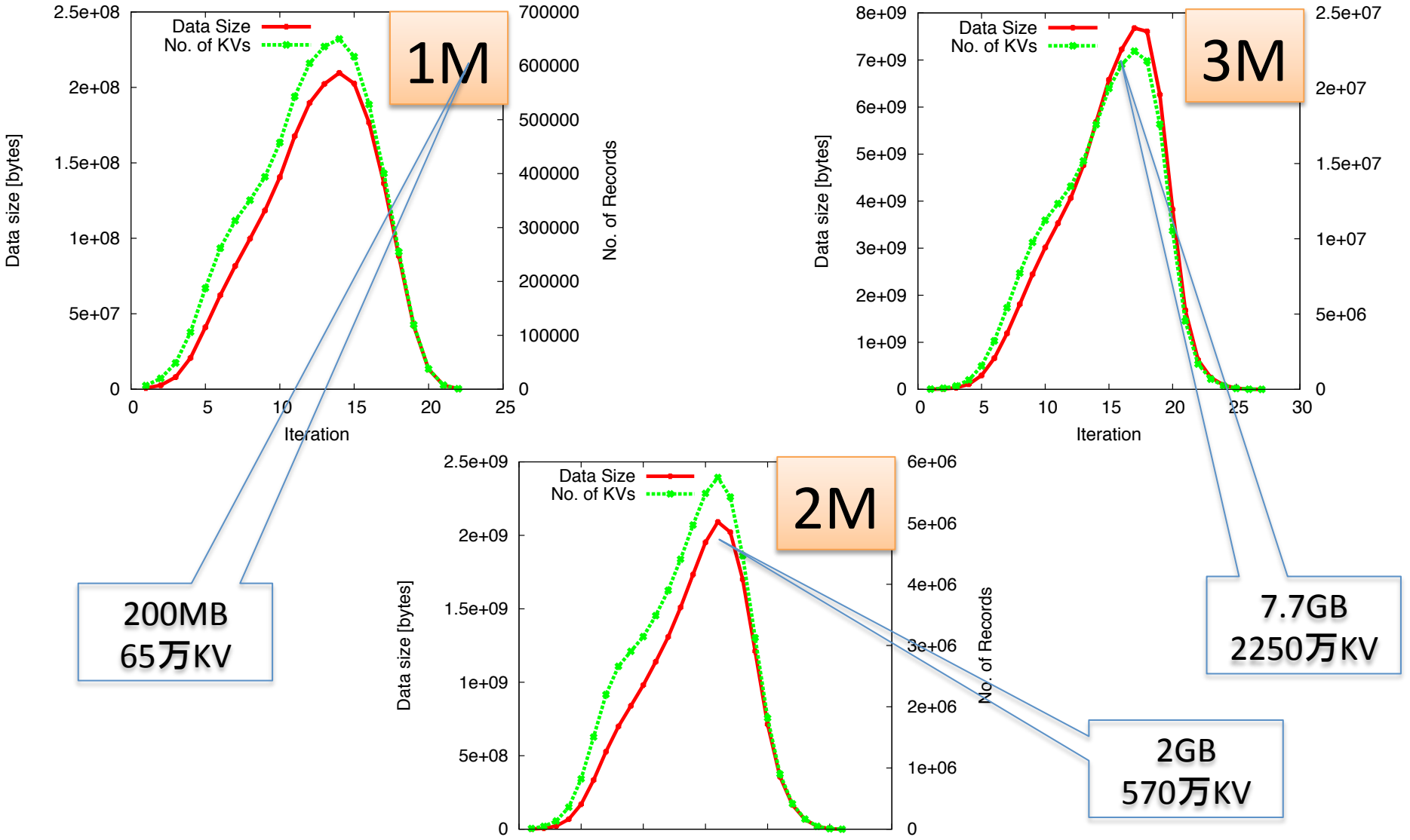
- クラスタを使用
  - Number of nodes: 16 + 1 (master)
  - CPUs per node: Intel Xeon W5590 3.33GHz x 2
  - Memory per node: 48GB
  - OS: CentOS 5.5 x86_64
  - Storage: Fusion-io ioDrive Duo 320GB
  - NIC: Mellanox ConnectX-II 10G
- ソフトウェア
  - SSS
  - Hadoop 0.20.2
    - HDFSレプリカ数を1に設定
- ノード数 4, 8, 16

入力データ

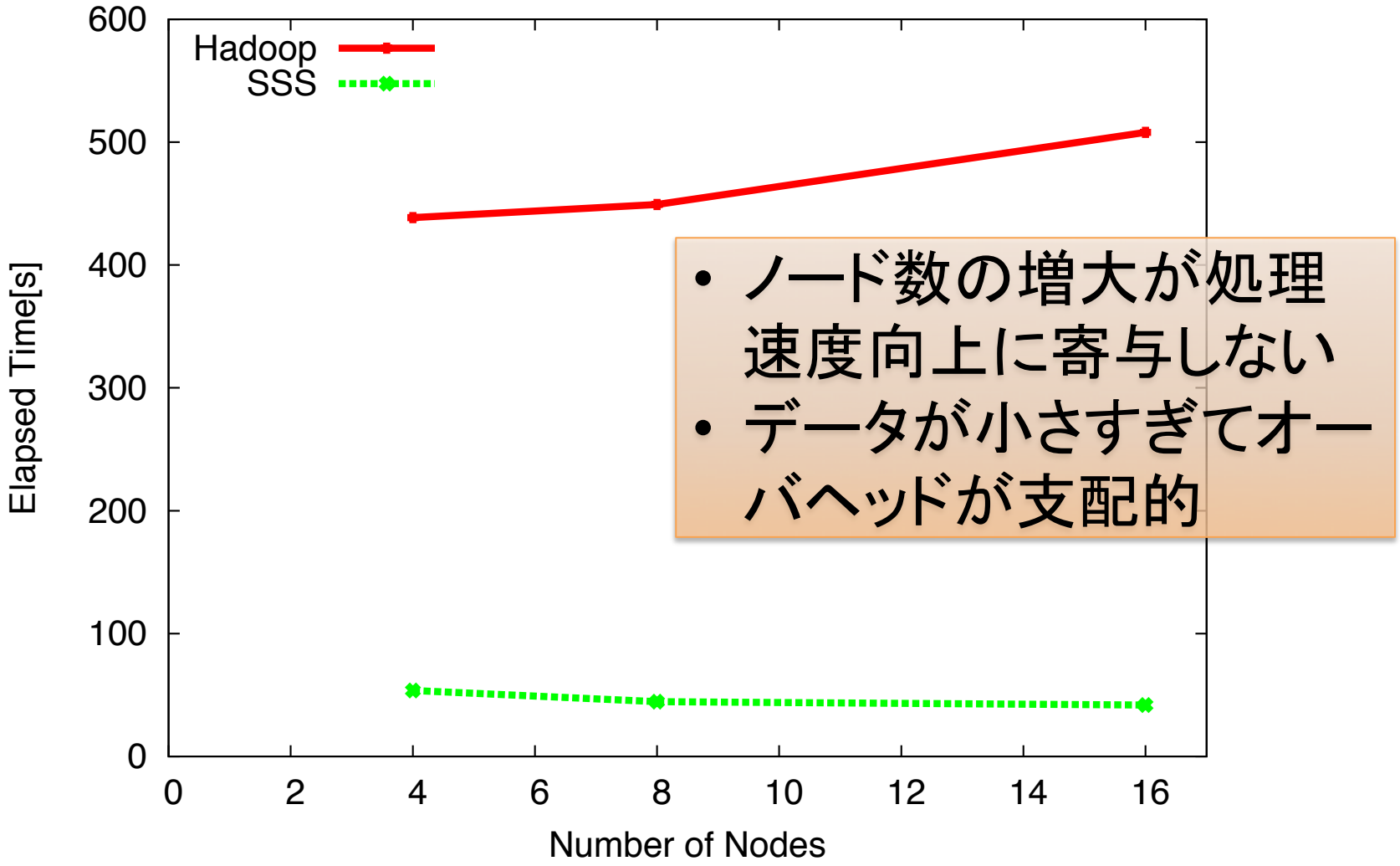
- ソースコード量にして1M,2M,3M
  - 実オープンソースプログラムから取得
- コード化後のデータ量 72K, 136K, 200K
- 大半は長さ30以下
- 1つだけ長さ245が
- 出現回数10個以上を対象



各繰り返しにおけるデータ量

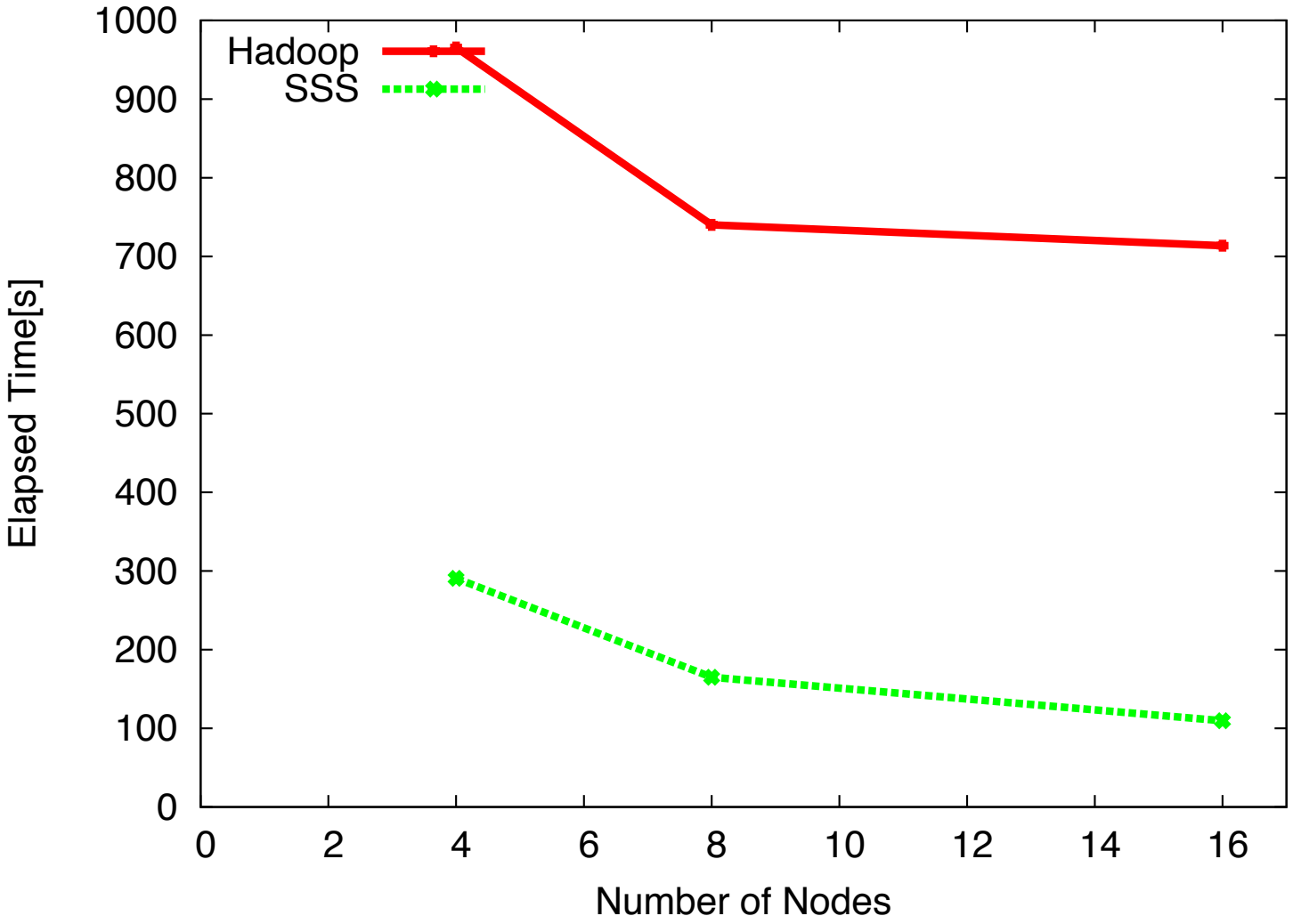


結果：1M

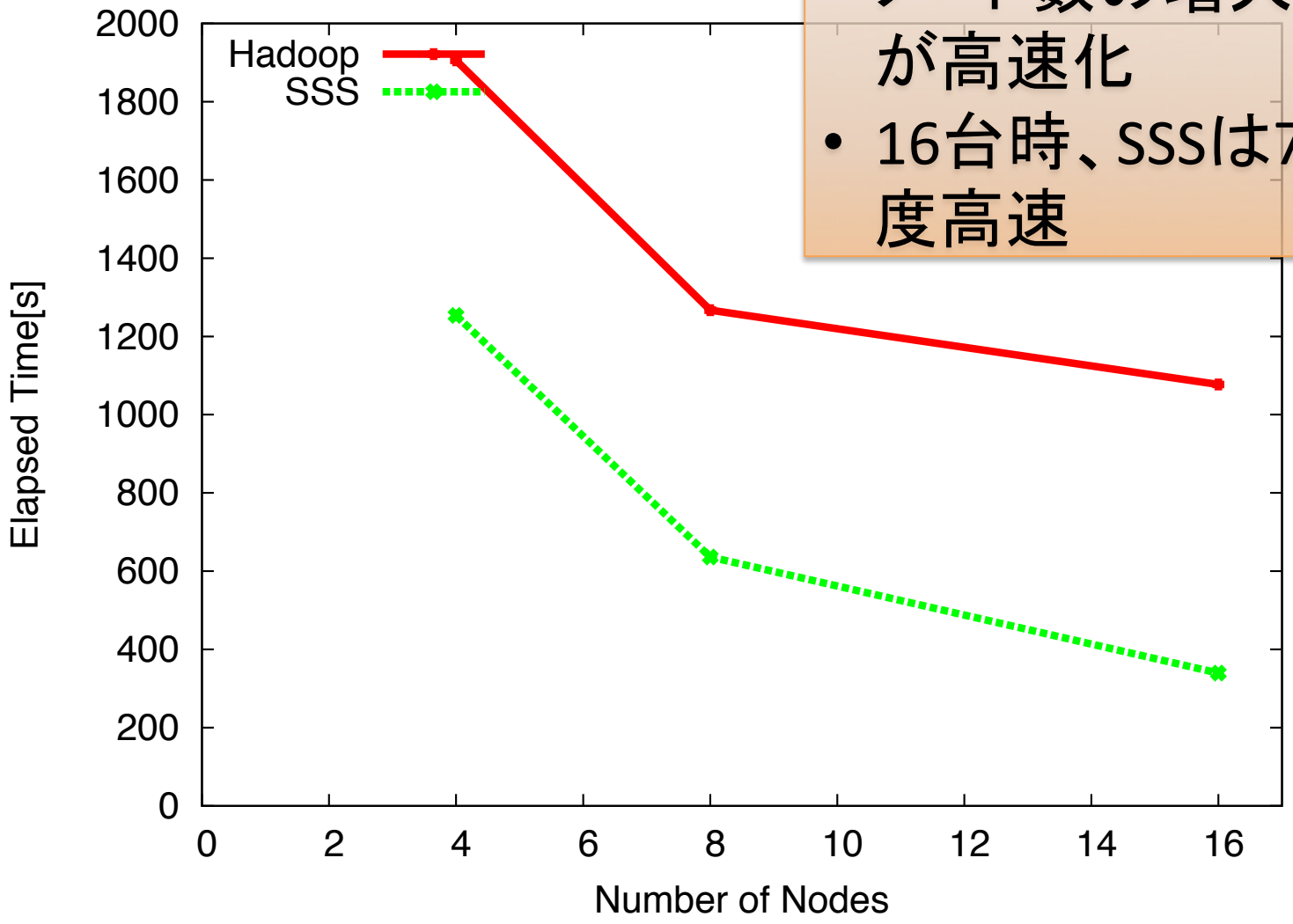


- ノード数の増大が処理速度向上に寄与しない
- データが小さすぎてオーバヘッドが支配的

結果:2M

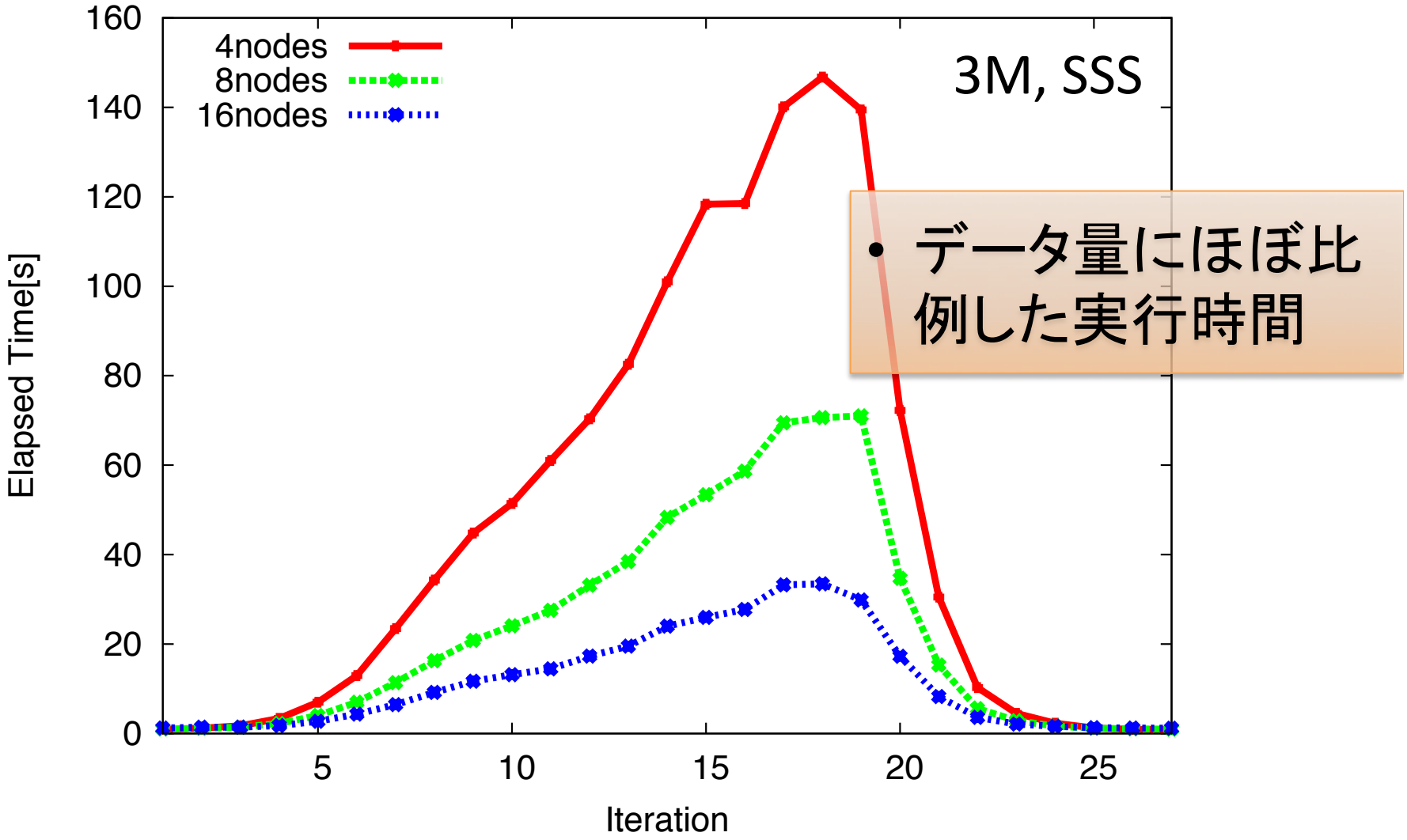

Hadoop
SSS
Elapsed Time[s]
Number of Nodes
0
100
200
300
400
500
600
700
800
900
1000
0
2
4
6
8
10
12
14
16

結果:3M



- ノード数の増大で処理が高速化
- 16台時、SSSは700秒程度高速

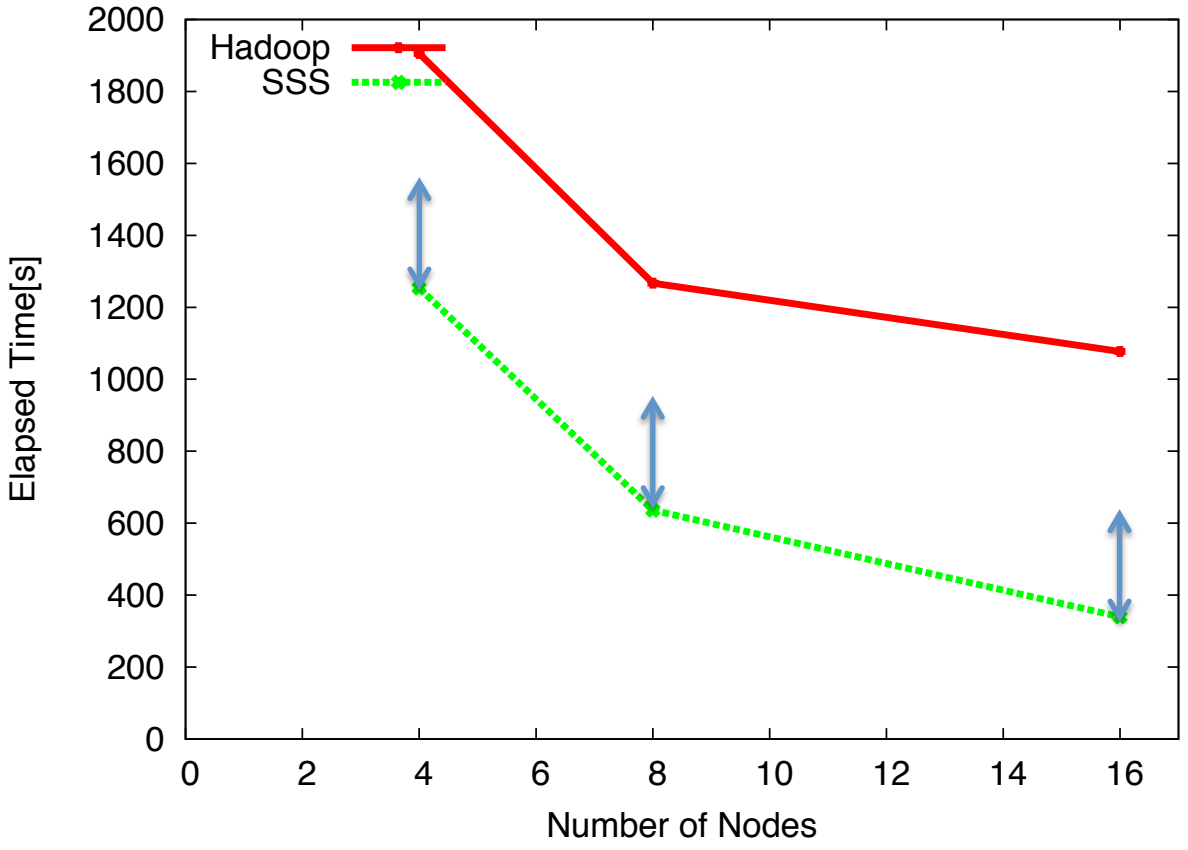
各イタレーションでの実行時間



- データ量にほぼ比例した実行時間

考察

- 繰り返し処理
  - Hadoop : 11.3s, SSS: 0.4s ― 空のジョブの実行時間
  - 27回では294sにも
- シャッフル操作
  - SSSは高速



まとめ

- MapReduce 処理系SSSを、実アプリケーションである、PrefixSpanによるソースコード解析に適用
- Hadoopと比較して高速であることを確認
  - 繰り返し処理の高速性
  - シャッフルの高速性

今後の課題

- s-BE 法での評価
  - 問題サイズによってはs-EB法よりも高速
- プログラム構造の見直しによる高速化
  - データのコピーは不要？インデックスのみでよい
  - データ量のオーダは変わらないが絶対値はかなり小さく抑えることができるはず
- 他の実アプリケーションでの評価

謝辞

- PrefixSpan法のプログラムおよび入力データは、大阪大学井上研究室ならびに萩原研究室の井上佑希氏、置田助教、萩原教授にご提供いただきました。深く感謝の意を表します

- 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