Spark RDDの入出力性能の高速化に関する検討

張 凱輝　（筑波大）
谷村 勇輔　（産総研）
中田 秀基　（産総研）
小川 宏高　（産総研）

研究背景（1）

- Apache Spark(以下，Spark)は機械学習やデータマイニングを高速に実行できるオープンソースの並列データ処理フレームワークとして，ビッグデータ解析や人工知能分野において高い注目を集めている

- 中間データを HDFS(Hadoop Distributed File System)に置くのではなく，ワーカノードのメモリやディスクに保持することにより，反復操作や対話処理において Hadoop より優れた性能を提供

研究背景(2)

- 中間データをディスクに保持することで Spark の実行性能を低下させてしまう可能性がある

- 中間データをディスクに保持した場合の性能低下の程度，その低下を抑制するための方法は明らかになってない

研究の目的

- 中間データにディスクを利用した場合でも，メモリだけを利用した場合にできるだけ近い性能を実現

- Spark アプリケーションの実行性能に与える影響を調査
  - RDDキャッシュの有無
  - RDDキャッシュにディスクを利用
  - 異なる性能のディスクを利用

発表の構成

1. 研究背景
2. 研究目的
3. Sparkの紹介
    1. データフロー
    2. RDD
    3. ストレージレベル
4. 評価実験
    1. 実験項目と方法
    2. 実験環境
    3. 実験結果
5. まとめ
6. 今後の課題

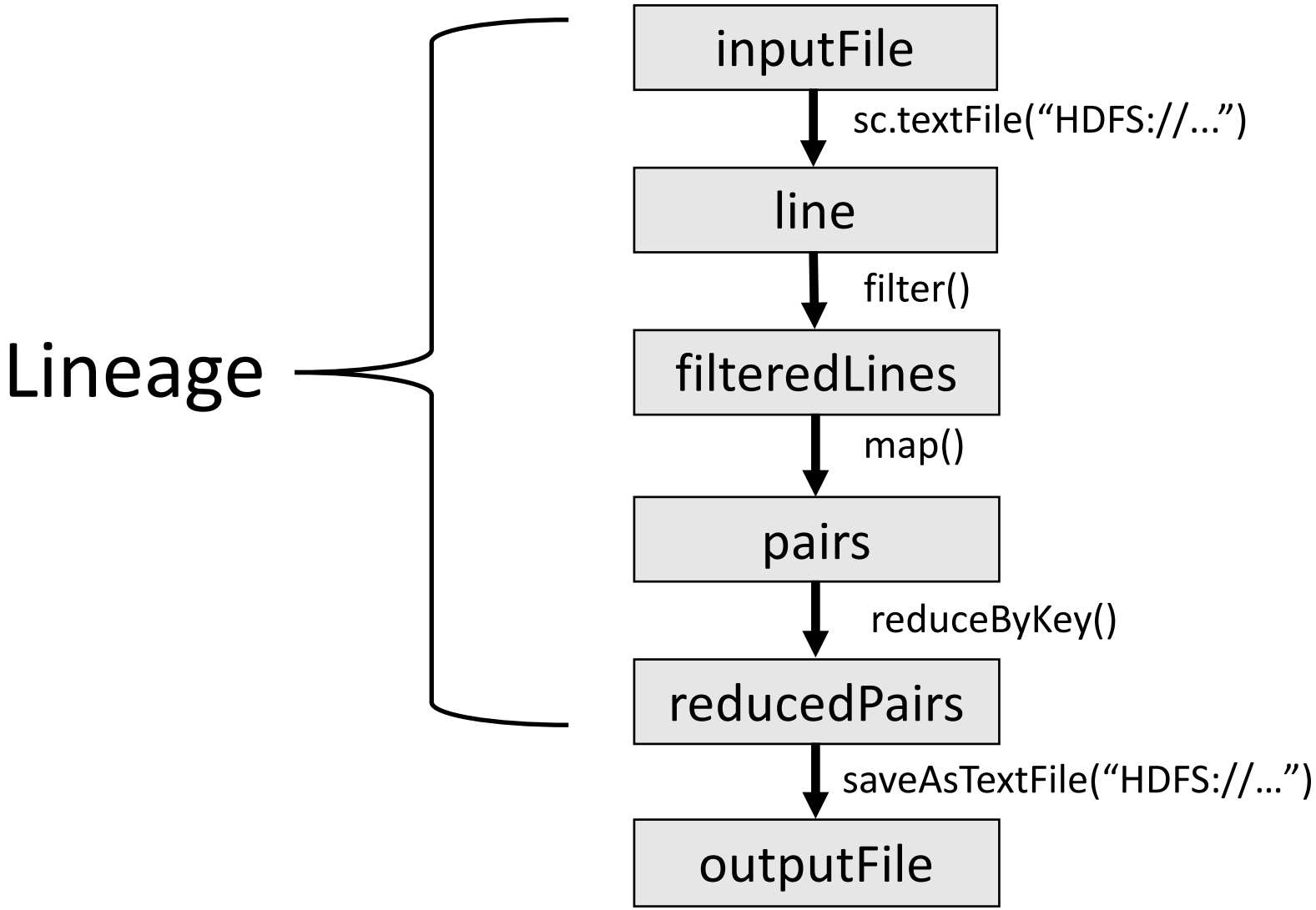
Apache Sparkとは

- オープンソースの並列データ処理フレームワーク

- Hadoop と同様に MapReduce のアルゴリズムに基づいた分散処理が可能
  - コンピューティングとストレージの両機能を持つノードからなるクラスタで実行可能

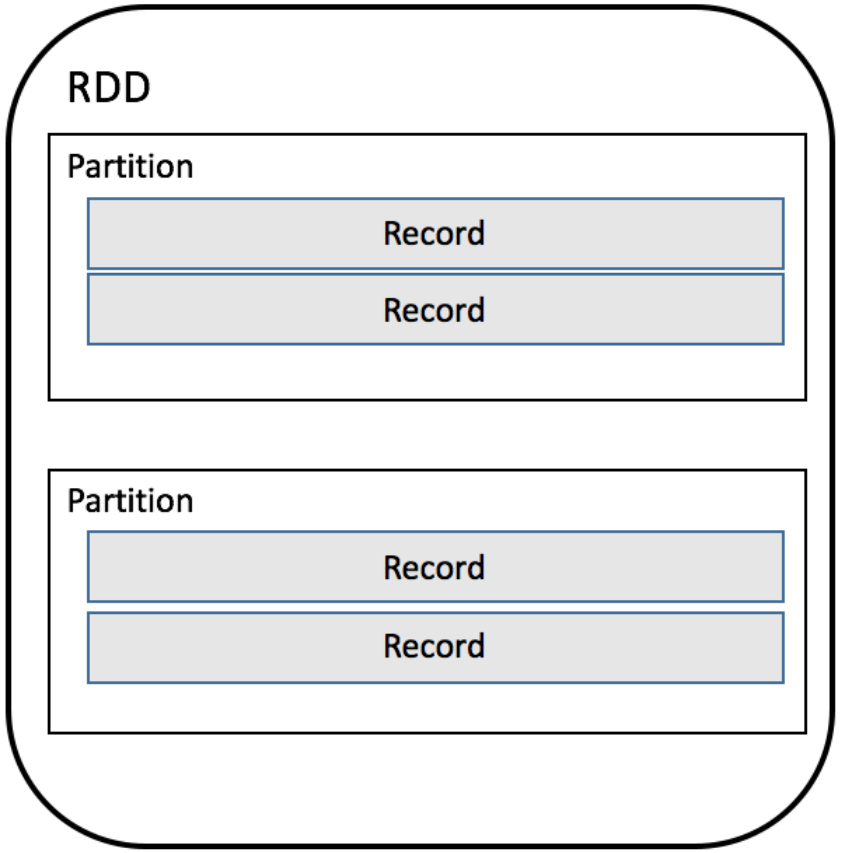
- 中間データをメモリに保持することで機械学習やデータマイニングなどの反復計算を効率良く実行
  - 反復計算の負荷が大きいほど，Spark を利用メリットはより大きくなる

Sparkのデータフロー



RDD : Resilient Distributed Datasets

- 読み取り専用の分散データ構造
- 内部はパーティションに分割
- 各パーティションは複数のワーカノードに分散配置
  - データ処理の単位として分散並列実行が可能



RDD キャッシュ（Persist）

- RDD を生成したノードは，自身が生成したパーティションを明示的にメモリあるいはディスクに保存することが可能（Persist）

- RDD に対する反復操作や障害からの復旧を高速化する効果

- RDD に対して persist() メソッドを呼び出して生成
  - 引数にストレージレベルを指定

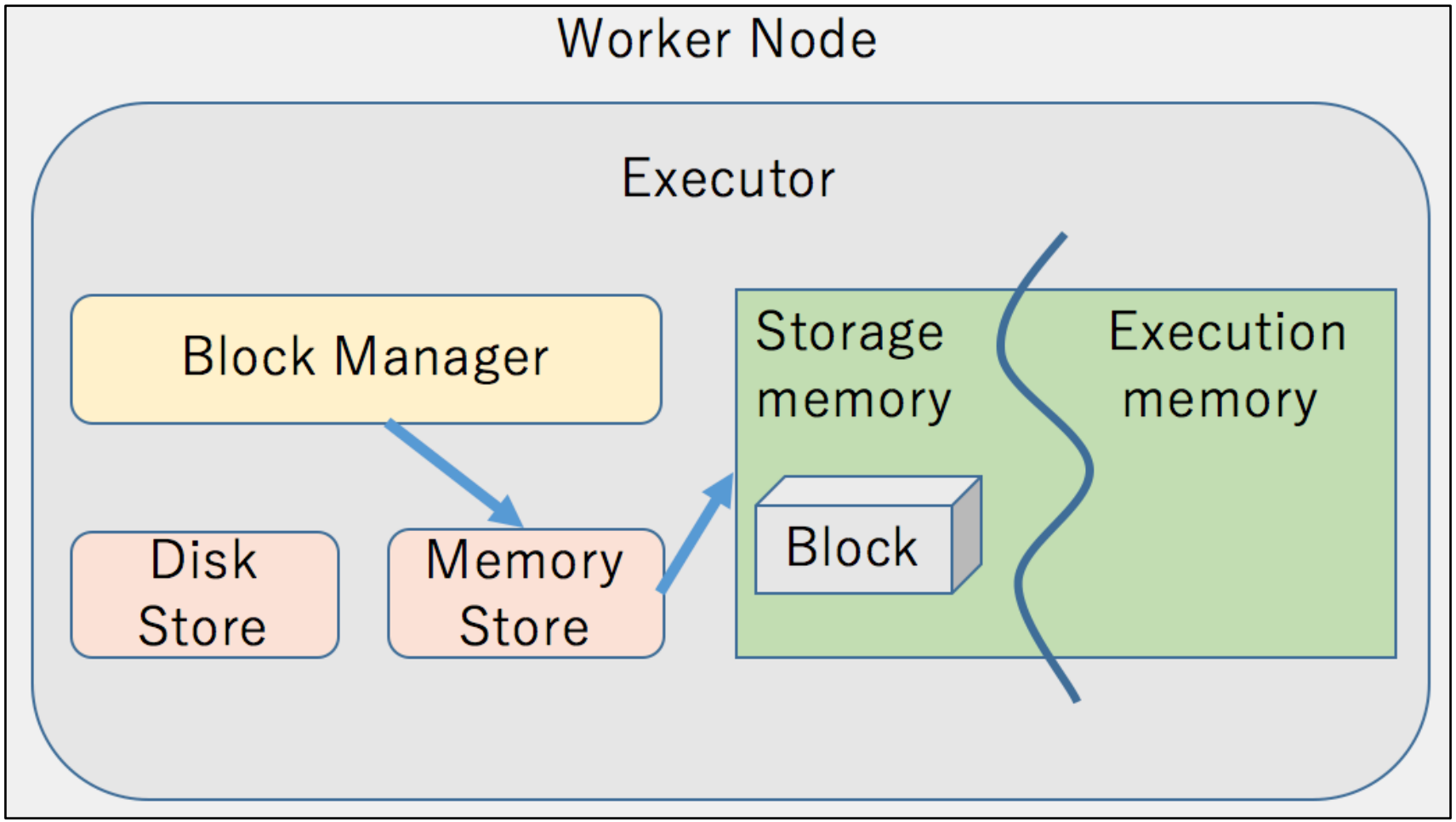
RDDキャッシュのストレージレベル

- メモリあるいはディスクを指定
- 併用する指定も可能
- メモリを指定する場合にはシリアライズの有無も指定可能
- 引数:
  - MEMORY_ONLY
  - MEMORY_ONLY_SER
  - MEMORY_AND_DISK
  - DISK_ONLY

  など

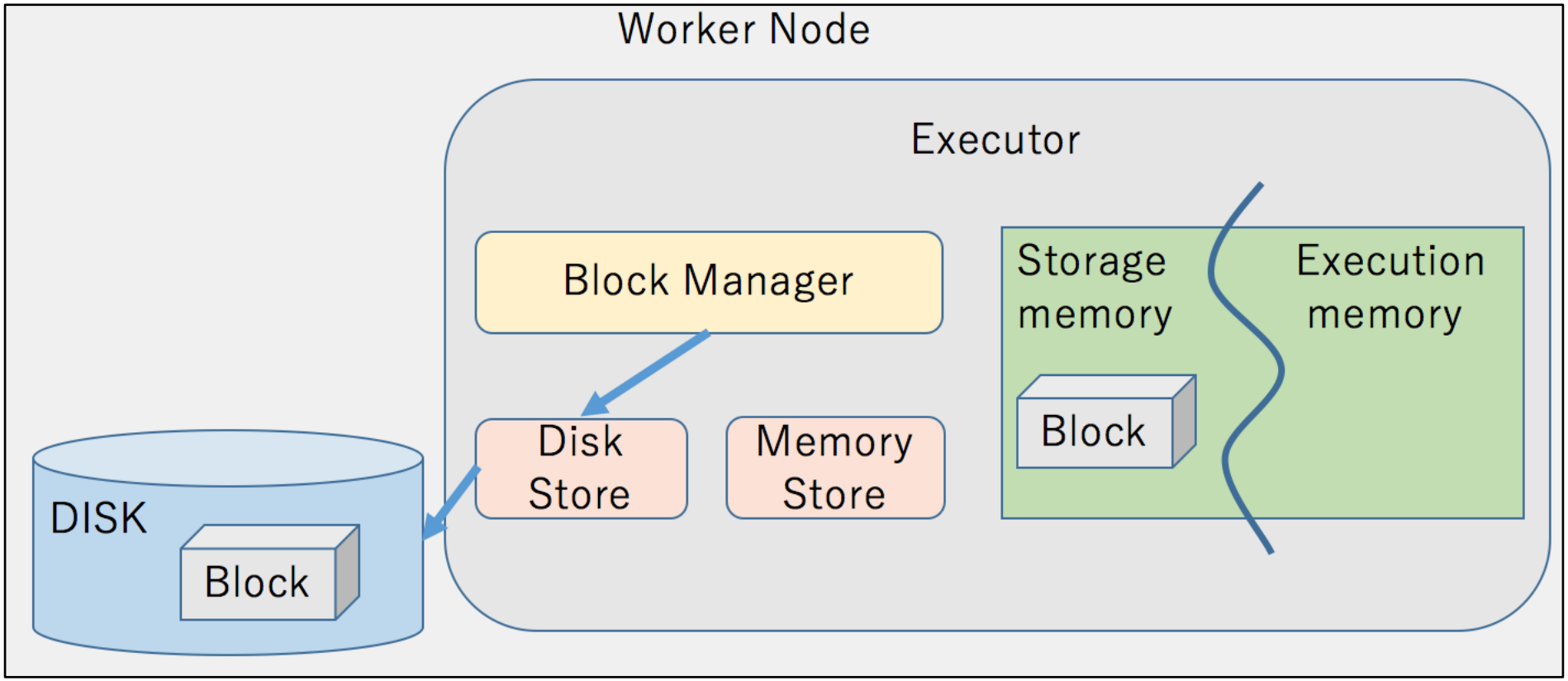
MEMORY_ONLY

- RDDキャッシュをメモリ上にのみ生成
- 性能は最も高速



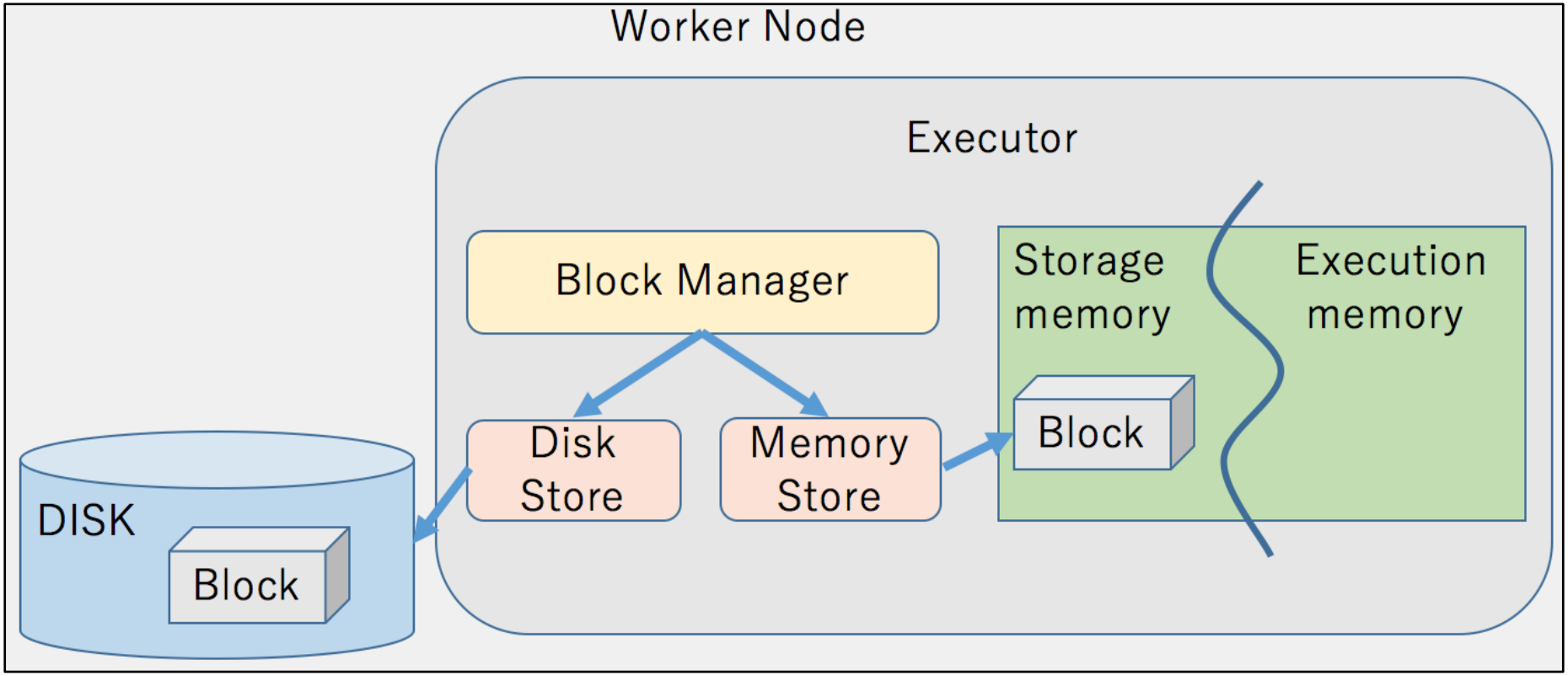
DISK_ONLY

- RDDキャッシュをディスク上にのみ生成
- 性能は最も低速であるが，メモリを消費しない



MEMORY AND DISK

- メモリとディスクを併用して RDDキャッシュを生成
- メモリにロードされたパーティションに対しては Least-Recently-Used(LRU) 規則が適用され,アクセス頻度が最低のパーティションはドロップされる
- メモリを優先的に利用し,収まりきらない場合，新しい Block のために古い Block をシリアライズしてディスクに移動
- ディスクに保存された RDD を参照する際は,メモリに戻す操作を行う



評価実験

- 調査項目
  1. RDD キャッシュの有無とストレージレベルの違いによる性能の違い
  2. ディスク利用による性能
  3. メモリとディスクの併用による性能
  4. ストレージデバイスによる性能の違い

- 調査方法
  - Spark の機械学習ライブラリ(MLlib)に含まれるベンチマークや独自の測定プログラムを実行
  - RDDキャッシュの有無、ストレージレベル、RDDのサイズ、ドライバーのスレッド数、ストレージデバイスなどを変更して性能を測定

実験環境

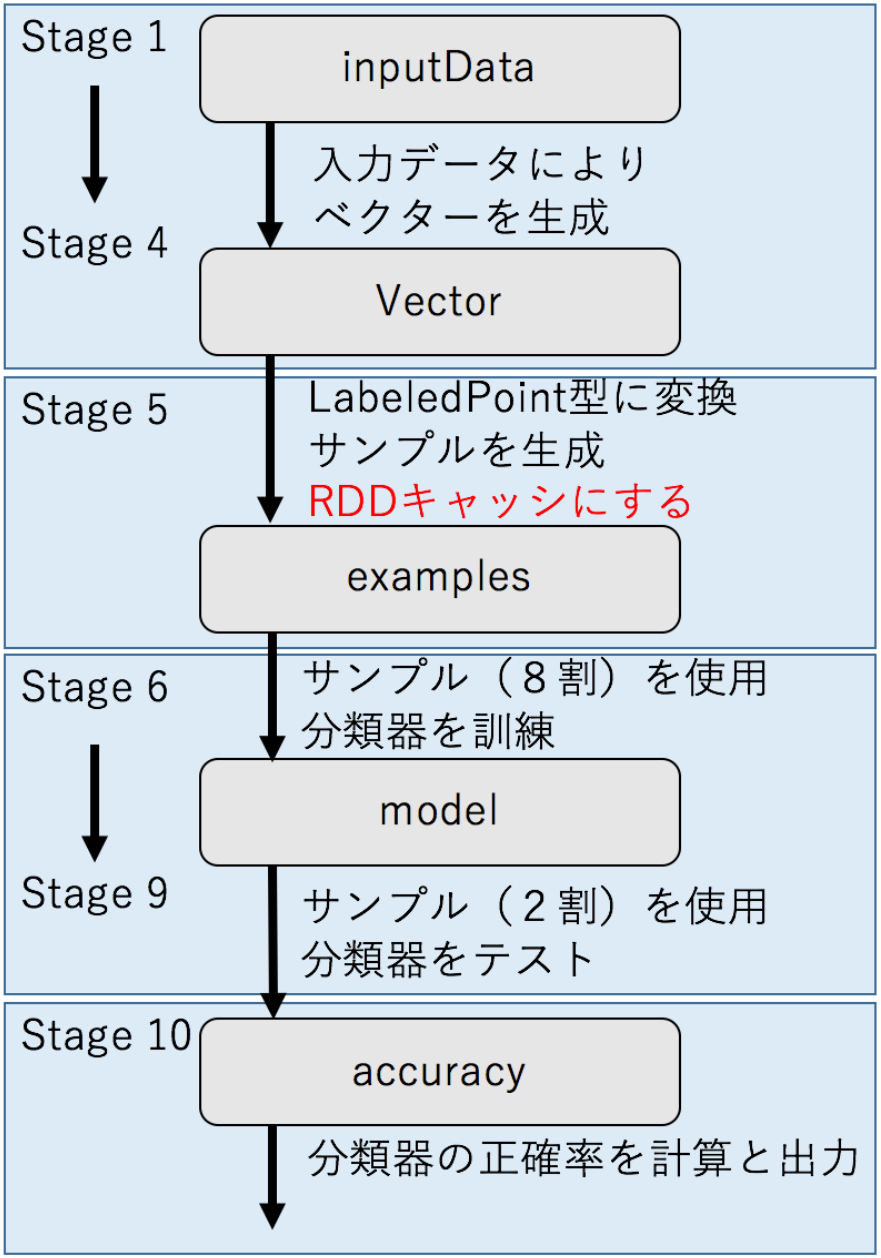
- ローカルモードで Spark の各ベンチマークプログラム(SparseNaiveBayes, DenseKMeans, RDDTest)を実行
- RDDキャッシュにディスクを利用する場合には,**RamDisk**, **NVMe-SSD**, **SATA-SSD** および **HDD** の 4 種類を使用
- **Spark v.1.6.1** を用い,**Scala v.2.10.6** および **Java v.1.8.0 66**

CPU	Intel Xeon CPU E5-2620v3 2.40GHz, 6 cores x2
Memory	128 GB
NetWork	10 Gbps (for HDFS connection)
NVMe-SSD	Intel SSD DC P3700
SSD	OCZ Vertex3 (240GB, SATA6G I/F)
HDD	Hitachi Travelstar 7K320 (SATA3G I/F)
OS	Ubuntu 14.04 (Kernel v.3.13)

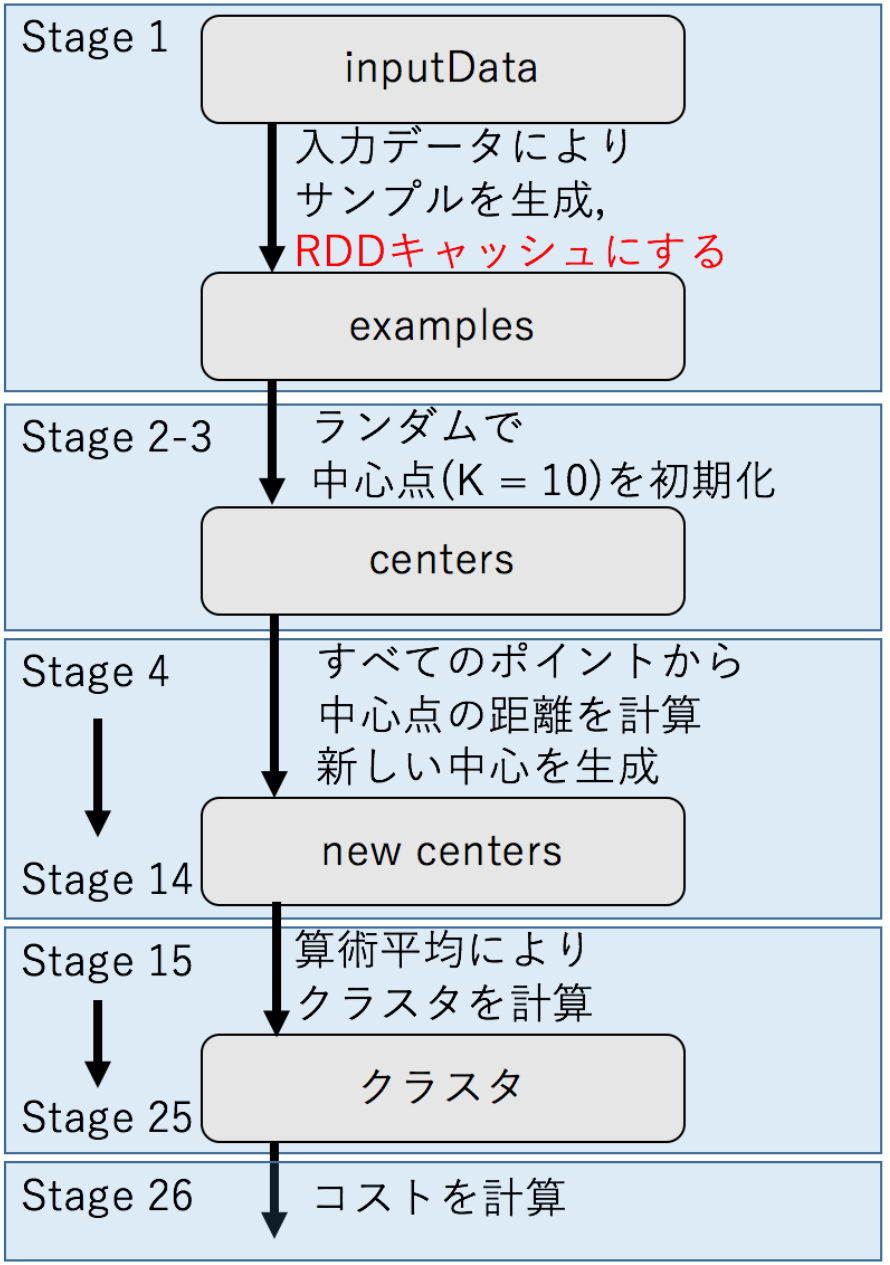
機械学習プログラムによるベンチマーク

- SparseNaiveBayes, DenseKMeans を利用
  - NaiveBayes: 単純ベイズ分類器
  - Kmeans: k平均法
- 各入力データは HiBench を用いて生成,データサイズには Tiny を指定
  - 入力データサイズ：370MB(NaiveBayes)、4GB(Kmeans)
  - 中間データサイズ：
    - NaiveBayes:780MB(DISK_ONLY)、788MB(MEMORY_ONLY)
    - Kmeans:7798MB(DISK_ONLY)、8546MB(MEMORY_ONLY)
- 異なるストレージレベルにおいて（各ステージの）実行時間を比較

SparseNaiveBayes



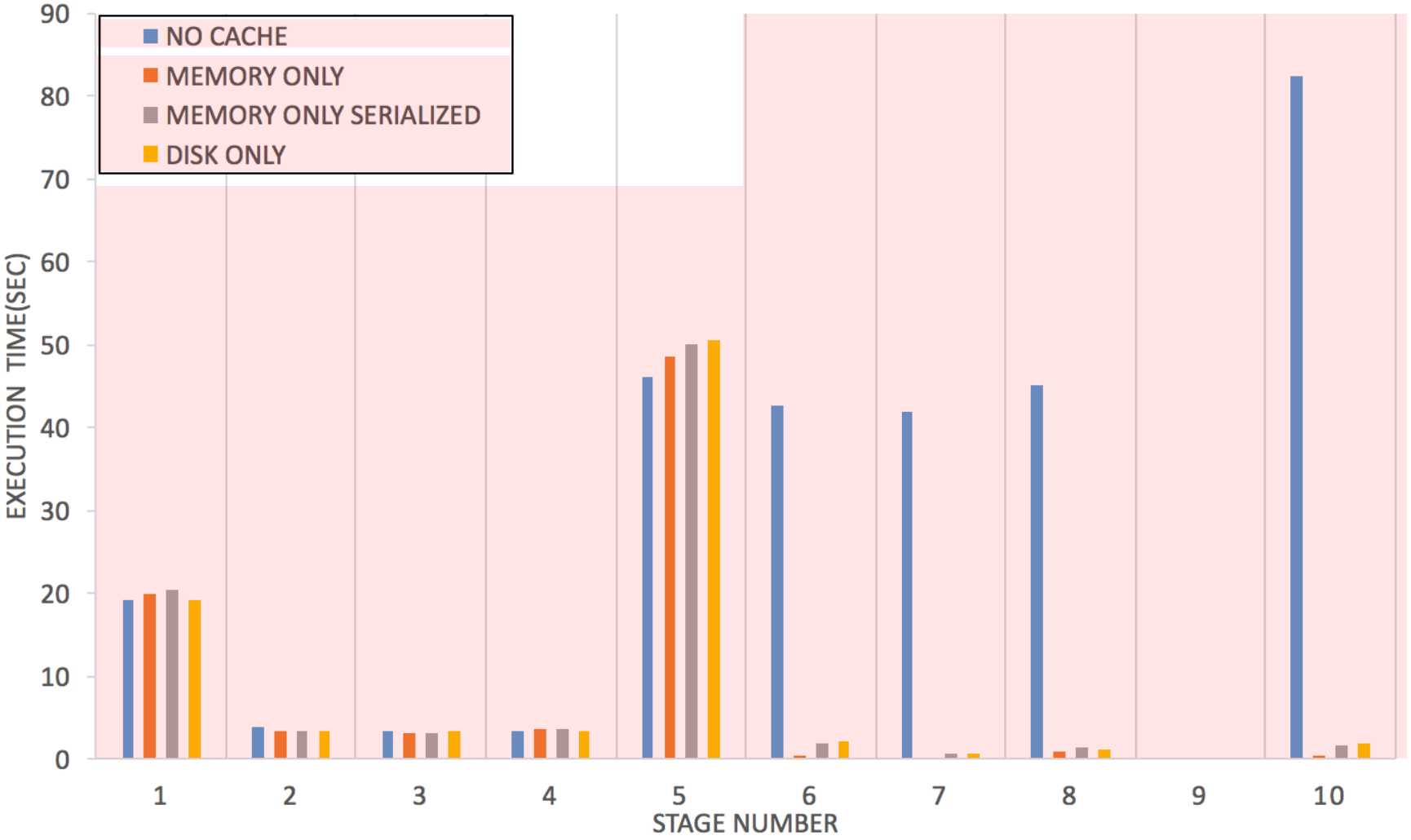
DenseKmeans



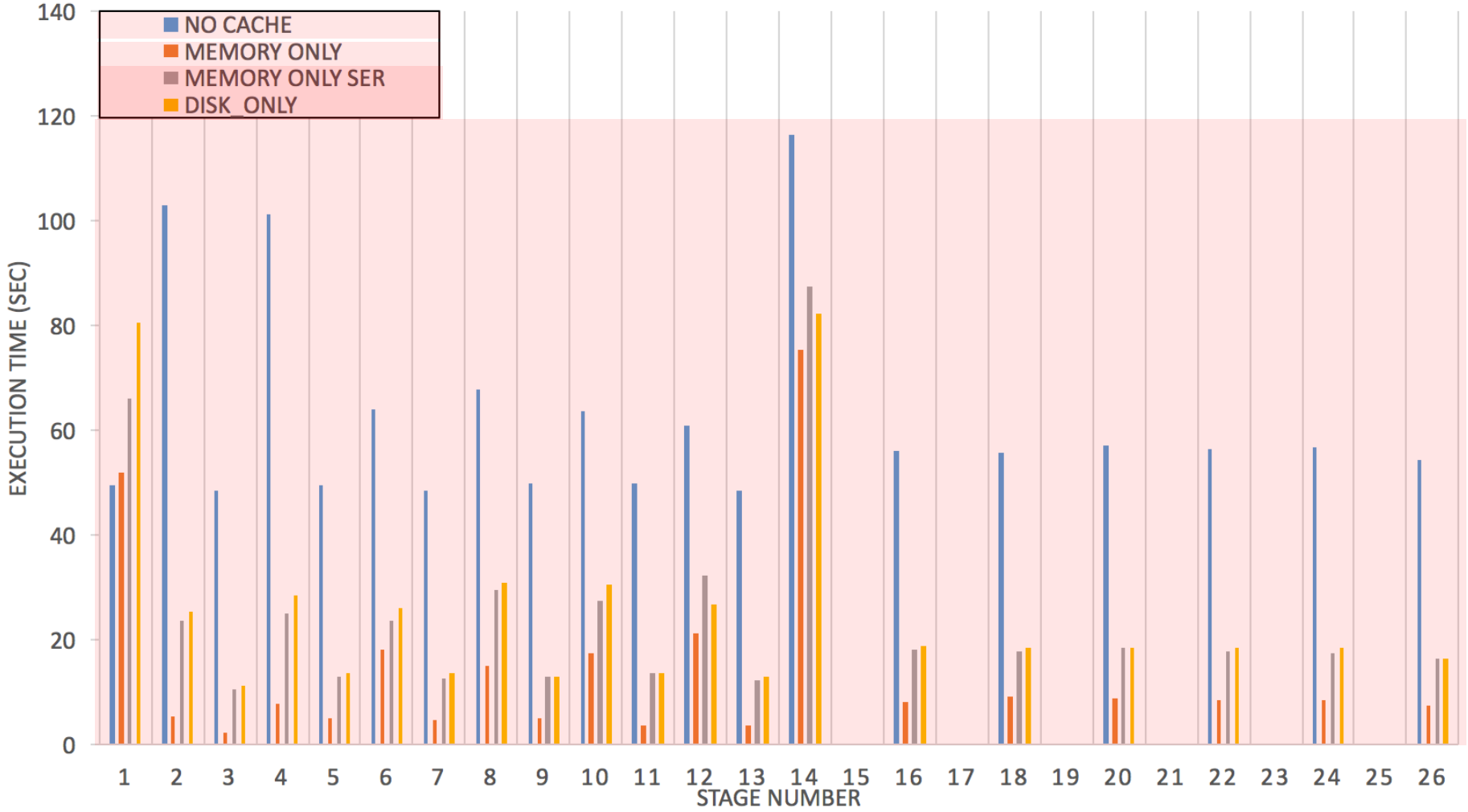
RDD キャッシュの効果とディスク利用による影響（実験１）

- 設定
  - スレッド数：**1**
  - Driverのメモリ：**60GB**
  - ストレージレベル：**NOCACHE**
    MEMORY ONLY
    MEMORY ONLY SER
    DISK ONLY
  - プログラム：**SparseNaiveBayes, DenseKMeans**

RDD キャッシュの効果とディスク利用の影響 (SparseNaiveBayes)



RDD キャッシュの効果とディスク利用の影響 (DenseKMeans)



メモリとディスクの併用による影響（実験２）

- 設定
  - スレッド数：**1**
  - ストレージレベル：**MEMORY AND DISK** の場合
    Driverのメモリ量：**512MB**, **1GB**, **2GB**

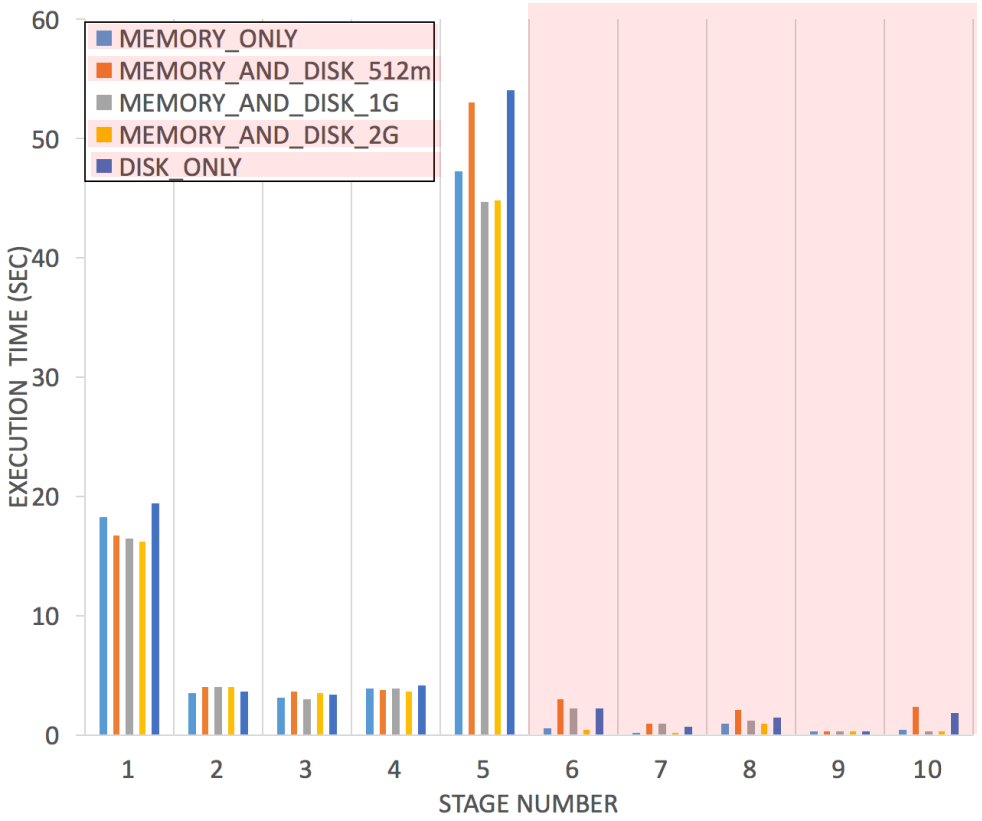
  - ストレージレベル：**MEMORY ONLY, DISK ONLY** の場合
    Driverのメモリ量：**60GB**

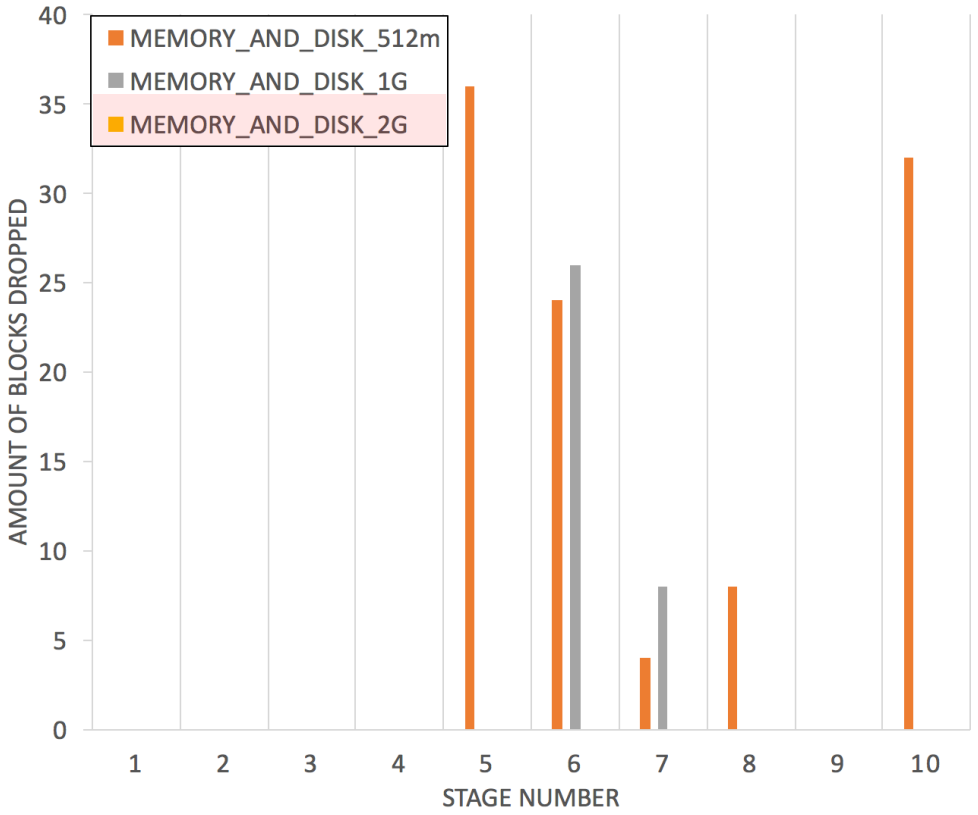
  - プログラム：**SparseNaiveBayes, DenseKMeans**

SparseNaiveBayesの結果

左：メモリとディスクの併用による影響

右：MemoryStore からのドロップの発生結果

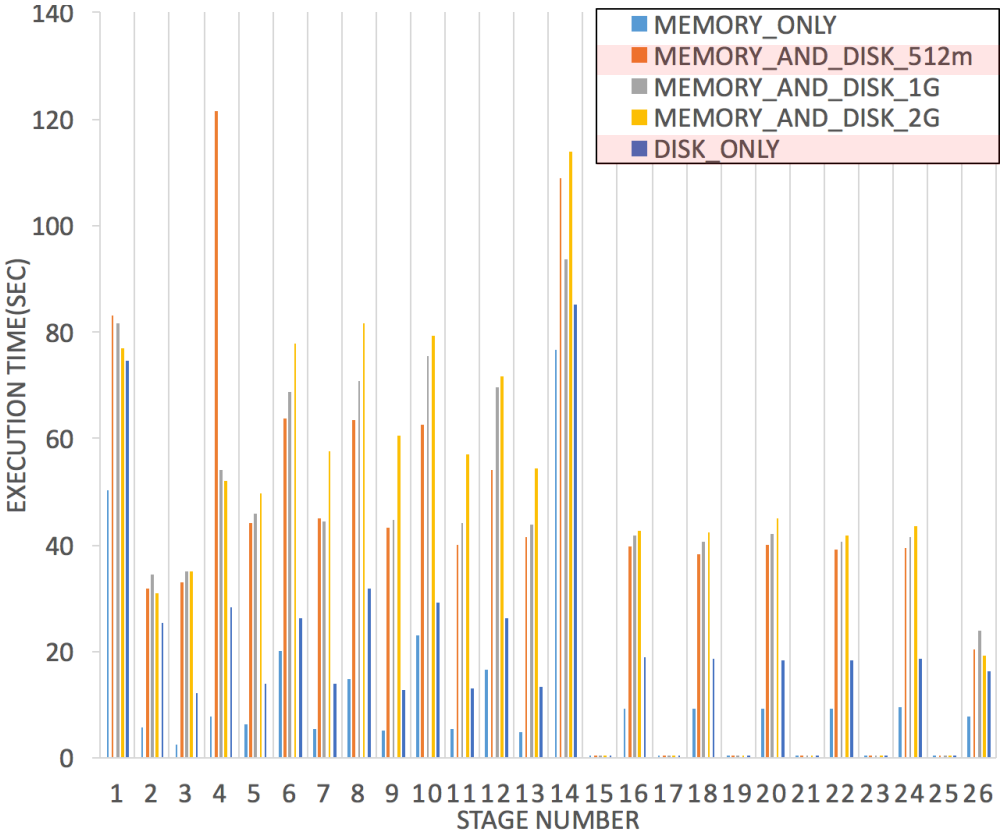


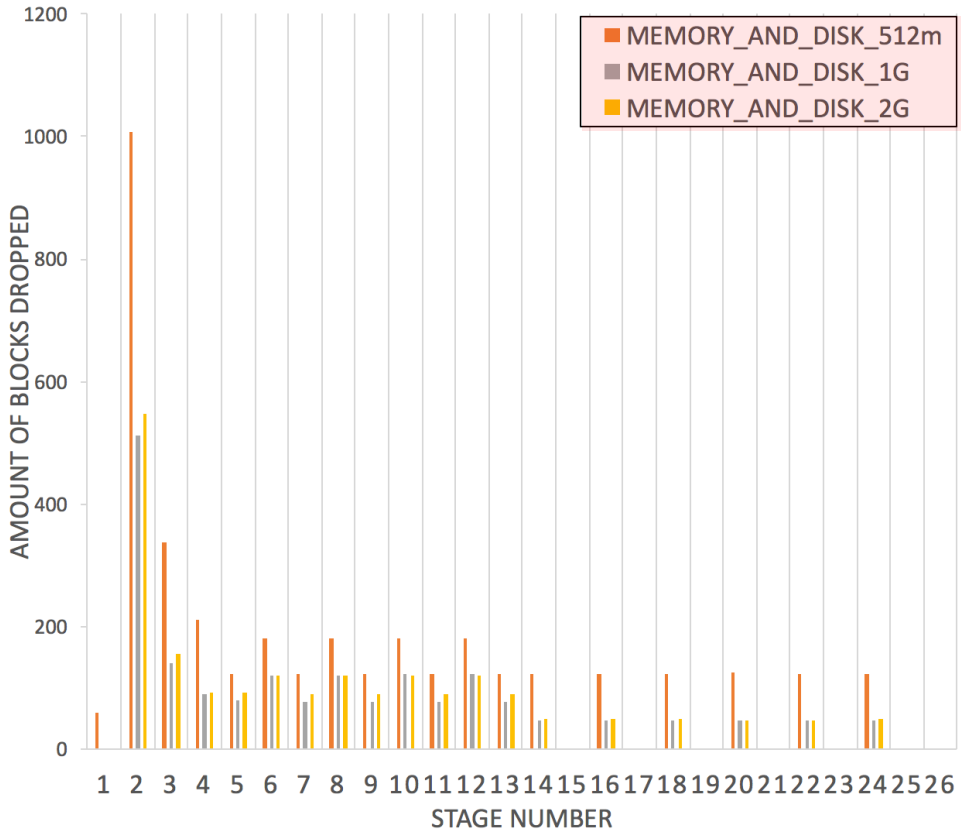


DenseKmeansの結果

	Full GCの回数	Full GCに要した時間
memory and disk 512m	686	220s
disk only	2	0.21s

左：メモリとディスクの併用による影響
右：MemoryStore からのドロップの発生結果

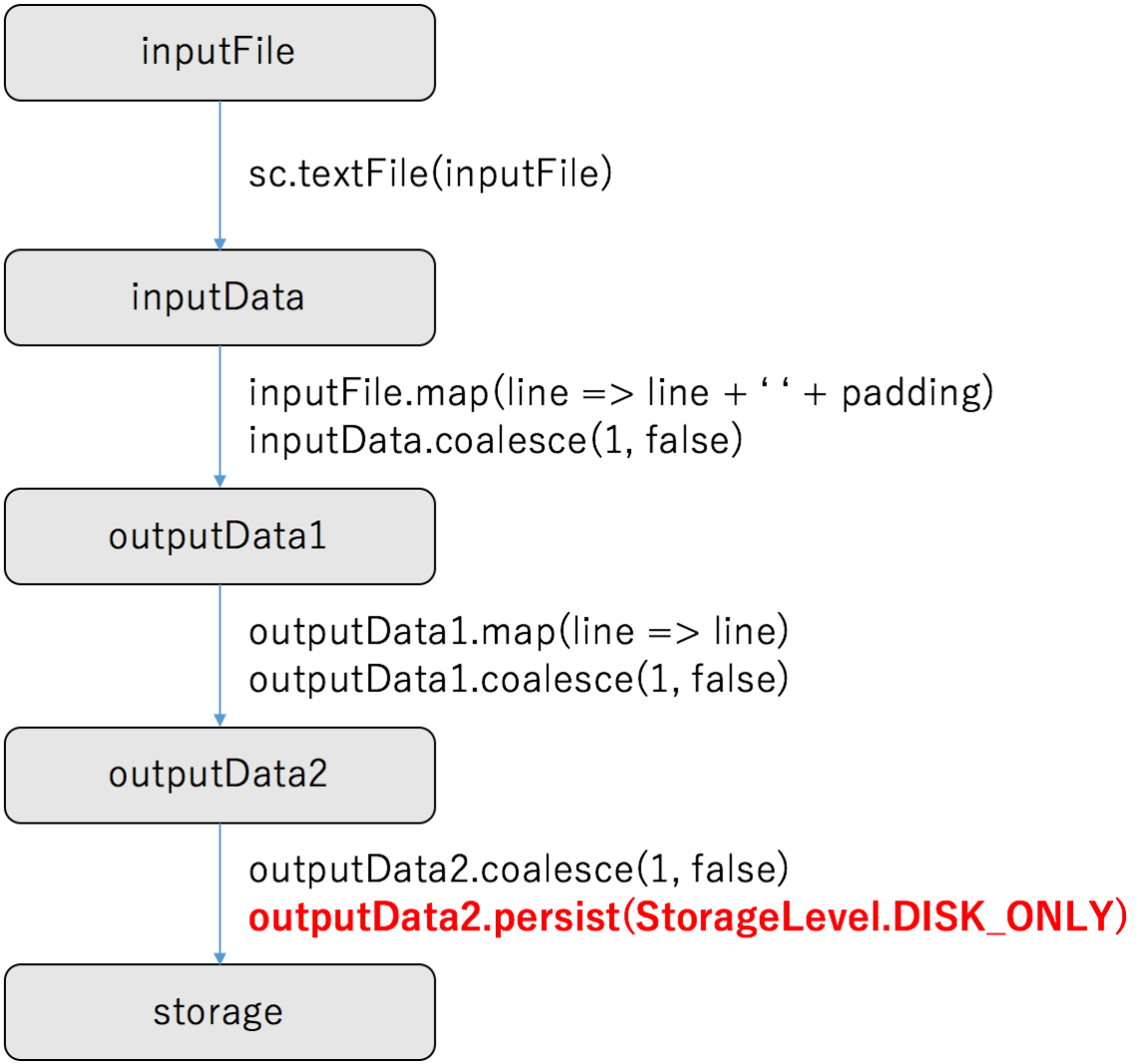




ストレーシジデバイスによる違い（実験３）

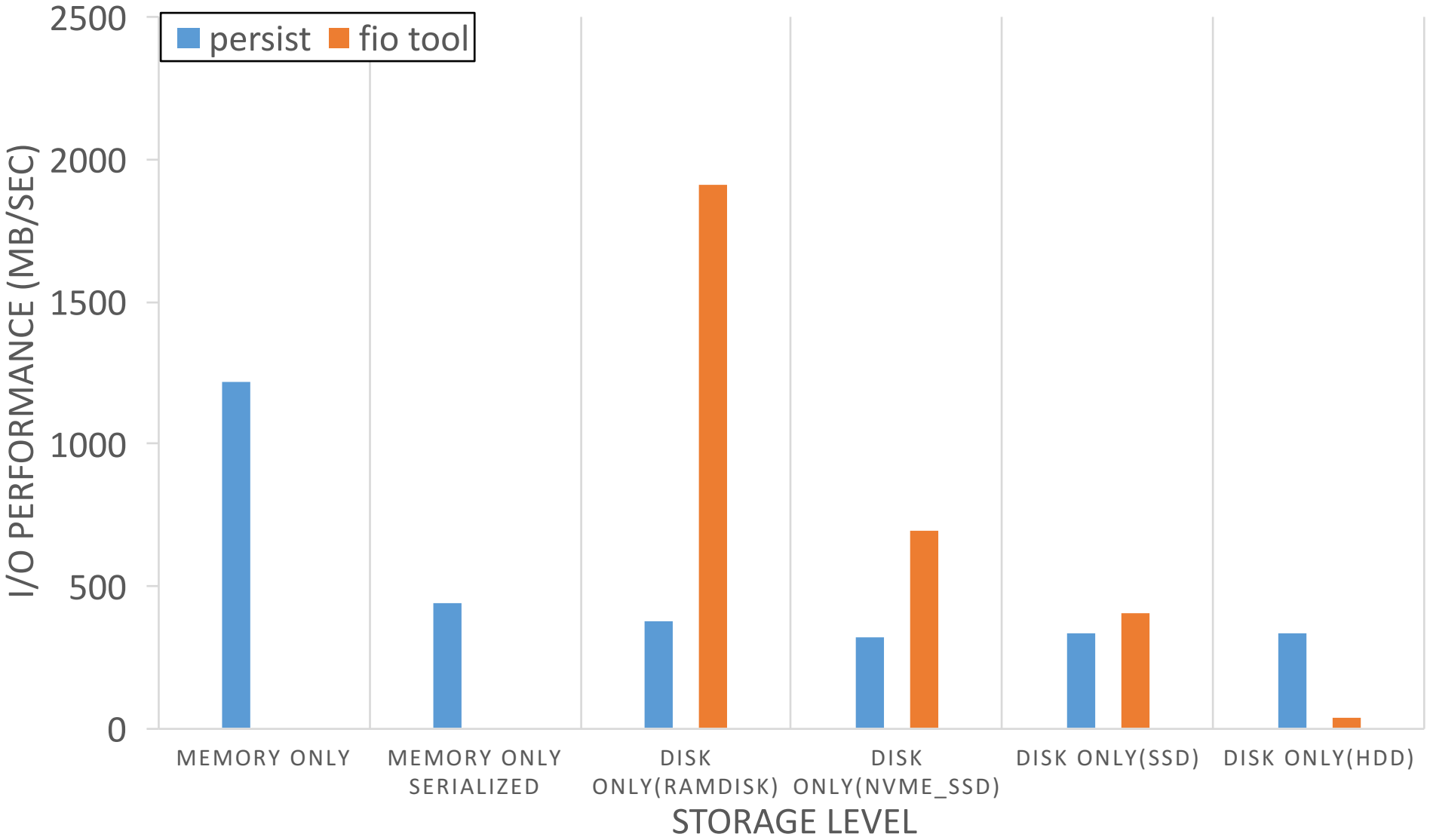
- 設定１
  - スレッド数は **1**
  - Driverのメモリ量 : **64GB**
  - ストレー ジレベル : **DISK ONLY**
  - RDDのサイズ : **1000MB**
  - プログラム : **RDDTest, Fioベンチマーク**

RDDTest



- Spark のワーカ処理において任意のサイズの RDD を生成し，Persist を実行するプログラム

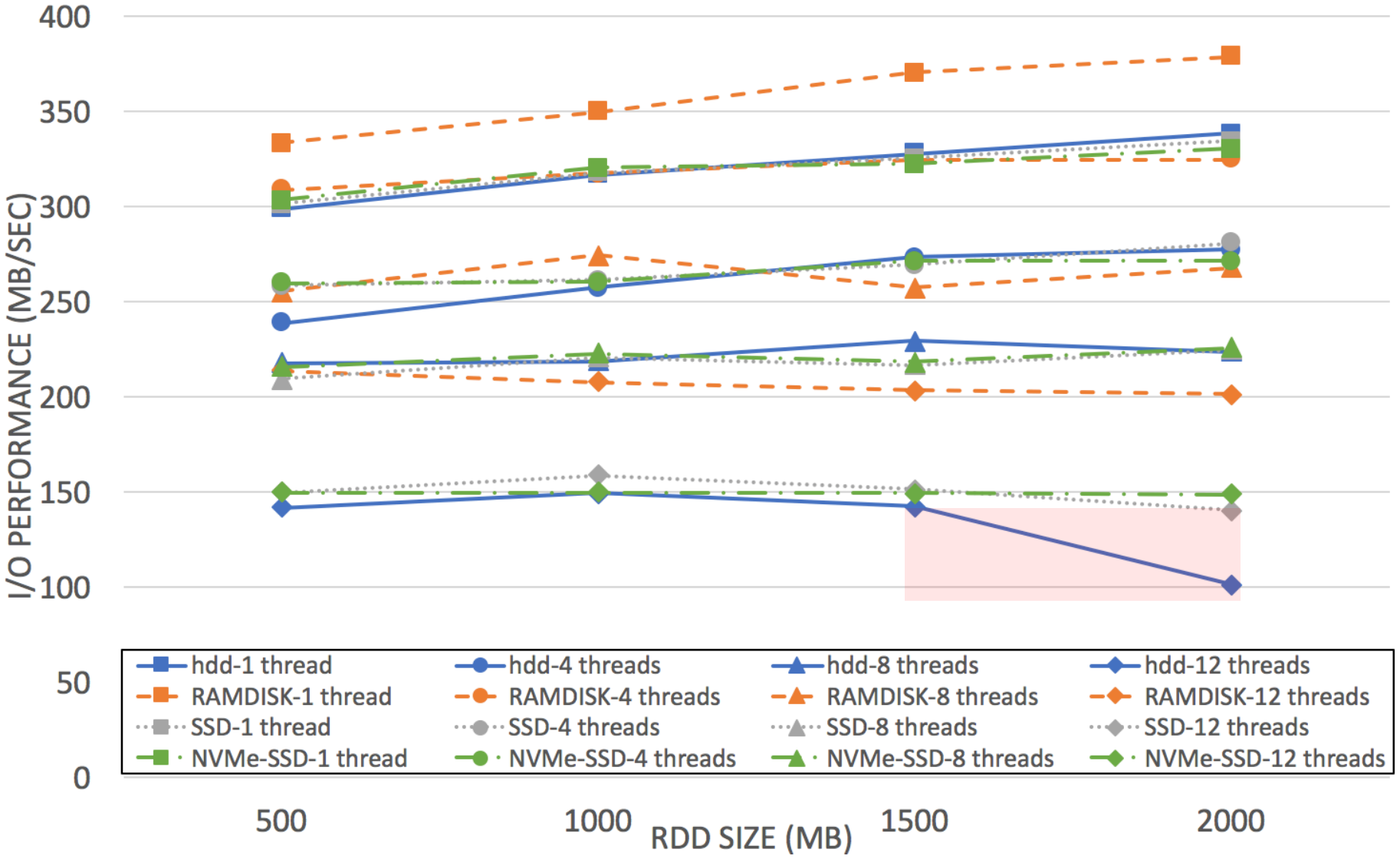
Persist の実行速度の比較(1 スレッド)



ストレージデバイスによる違い（実験３）

- 設定２
  - スレッド数：**4, 8, 12**
  - Driverのメモリ量：**64GB**
  - ストレージレベル：**DISK ONLY**
  - RDDのサイズ：**500MB,1,000MB,1,500MB, 2,000MB**
  - プログラム：**RDDTest**

Persist の実行速度の比較 (複数スレッド)



まとめ（1）

- RDDキャッシュのサイズに応じて，MEMORY_ONLY と DISK_ONLY の差が広がる

- RDDキャッシュにおいてメモリとディスクを併用する場合，DISK_ONLYを利用した場合より性能が低い場合がある
  - メモリが不足する場合にディスクへのドロップが多発し，さらにガベージコレクションの影響も受けるため，性能が大きく落ちる
  - この場合はディスクの利用は有効な選択肢の1つ

まとめ（２）

- RDDキャッシュにおいては,OS のバッファキャッシュがあるためにディスクの性能はボトルネックになりにくい
- 高速なディスクよりも容量の大きい HDD などを用いた方が高コストパフォーマンス
  - ただし、複数スレッドの場合はOS のバッファキャッシュの効果が低減
- RDD サイズとスレッド数の増加に対して，HDDでは性能が低下

今後の課題

- ディスクを利用した場合の高速化
  - メモリとディスクを併用する方式において，同じブロックが何度もドロップされないようにする
  - 新しいブロックを Persist する直前にドロップを実行するのではなく,他のRDD が処理されている間に非同期にドロップを行う

ご清聴ありがとうございました

本成果の一部は，国立研究開発法人新エネルギー・産業技術総合開発機構(NEDO)の委託業務の結果得られたものである．また，本研究の一部は JSPS 科研費 JP16K00116 の助成を受けたものである