NTT

Apache Hadoop エコシステムを中心とした分散処理の今と未来

小沢 健史

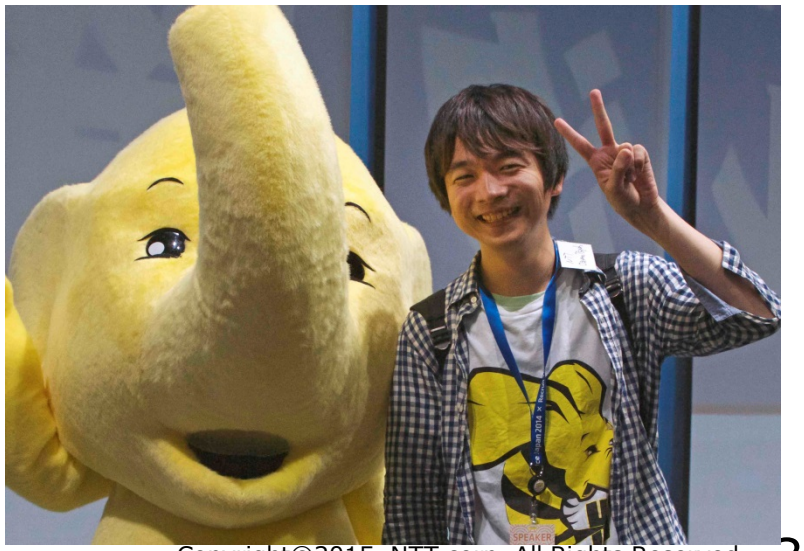
ozawa.tsuyoshi@lab.ntt.co.jp
ozawa@apache.org

アジェンダ

- **処理基盤の意義**
- **MapReduce の動向と進化**
  - MapReduce の概要
  - MapReduce の課題と解法
- **分散処理基盤の動向**
  - Google の処理基盤スタック
  - Microsoft の処理基盤スタック
  - オープンソースの処理基盤スタック
- **オープンソース開発**
  - なぜ会社として貢献するか
  - 活動内容
  - 面白かったこと，大変なこと
- **これからの目標**

About me

- 小沢健史(おざわつよし)
- Research & Engineer @ NTT
  Twitter: @oza_x86_64
- Apache Hadoop コミッタ(主要開発者)
- 著
  - “Hadoop 徹底入門 2nd Edition” Chapter 22(YARN)



処理基盤の目的

- よく行う処理を簡単に自動化したい
- 要件を満たす性能を出したい

- プログラミング言語
  - 例: Ruby/Python/Erlang/Scala/C etc…
- ライブラリ
  - 例: python + scikit/numpy
- フレームワーク
  - 例: MapReduce(Hadoop), MPI(OpenMPI) etc….
- RDBMS
  - 例: PostgreSQL/MySQL, Vertica, etc.

様々な処理基盤

- **なぜこんなに種類があるのか**
  - 要件がまちまちなので、使い分ける必要がある
    - 性能(遅延とスループット)
    - 使いやすさ(既存ツールとの親和性、インタフェース、インストールのしやすさ)
    - スケーラビリティ
    - 耐故障性

      etc…
- **この中でも，スケーラビリティに重きをおいているのが“分散処理基盤“**
  - スケーラビリティ ＋ α に重きを置いている処理基盤が多い

スケーラビリティとは

- **問題の規模の拡大・増大に対して対応できるかどうかという指針**
  - 用例
    - "ビジネスがスケールする”
    - "システムがスケールする”
- **システムにおけるスケーラビリティの例**
  - データ量の増大に対処するために，システムを拡大していくことができるか？
    - データ量が3倍になったときに、プログラムの改変なし・設備投資のみで想定時間内で処理を終わらせることができるか
  - システム規模拡大につれて，運用コストが跳ね上がらないか？
    - 1000台と2000台で気にしなければならないことの違いは？

分散処理基盤の目的

- **部分的な故障が起きても動き続ける**
  - データや処理を冗長化しておくことで、復旧が可能
- **処理のスループットを上げる**
  - 計算機を並べることで，処理を並列化

100Mbps x 10台 ＝　1Gbps で読み込みができる！

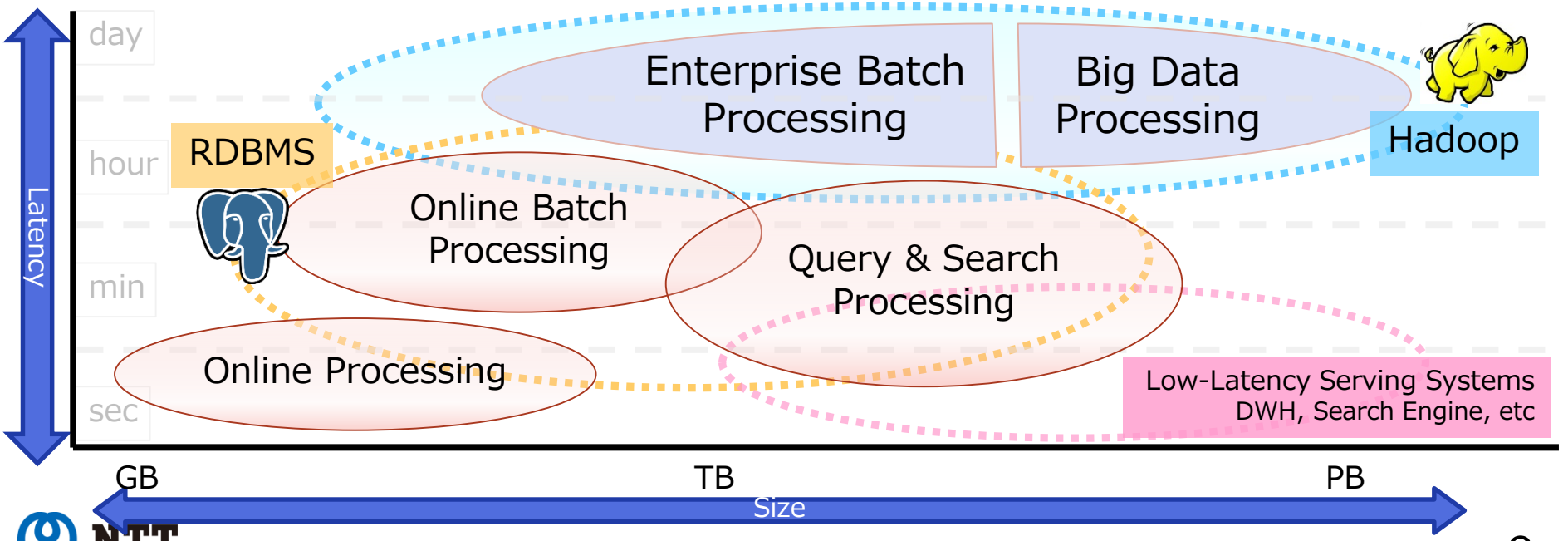
- **分散処理をしても処理が早くなるとは限らない**
  - 余計な通信遅延が発生しない分、1台で十分な処理は1台で行うべき
    - 一般的には
      CPU(nsec) > Memory(nsec) > Network(μsec-msec) > Disk(msec)
  - 光速の壁
  - ネットワークが飽和しがち

代表的な分散処理基盤

- **MPI(Message Passing Interface)**
  - 主な用途: 科学技術計算など
  - CPU インテンシブな処理向き
  - 通信の箇所をメッセージパッシングという形で隠蔽
- **並列DB**
  - 主な用途: クエリを低遅延で実行
  - 並列IO
  - インデックス，データ配置
- **MapReduce**
  - 主な用途: 大量のデータの変換を行うための基盤
  - 分散ファイルシステムを前提とした処理基盤で，並列IOによりスループットを稼ぐ
  - 故障処理、分散処理、IO をAPI (Map関数/Reduce関数) の下に隠蔽

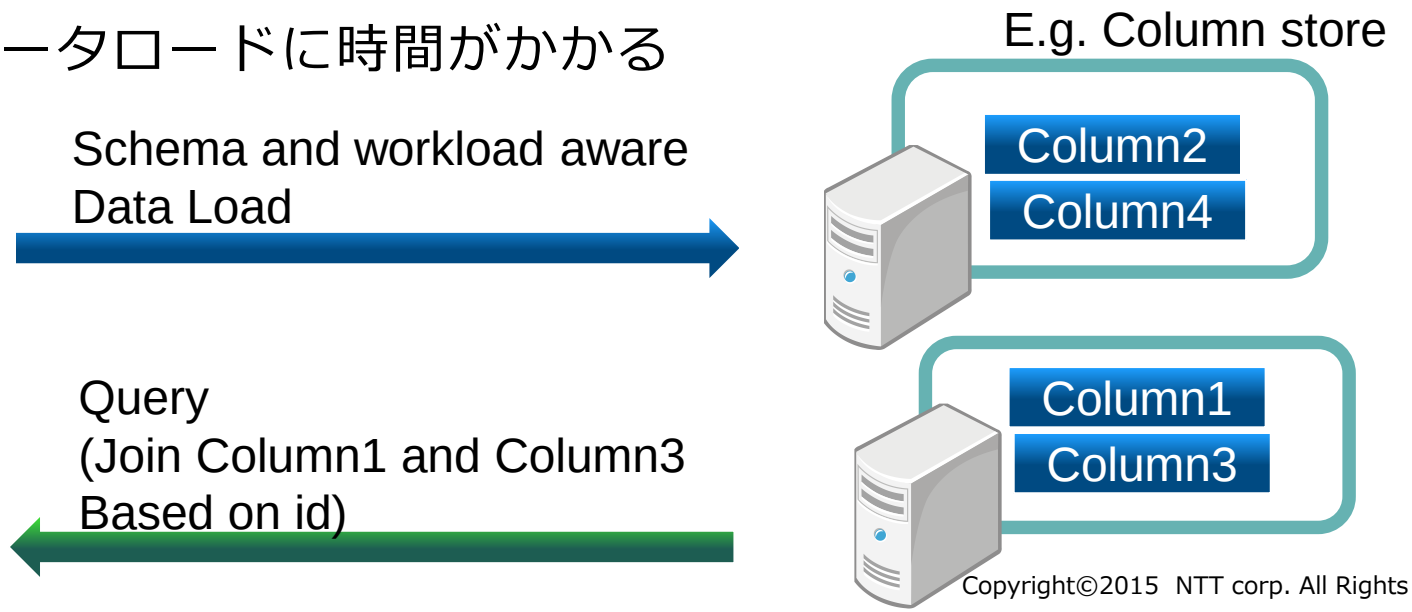
RDBMS/MapReduce(Hadoop) の事例

- **NTT DATAにおける使い分けの例**
  - MapReduce(Hadoop) は巨大なデータかデータが増えそうな見込みの処理に利用
  - RDBMS(PostgreSQL) も低遅延のジョブでも利用されるが、巨大すぎるデータでは利用できず
  - 並列DB は巨大なデータに対して高速に処理をかけたいときに利用



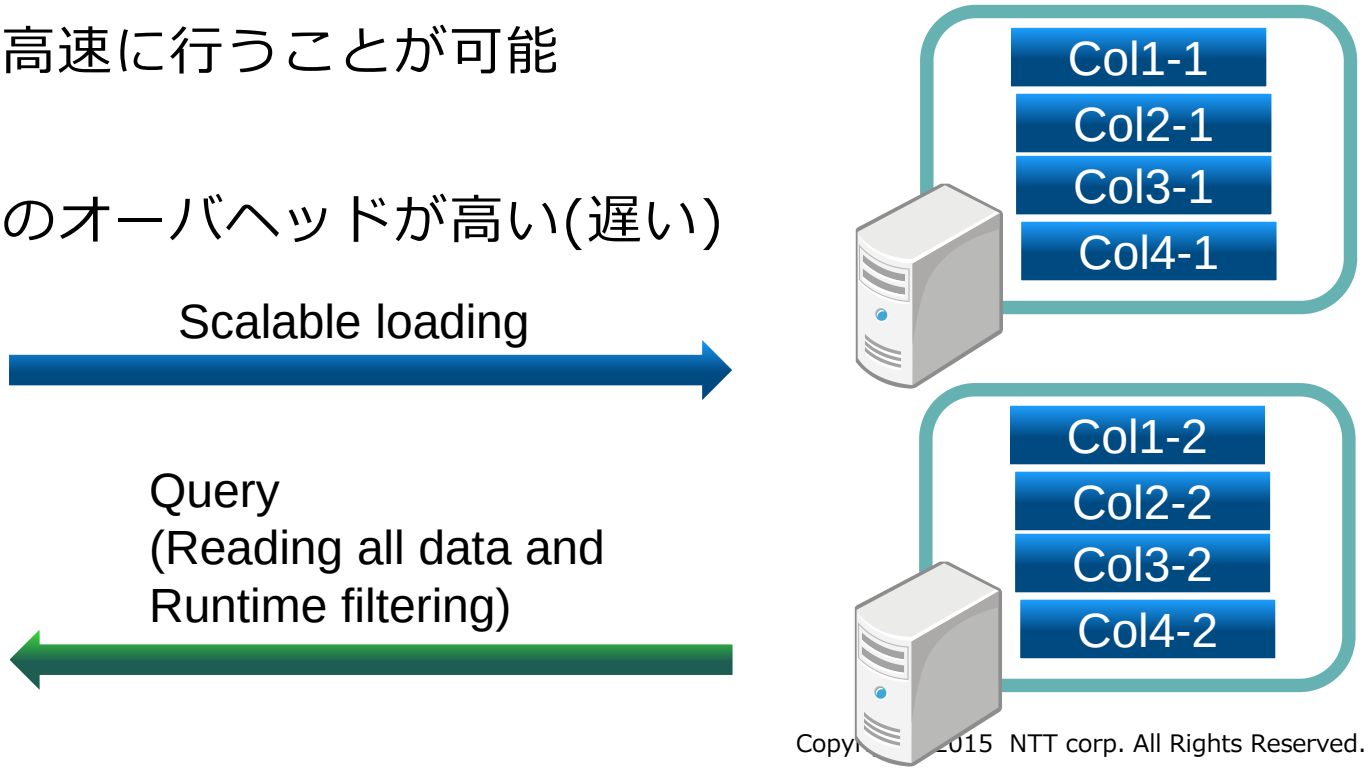
並列DBの特徴

- **Schema on Write**
  - スキーマ情報を意識して書き込んでおくことで問い合わせを高速に処理する
- **利点**
  - 問い合わせ時のオーバヘッドを最小化
- **欠点**
  - 設計時にスキーマや処理内容を意識する必要がある
  - データロードに時間がかかる



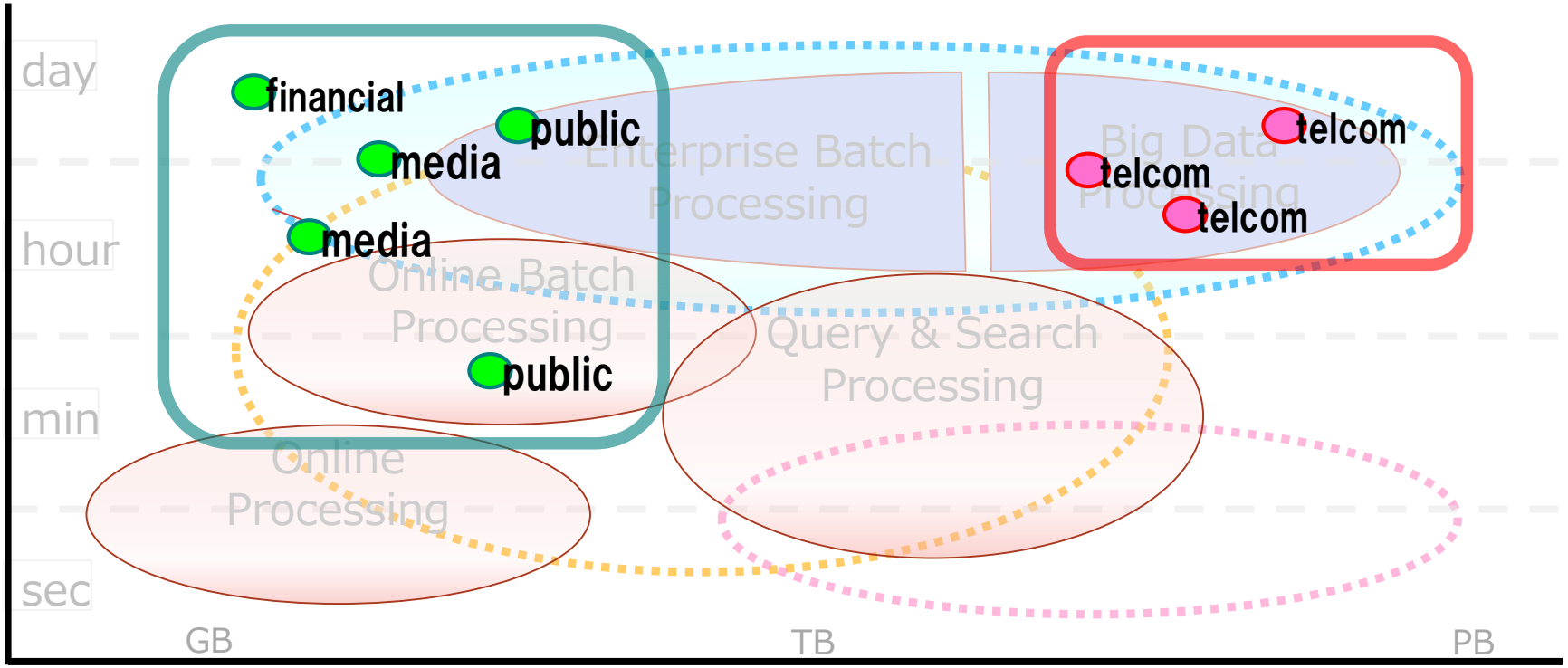
Hadoop の特徴

- **Schema on Read**
  - Hadoop のアプローチ
  - 書くとき適当、処理時にスキーママッピング
- **利点**
  - スキーマやワークロードの変化に強い
  - ロードを高速に行うことが可能
- **欠点**
  - クエリ時のオーバヘッドが高い(遅い)



実際の適用領域

- Hadoop の計算機の量は20 ~ 1200+ ノード,
  データ量は最大で 4PB



実情から見る Hadoop を選択する状況

- **処理しなければならないデータが巨大である場合**
  - 例: 一日100GBずつデータが増えていく
- **データベースに入れる前に、非構造化データに対して前処理を行いたい場合**
- **スキーマが頻繁に変更される場合**
  - 例: アプリケーションからのログ形式の変更が多い場合

- **これ以外の場合は、基本的にRDBMSを選択する方が無難**
  - 複数の計算機の面倒を見る必要がある
  - かえって処理が遅くなることもあり得る

MAPREDUCE

MapReduce が作られた経緯

- **MapReduce は元々 Google の内部システム**
  - MapReduce: Simplified Data Processing on Large Clusters
    - Jeffrey Dean and Sanjay Ghemawat, OSDI'04: Sixth Symposium on Operating System Design and Implementation
  - 今年で11周年(!)
- **背景より引用**
  - Google を構築するための rawデータ(ログ、クロールしたデータ)を処理するシステムを5年間作ってきた
  - 毎回100台レベルでスケールさせるのが面倒
    - 故障時の動作
    - データ分配の仕方
  - フレームワーク化したら便利になった
    - コードが短くなった
    - 開発期間が短くなった

MapReduce が作られた経緯

- **検索エンジンを作るときの構成要素**
  - 転置インデックス
    - キーワードからドキュメントを引くための目次
  - 問い合わせに対するランキング(スコアリング)

- **いろいろなキーに対するソート**
- **クロールしてきた HTML からデータを抽出**
- **集計**
  - 基本、メモリからあふれる
  - 1台の DB には到底入りきらない(100台規模)
    - 故障が起きうる
- **これらの処理を簡単に作るにはどうすればよいか？**

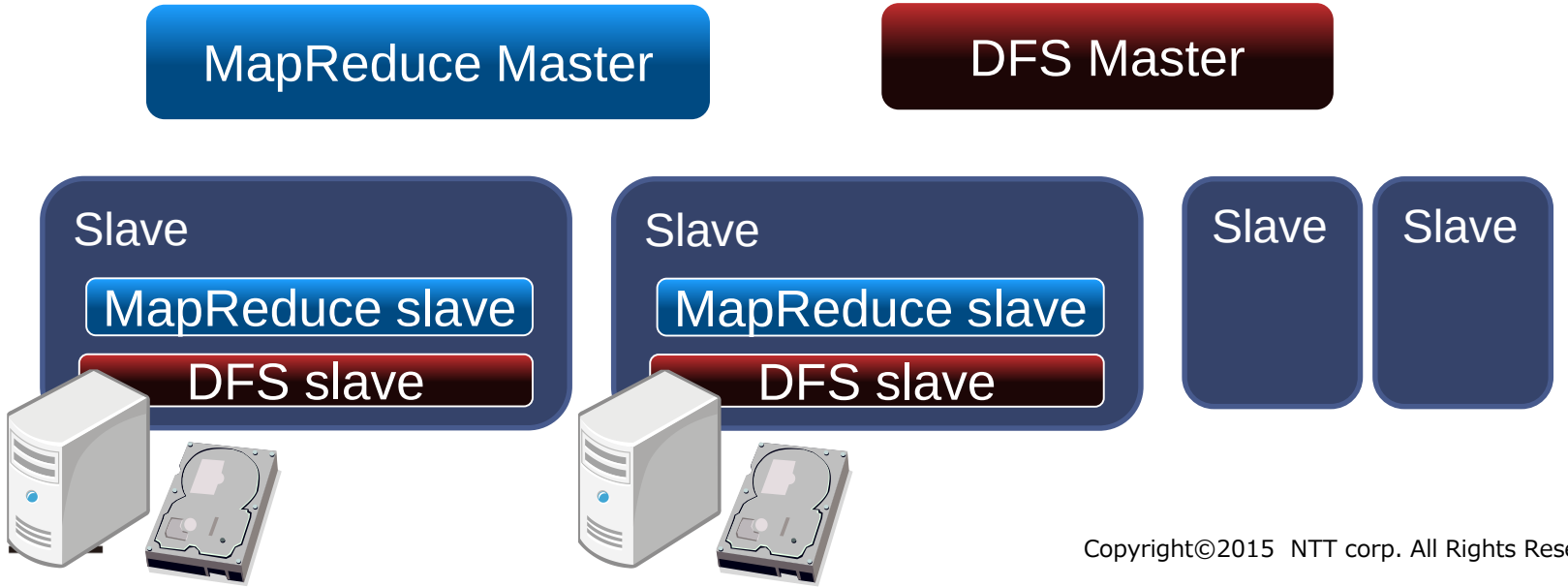
MapReduce の登場

MapReduce の考え方

- **前提**
  - 生データが相手なので事前計算はできない(索引など)
  - 計算機故障が頻発しうる
- **目標**
  - 計算機を足せばデータが増えても処理時間を一定にできる
    - ネットワーク通信量を最小にする
  - 計算途中で計算機が故障してもジョブが動き続ける
  - ソート・集計などがやりやすい
- **手段**
  1. なるべくデータをローカルから読めるように **分散ファイルシステム上に処理系を構築**
  2. 上記の処理を、すべて**簡易なAPI**でラップ

MapReduce のアーキテクチャ

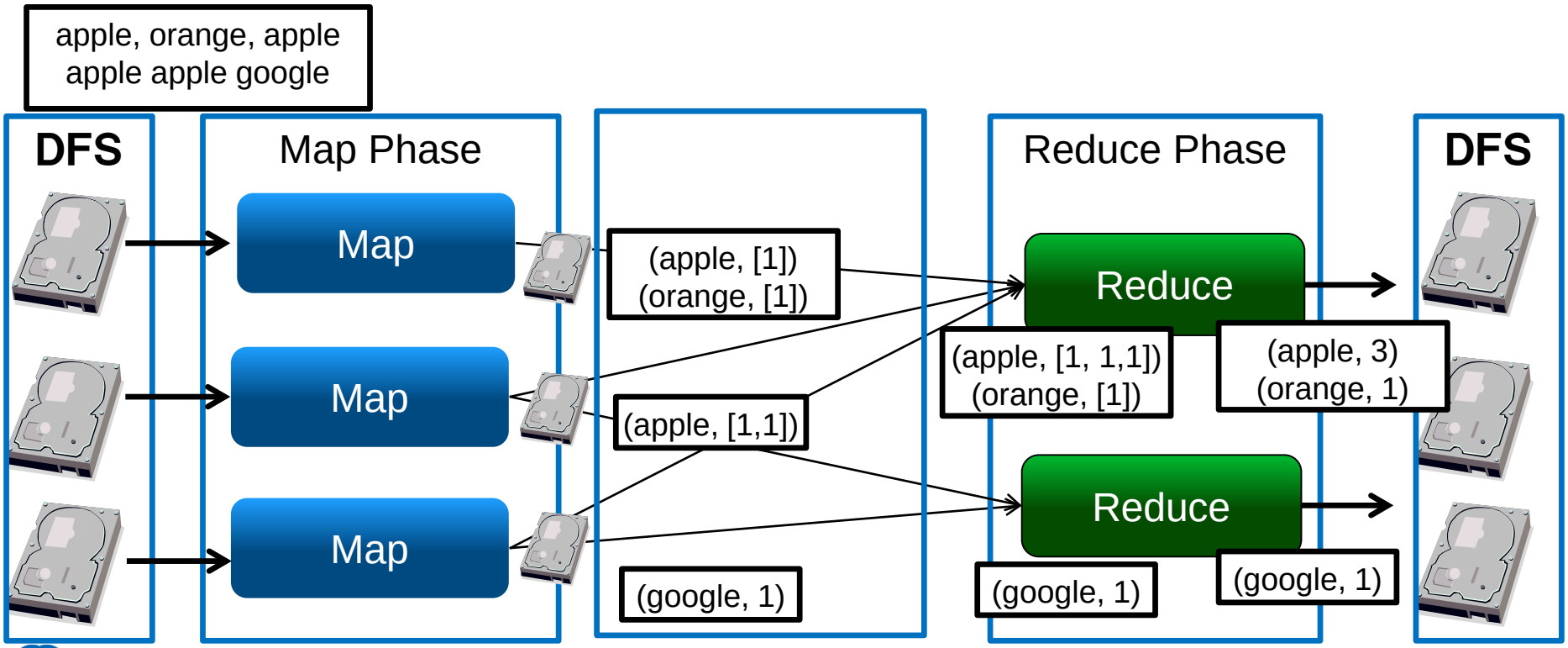
- **Master – Slave 型のアーキテクチャ**
  - Master は、ジョブの状態・FSの状態を管理して指示を出す
  - Slave は Master の指示に従って処理を実行
- **分散ファイルシステム(DFS)のSlaveと処理系(MapReduce)のSlaveを同居**
  - ローカルにあるデータはネットワーク通信なしで処理が可能
  - 必要なときは他の計算機のデータをコピーすることがある



MapReduce の動作フロー

1. ユーザは Map 関数と Reduce 関数を書く
2. Map 関数と reduce 関数間は Key と Value のペアの受け渡し

ドキュメントから単語頻度を数え上げる MapReduce の例



MapReduce の課題とその解法

- **課題1: プログラムの書き方がトリッキー**
  - 問題を分割統治できるようにモデリング・変形する必要がある
- **→専用言語の登場**
  - Script languages on MapReduce
  - SQL on MapReduce
  - 宣言的言語を MapReduce に変換

- **例 Sawzall, Hive, Pig etc.**

MapReduce の課題とその解法

- **課題2: 遅い**
  - 多段の MapReduce を実行する際、プログラムとプログラムの間で分散ファイルシステムにデータを書き込む必要
- **→1.有向非循環グラフによる効率の良い実行エンジンの登場 (Better MapReduce)**
- **→2. 専用言語を前提に MapReduce の段数を減らすような最適化(コンパイラの改良)**
- **→3. 使い方を絞って高速化 (特殊化)**

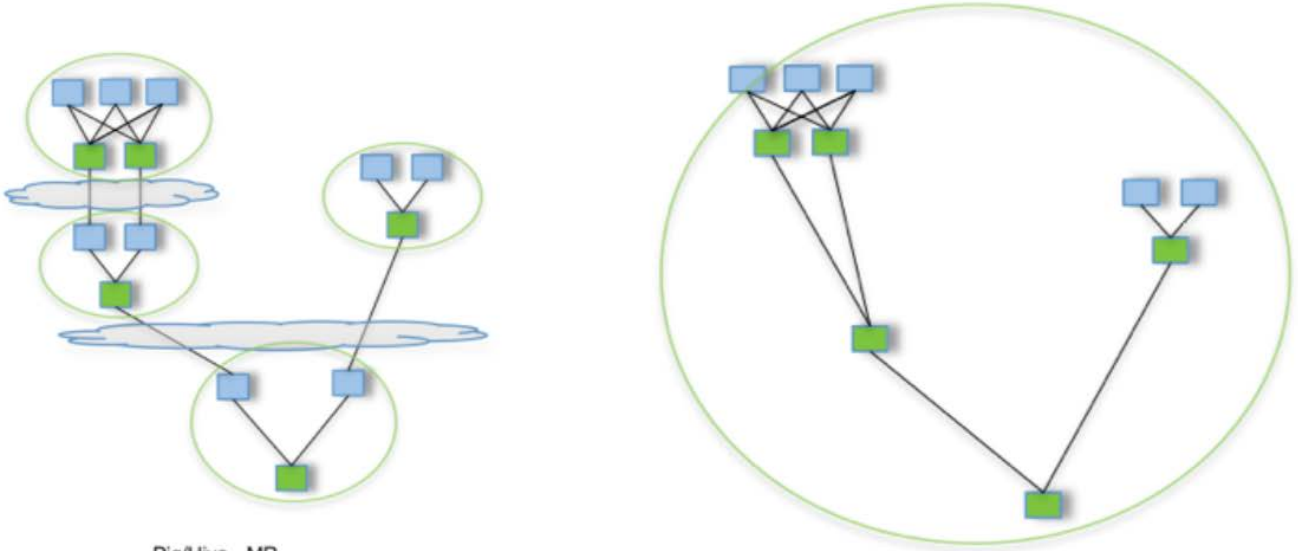
有向非循環グラフによる効率の良い実行エンジン

- **有向非循環グラフ: Directed Acyclic Graph**
  - 点・処理内容
  - 辺・データの受け渡し

・多段の MapReduce は DAG そのもの

・それを直接処理できるようにすれば良い

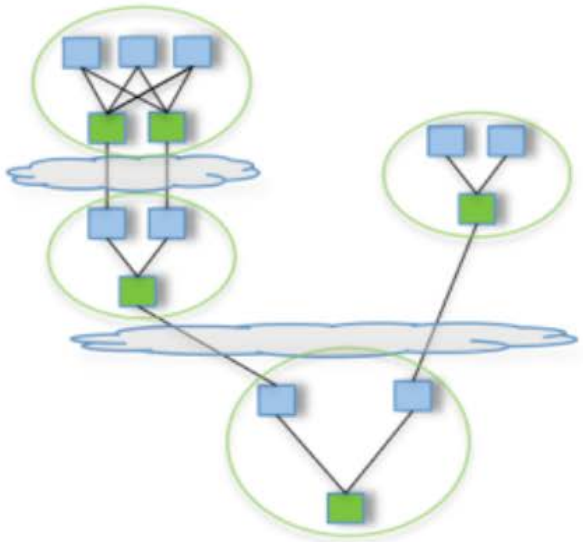
関連技術:
Dryad, Tez,
Spark



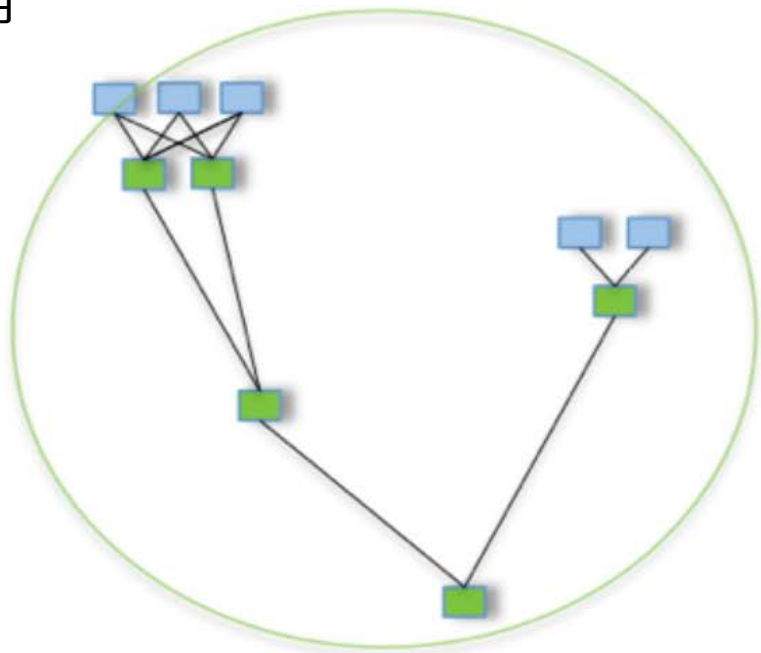
図は http://tez.apache.org/ より引用

専用言語を前提にMapReduce の段数を減らすような工夫

- **例: YSmart: Yet Another SQL-to-MapReduce Translator[Rubao et. al ICDCS '11]**
- 基本的にはデータベースの技術を利用し、それを MapReduce に応用
  - MapReduce の各段に当たる箇所を、データの再利用できる箇所を自動的に検出してうまく減らす
    図は http://tez.apache.org/ より引用



Pig/Hive - MR



Pig/Hive - Tez

使い方を絞った高速化

- **例: Dremel: Interactive Analysis of Web-Scale Datasets [Sergey et. al VLDB '10]**

- **発想**
  - 耐故障性が不要な場合・重要性が低い場合は MapReduce は不要
    - データ全体に対する傾向を見て、試行錯誤するとき
      - 故障解析
      - アクセス解析 など

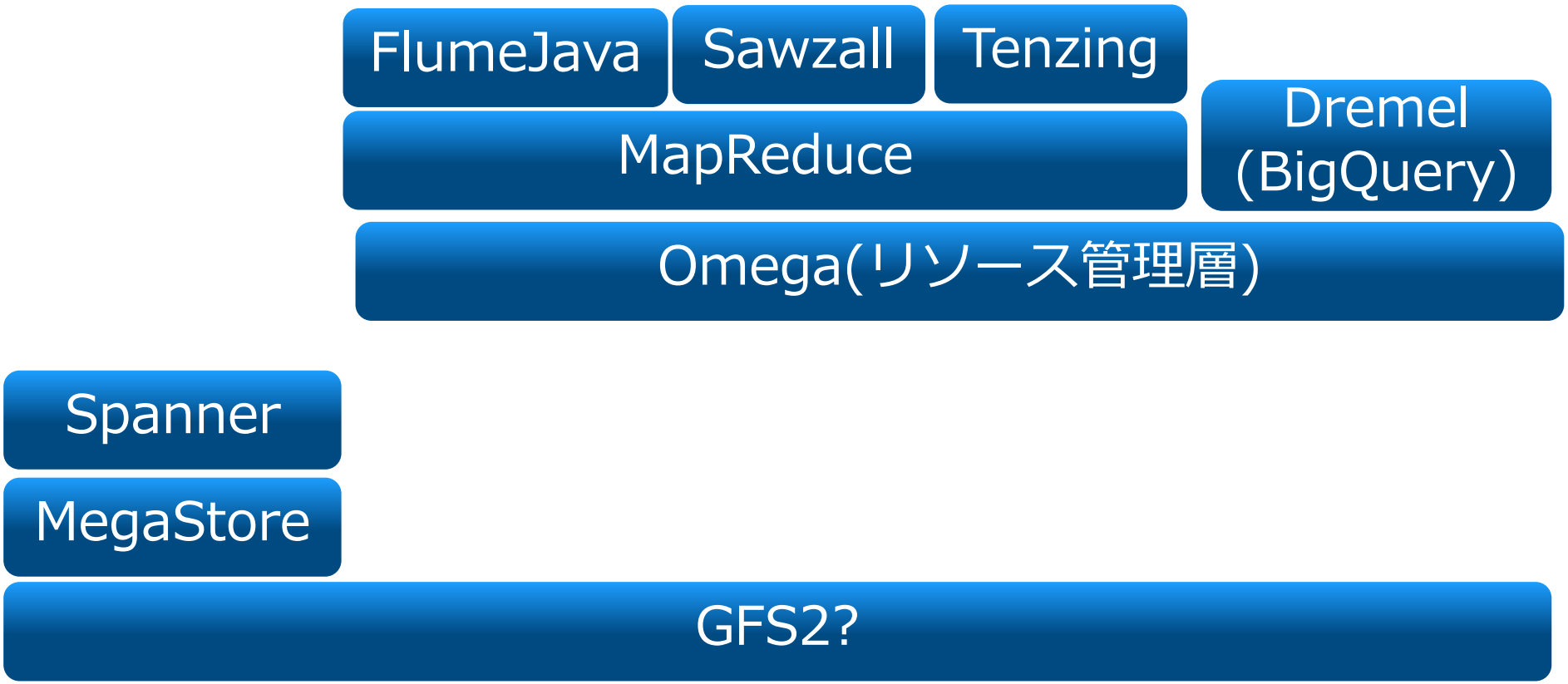
- **アプローチ**
  - 分散ファイルシステム部分を利用して、その上に分散データベースの実行基盤を構築
    - 短い実行時間のクエリが失敗した場合は再度実行する前提がとれるので、ディスクに中間データを可能な限り書かない
    - 読み込みに特化してデータ構造を最適化
  - 関連技術: Impala, Presto, Parquet, ORCFile etc.

処理基盤の動向

Google の処理基盤に関する論文

- Google における変遷(論文ベース)
  - MapReduce [OSDI’ 04]
  - Sawzall[Sci. Program’ 05]
    - MapReduce 用の専用言語
  - FlumeJava [PLDI’ 10]
    - MapReduce パイプラインを段数の少ないMapReduceにマッピングするライブラリ
  - Dremel[VLDB ’ 10]
    - インタラクティブクエリ用の非MapReduceの処理基盤
  - Tenzing [VLDB’ 11]
    - MapReduce の SQL インタフェース
      + それに最適化された MapReduce 処理系
  - Omega [EuroSys 2013]
    - リソース管理層
      - スケジューラの平行性能の向上

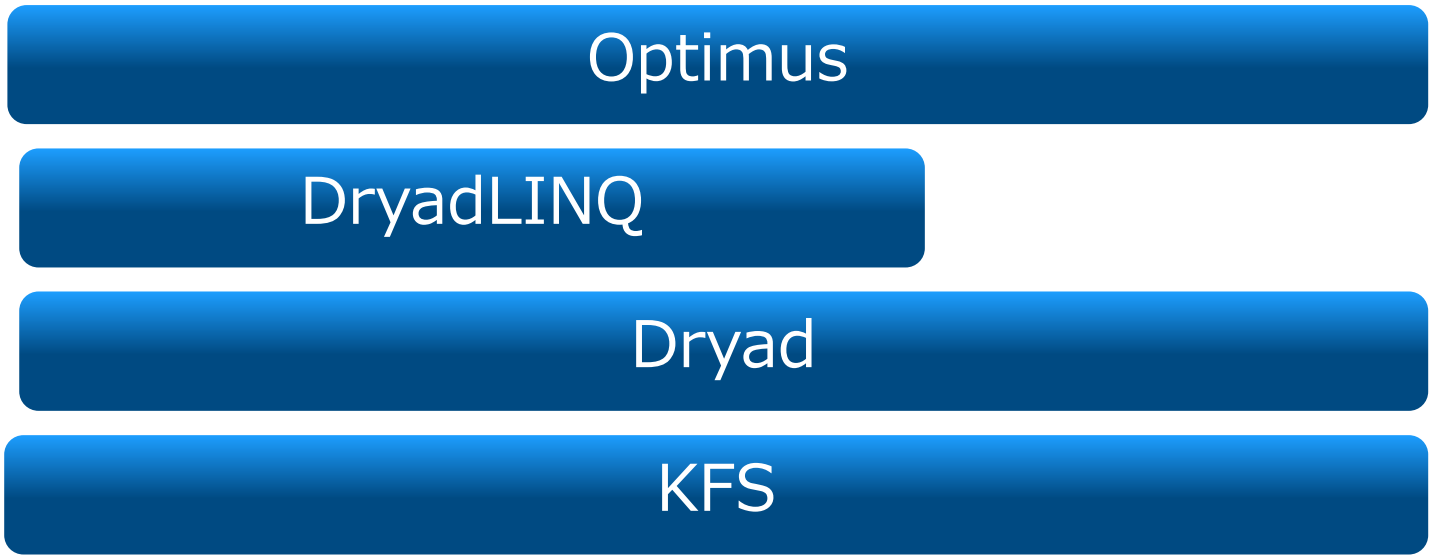
Google の処理基盤スタック(論文より推測)



Microsoft の処理基盤に関する論文

- Microsoft の変遷(論文ベース)
  - Dryad [EuroSys’ 07]
    - 非循環グラフで処理を記述可能な MapReduce よりも汎用的なフレームワーク
  - DryadLINQ [OSDI’ 08]
    - LINQ で Dryad の DAG を記述できる Optimizer + コンパイラ
  - Optimus [EuroSys’ 13]
    - LINQ インタフェースから， DAG を動的書き換え可能にするフレームワーク

Microsoft の処理基盤スタック(論文より推測)



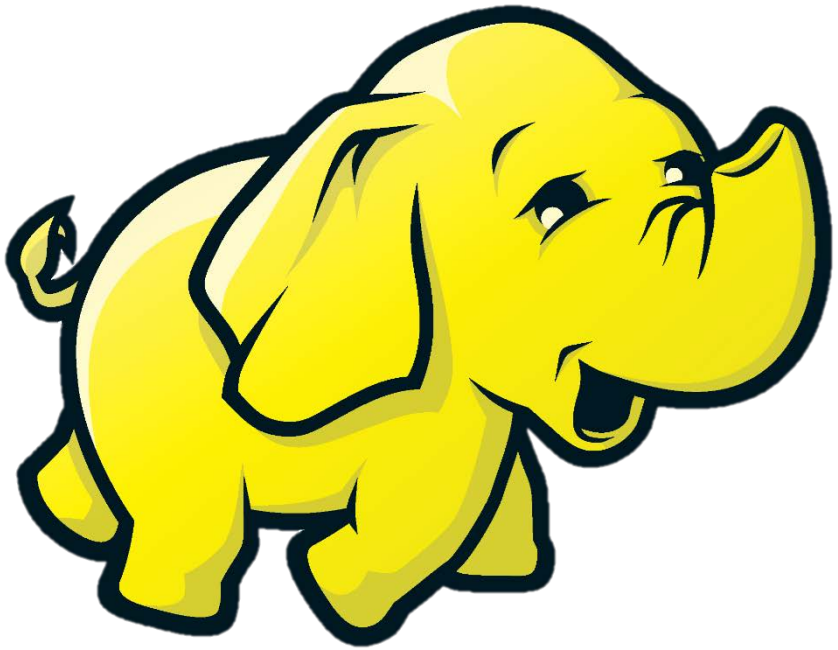
オープンソースソフトウェアの世界ではどうなっているか？

Hadoop とは

Apache Hadoop, and the logo are trademarks of The Apache Software Foundation.

Hadoop とは

- **Google MapReduce のオープンソース実装**
  - ソースコードが世界に向けて公開
  - 世界中の人々が開発
    - これまでに3000人ほどの人が開発に関与
  - 世界中の人々が利用
    - Facebook, Yahoo!, Twitter, LinkedIn, NTT, etc.
- **周辺プロジェクトが充実**



Hadoop エコシステム

HIVE

TEZ

APACHE HBASE

APACHE PIG

Spark
Lightning-Fast Cluster Computing

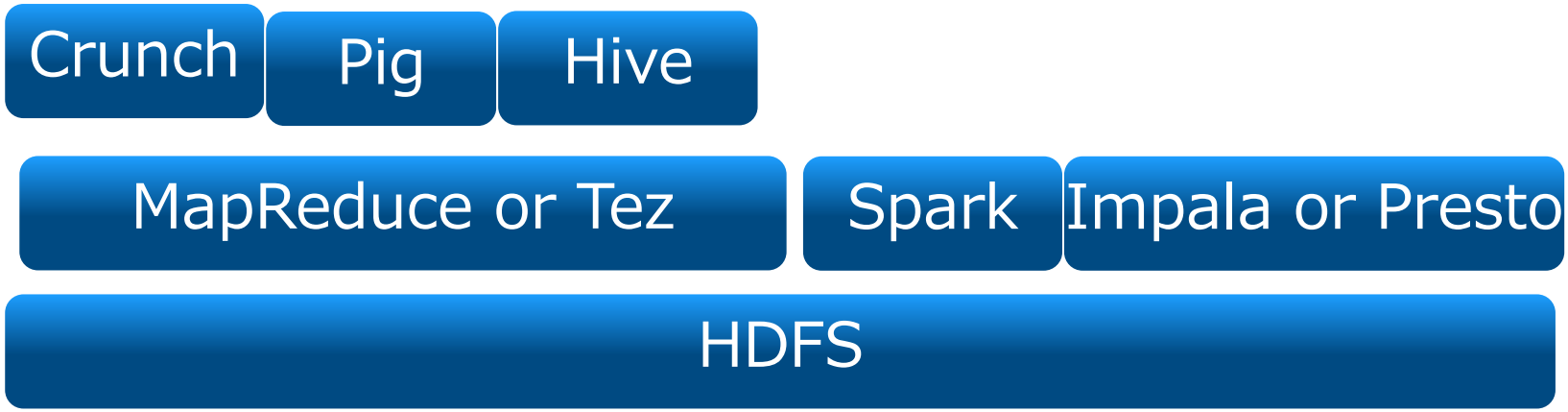
OOZIE

APACHE STORM™
Distributed • Resilient • Real-time

Hadoop の処理基盤スタック

- 2006 Hadoop
  - MapReduce のオープンソース実装
- 2008 Hive, Pig
  - MapReduce に変換する専用言語
- 2009 Spark
  - Dryad + DryadLINQ Ispired な DAG 処理基盤
  - とくにインメモリ処理に特化
- 2012 Impala
- 2013 Tez
  - Hive を前提とした DAG 処理基盤
- 2013 Presto
  - Dremel にインスパイアされた分散データベース

Hadoop の処理基盤スタック

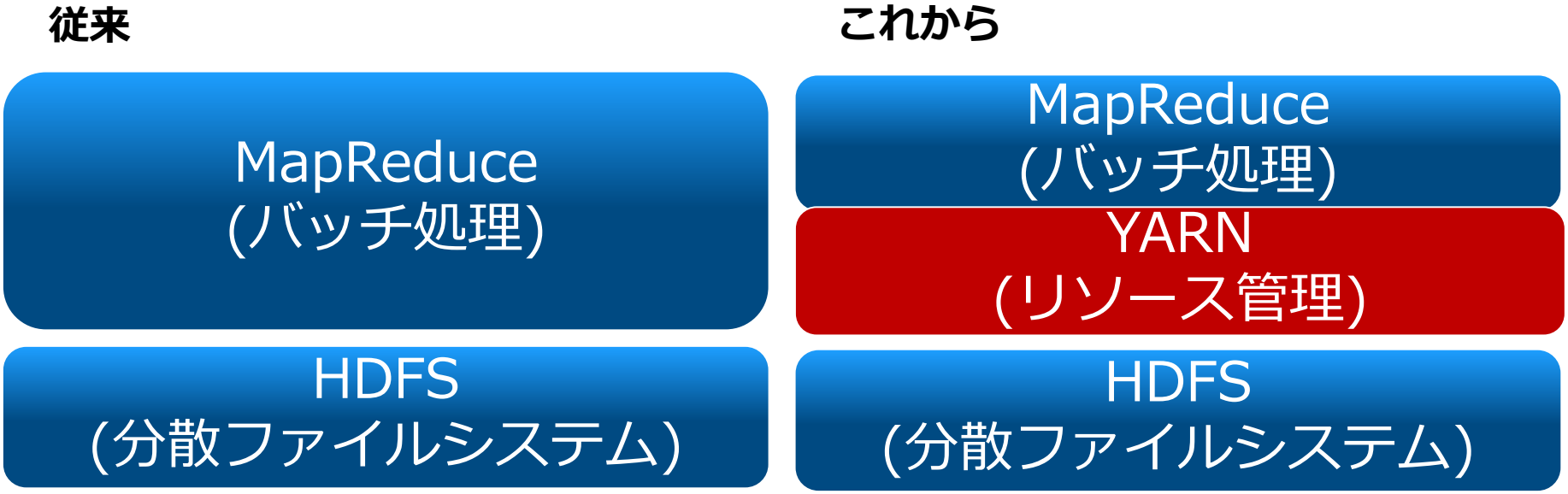


現状

- インタフェースとしては，HiveQL が前処理を行う上でもデファクトスタンダード
  - Pig は Hive が安定する前から使っているユーザが利用している
- MapReduce は利用されつつも，ゆるやかに別の処理基盤への移行が進行中
  - Apache Spark(インメモリベース)
  - Apache Tez(ディスクベース)
  - Impala/Presto(試行錯誤を多くする場合)
- **課題**
  - HDFS 上で様々な処理基盤が動作
    →データを動かさずに、いろいろな処理基盤を動かすと無駄が生じる
    → **Hadoop 上にリソース管理基盤である YARN が登場**

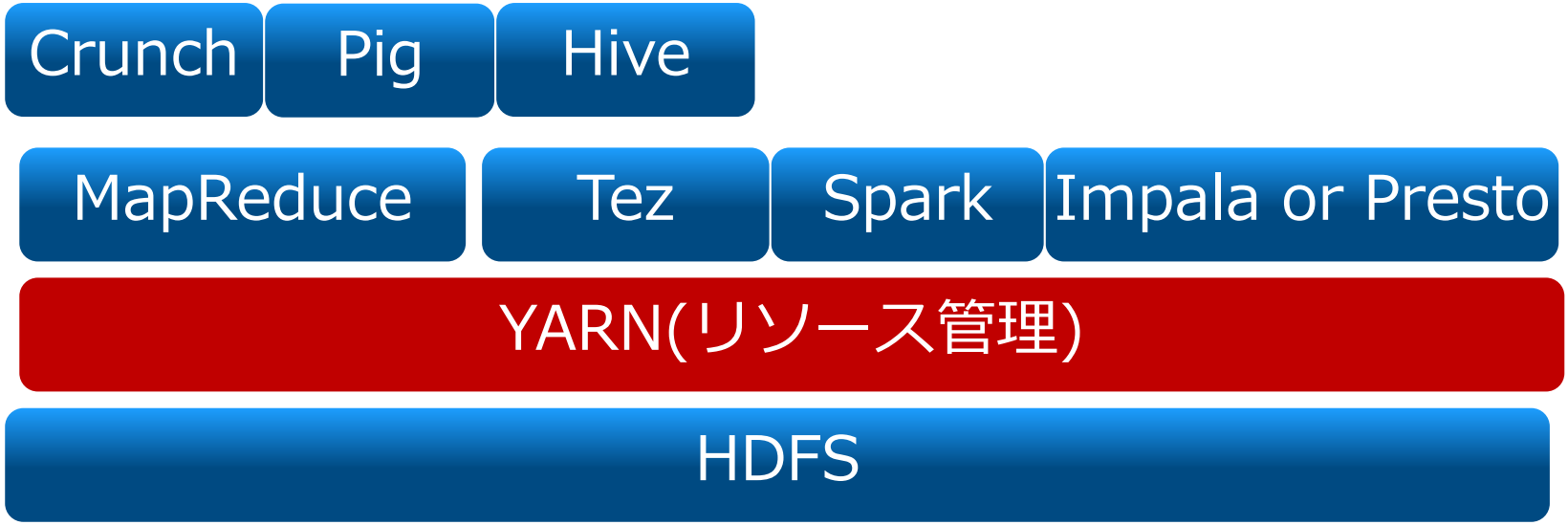
従来の Hadoop とこれからの Hadoop

- 並列分散処理を行う層から、
  リソース管理部分を切り出し



YARN が入ると…

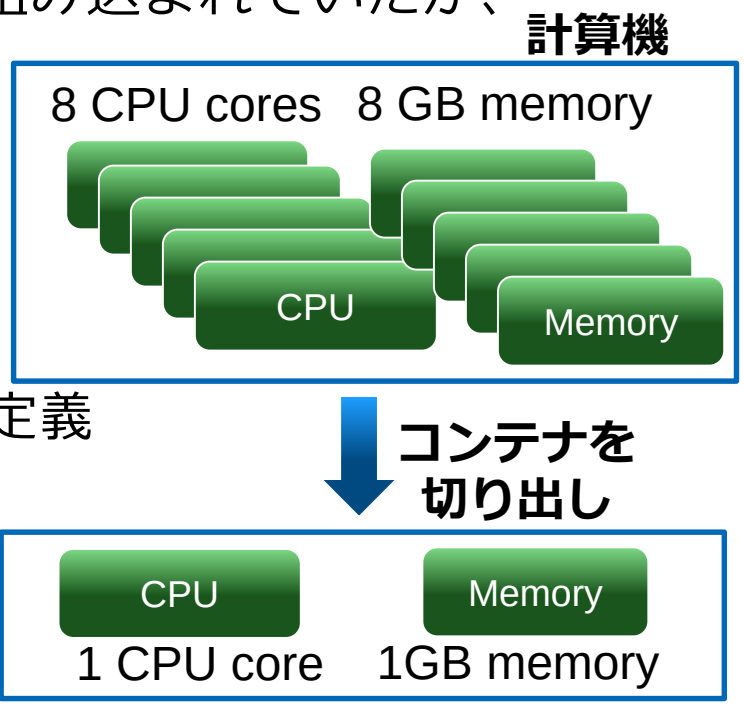
- データを移動せずに，さまざまな処理基盤が利用可能になる



リソース管理基盤の動向

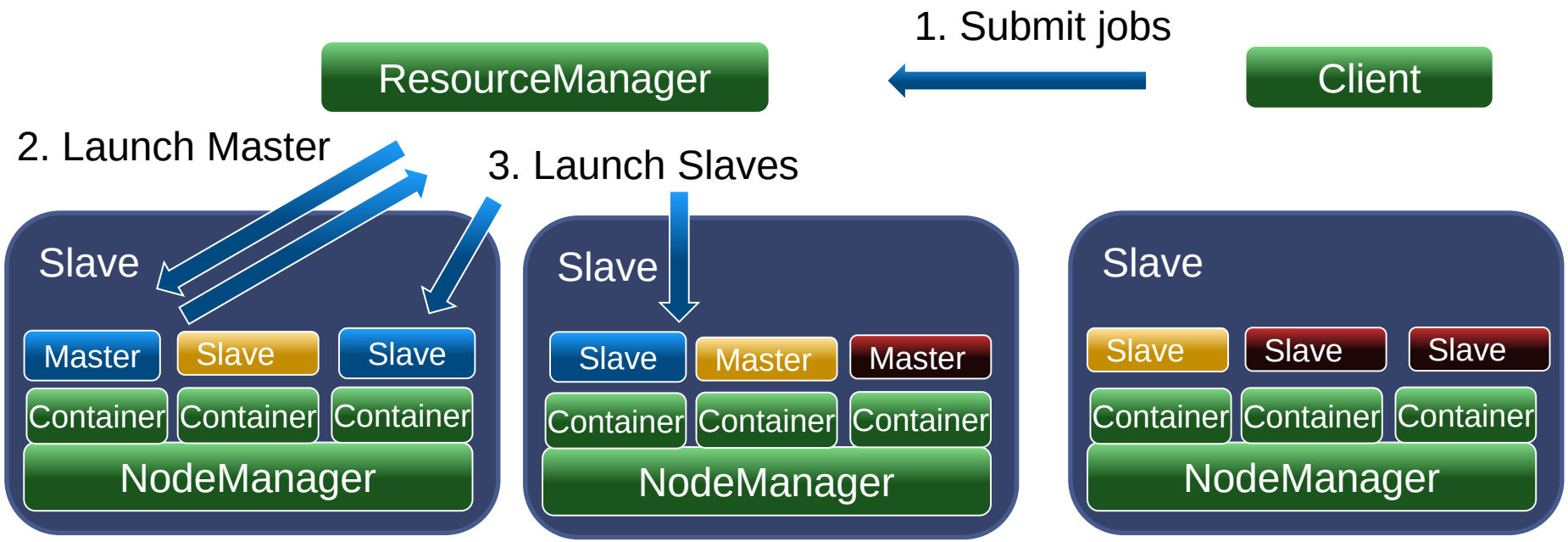
リソース管理基盤とは

- 複数の処理系が動作するようにリソース管理を統一的に行う技術
  - 従来の Hadoop/MapReduce にも組み込まれていたが、MapReduce 特有の管理方法を採用
    -> ほかの処理基盤に適用不可能
    - 「計算機ごとに Map 処理を M 個、Reduce 処理を R 個実行可能」
- **YARN**
  - 計算機ごとに使える計算リソースを定義
  - これを**コンテナ**として切り出し
  - **コンテナ**は汎用的な単位なのでどの処理基盤からも扱える



リソース管理基盤のアーキテクチャ(YARNの場合)

- **概要**
  - マスタがクラスタ全体のリソースを管理(ResourceManager)
  - スレーブが計算機ごとのコンテナを管理(NodeManager and Container)



リソース管理基盤に関する論文・またはOS

- Hadoop 系のリソース管理基盤の先駆けは 2010年の Mesos
  - いろいろな処理基盤を1つのクラスタ上でうまく動かすための技術
  - Spark と同じく amplab が公開
- 2011 年に Hadoop YARN が公開
  - Hadoop 上で動作する実装
- 2013 Google Omega の論文が公開
  - ジョブの同時実行数を向上するためのアーキテクチャと最適化技術の提案
- 2015 Microsoft より Mercury の論文が公開
  - 集中型と分散型のスケジューラをうまく切り替えてさらなるスケーラビリティを追求

リソース管理基盤の動向

- 基本的に Google, MS の技術は超分散並列方向に進化
- 1日辺り10億個のジョブを実行するための基盤 Mercury[Konstantinos et al. MSR-TR, 2015]

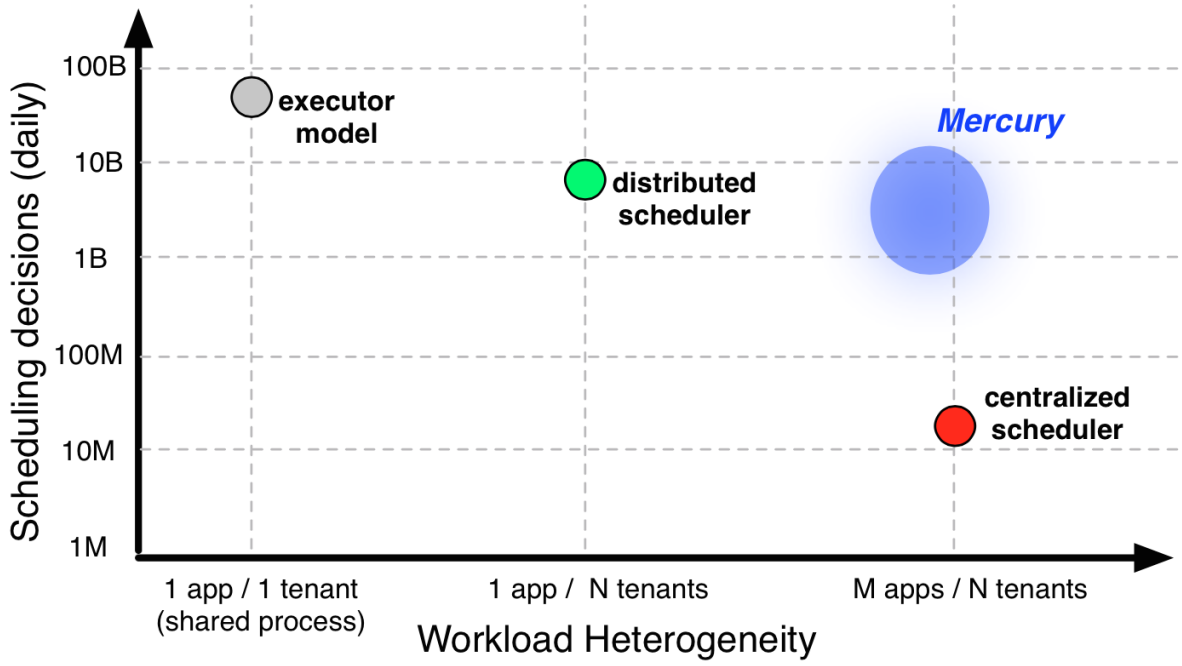


Figure 2: Ideal operational point of alternative scheduling approaches.

リソース管理基盤の関連技術

- **Quicny[SOSP ’09]**
  - Location aware な状態、かつ複数ユーザ間で Fairness を保証する技術
- **Dominant Resource Fairness: Fair Allocation of Multiple Resource Types[NSDI ’11]**
  - リソースが多次元(CPU + メモリなど)になったときに、複数ユーザ間でどのように公平にリソースを分ければ良いかを提言
- **Sparrow: Distributed, Low Latency Scheduling[SOSP ‘13]**
  - マスタを分散させるようなアーキテクチャをとることで超レイテンシクエリを処理できるようにしたスケジューラ
    - 10000コアくらいまでスケールすると主張

オープンソースソフトウェアの開発

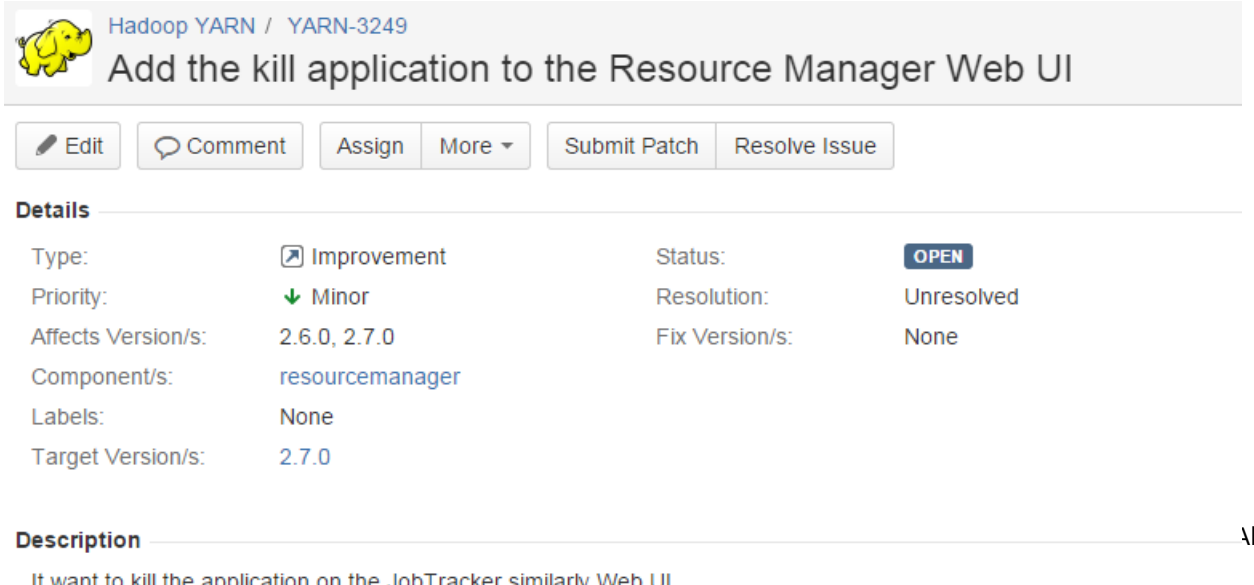
Hadoop プロジェクト概要

- **言語: Java**
  - ミドルウェアを書く言語として Java は優秀
- **Common：Hadoop 用のライブラリ**
  - 32万行
- **HDFS：分散ファイルシステム**
  - 70万行
- **MapReduce：MapReduce処理系**
  - 30万行
- **YARN: リソース管理部分**
  - 52万行
- **なかなかのサイズだが，Java なので読める**
- **エディタは IntelliJ IDEA などの IDE の利用をお勧め**

Hadoop の開発スタイル

- 基本はインターネットを介して行う

- JIRA と呼ばれるバグトラッキングシステムを利用して開発
  - github の issue みたいなもの
  - “Hadoop JIRA” で検索
  - まれに議論をメーリングリスト上で行うこともある



Hadoop の開発フロー

・コミッタ(ソースコードレポジトリの変更権限を持っている人)と貢献者(パッチを投げる人)が協力して行う

1. Issue をたてる (貢献者)
2. パッチを投稿　(貢献者)
3. レビュー(コミッタ)
4. コミット (コミッタ)

- 私は2014年12月にコミッタになったところで，新米コミッタ

誰がHadoopを作っているのか

Hadoop専業ベンダ、Hadoopを積極的に活用するサービス企業を中心に開発。
NTTデータ、NTTも世界有数のコントリビュータ(貢献企業)として活動。

2014年 Hadoopコミュニティ貢献指標

貢献コード行数

解決済みissue数

・商用環境においてHadoopを多数運用してきたことで得られた知見をもとに、Hadoopに対する改善提案をコミュニティにフィードバック．Hadoop本体への開発も実施．
運用上特に問題となるバグの改修や、利用者向けのドキュメントの拡充のためのパッチを投稿し、コミュニティに貢献．

なぜコミュニティに還元するか

- **ベンダとユーザ企業の優先順位は異なる**
  - ベンダは売りになるような機能を新規開発したい
    - 監視系・管理系・運用系は差異化ポイント
    - ドキュメント: 自社で配布しているものを読んでくれれば良い
  - ユーザ企業は今動いているものを安定して動作させたい
    - 監視系・管理系を充実させたい
    - ドキュメントは充実している方が良い

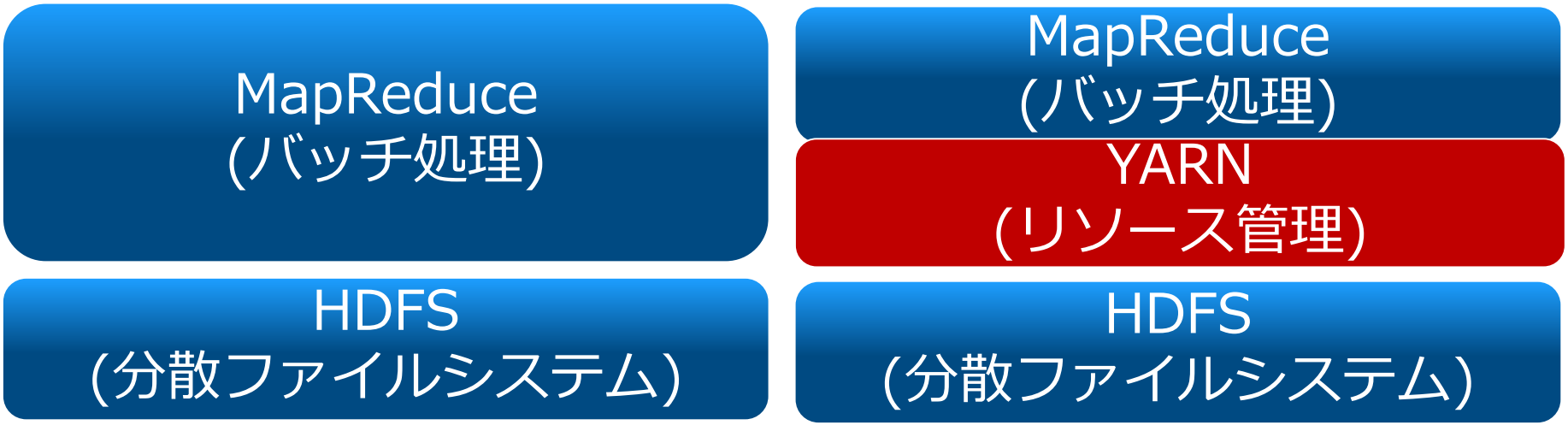
- **足りていない部分を自ら開発を行えば一番効率が良い**
  - 過程で得た内部のノウハウはそのまま差異化要素となる
  - たとえば…
    - 新機能の使い方，使いどころ，チューニングポイント
    - バグ修正を顧客の環境に反映するためのノウハウ etc.

Hadoop との関わり

- **2012年に上司の鬼塚さん(現阪大)と研究開発を開始**
  - Cloudera の方が開催する Hadoop のイベントにもぐりこんでパッチの投げ方などを教えてもらう
  - 成果を Pre Hadoop World/Strata Conference で発表
- **2013年からコミュニティ活動を本格的に開始**
  - YARNのマスタの高可用化に貢献
  - 第8回日本OSS奨励賞受賞
- **2014年9月にコミッタに推薦**
- **2014年12月コミッタに就任**
- **2015年3月**
  - 現時点までに約75件の修正をレビューし、コミット
  - 自分の行った変更は合計で約65件ほど

Hadoop (2012-2013年時点)

- **Hadoop 2系が主流になろうとコミュニティが奔走**
  - Hadoop 2.0 では YARN と呼ばれる新たなコンポーネントが導入



理想(設計・ドキュメント)と現実(実装) (2013)

- **「YARN は単一障害点がないように設計されている」**
  - 実装的に**単一障害点が増えていた**
  - 高可用構成を行うコードは **TODO としてコメントアウト**

- **「テストが通ったらレビューをしてもらえる」**
  - 変更していない箇所のテストがこける
    - そちらのテスト修正から直す…?
  - マニアックな変更をしてもみんな忙しすぎて
    レビューしてもらえない

- **「ビルドの方法は BUILDING.txt に書いてある」**
  - ドキュメントが古くてどうしたら良いかわからない
  - コミッタしかしらないオプションが…

対処

- **単一障害点が増えていた・ビルド方法が不明**
  - コミュニティと連携しながら修正
  - わかりづらい部分も指摘して自ら修正
- **変更していない箇所のテストがこける**
  - 未だに未解決だが，たまにこけるテストを見つけ出すスクリプトを作ってくれる人が出現
- **レビュワーがつかない**
  - 直接会って交渉
  - コミュニティの流れに乗るように開発する
    - 議論が盛り上がったらチャンス！
  - パッチを投げまくってると話題になる…らしい
    - また Tsuyoshi からパッチが来ているぞと話題になったと言われたことがある

2年間の貢献量

- **これまでに生成したパッチの数: 約600個**
- **行数にすると約16万行(重複含む)**
  - 重複含む
  - 約 219 line/day
  - それほど多くない
  - Minor fix やリファクタリングを含む

- **Hadoop Summit/Hadoop World やベンダ(Cloudera/Hortonworks)のオフィスにいくとオフ会状態になる**
  - 「おまえがアイツか！」
  - 「あのパッチはどうなった」
  - 「レビューありがとう！」
  - などなど.

これからの目標

- **まだまだスタート地点にたったところ**
- **事業的には Apache Hadoop の PMC (リリース権限がある人)になっていきたい**
- **オープンソースをうまく利用して R&Dを回すロールモデルになっていきたい**
  - 目標:
    - Matei Zaharia(MIT, Spark の作者，Hadoop のコミッタ)
    - Chris Dougras(Microsoft, Hadoop PMC)

    のような骨太なシステム研究者として
    世の中にインパクトのあるシステムを出していきたい