# データクラウドを支える技術と研究動向
# -- MapReduce 編 --

鬼塚　真
主幹研究員(特別研究員)
NTTサイバースペース研究所



# 目次

- MapReduce とは？
  - MapReduce のデザインパターン
- MapReduce 高速化の研究動向
  - 標準偏差 / PageRankの計算例
  - MapReduce の関連技術の研究動向
- MapReduce 高速化の取り組み
  - Map Multi-Reduce: reduce の事前実行
  - PJoin: 事前パーティション分割と準結合の利用
  - 複数分析処理におけるMapReduce最適化
- 研究の方向性

# MapReduce の概要

skip



# MapReduce とは?

- 分散処理のプログラミングモデル＆システム
  - 高スケール性: 1万ノード(マシン), ペタバイトデータ
  - シンプルなAPI: map関数/reduce関数
  - 分散処理特有の複雑さ(負荷分散・可用性)が隠ぺい
- Google が開発
  - webの検索エンジンのバックエンドの用途のため開発
    PageRank 計算, 転置ファイル構築
  - 2008年時点で 20PB/day を MapReduceで処理
  - [Dean et al., OSDI 2004, CACM Jan 2008, CACM Jan 2010]

**MapReduce**: Simplified data processing on large clusters
J Dean... - Communications of the ACM, 2008 - portal.acm.org
1 Introduction Prior to our development of **MapReduce**, the authors and many others at Google implemented hundreds of special-purpose computations that process large amounts of raw data, such as crawled documents, Web request logs, etc., to compute various kinds of ...
Cited by 2183 - Related articles - BL Direct - All 269 versions

- Hadoop project においてオープンソース化

# MapReduce の適用例

- n-gram によるWeb コンテンツ分析
  - 日本語 30億ページ
  - web 上で多い文章は何か？
  - 辞書: 音声認識，日英翻訳
- Web のクエリ・アクセスログ
  - 検索連動広告 (ad-words)
  - 利用者に適したコンテンツは？
  - パーソナライズ検索
- ネットワークトラフィック解析
  - 40Gbps ネットワーク
  - QoS 制御, 障害解析

4-gram

serve as the incoming 92
serve as the incubator 99
serve as the independent 794
…

検索ワード　検索連動広告

“焼酎” → 江戸切子 (edo kiriko)

パケット遅延，パケット欠損

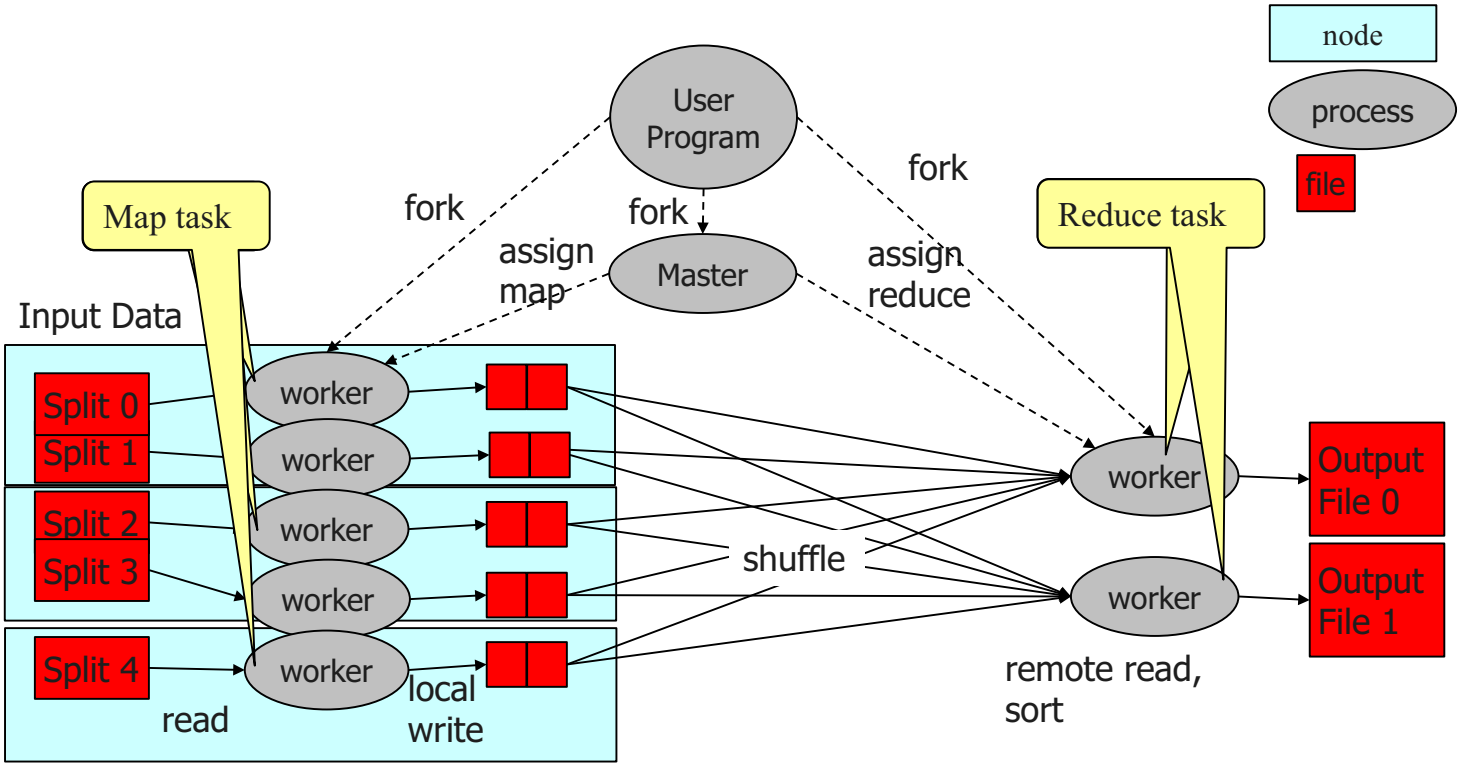


# MapReduce のアーキテクチャ

- MapReduce: 分散処理システム
- 分散ファイルシステム DFS 上で構築
  - DFS: GFS (Google FS), HDFS (Hadoop DFS)
- 2フェーズ + Shuffle
  - Map フェーズ: 入力レコード毎にmap関数を実行
    ポイント: **マシン毎に独立した処理**
  - Shuffle:同一key の(key, value) 群を束ねる
  - Reduce フェーズ: 束ねた結果に reduce 関数を実行
    ポイント: **複数マシンにまたがり必要な処理**

# MapReduce のアーキテクチャ(続き)








# プログラム例: wordCount

入力レコード毎に map関数を適用する

```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");

reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

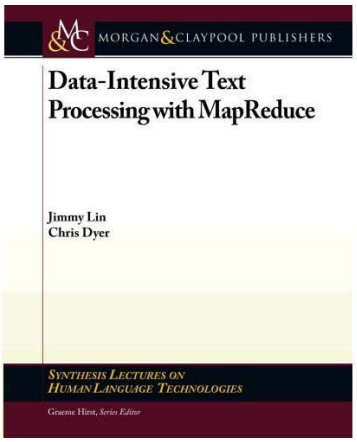
単語毎に (word, 1) を出力する

単語毎に頻度を積算する

[Dean et al., OSDI 2004]

# MapReduce のデザインパターン

- ある種の高速化のテクニック集 + 実アプリ
- Data intensive text processing with MapReduce, [Lin et al., 2010]
  - Local Aggregation
  - Pairs and Stripes
  - Computing Relative Frequencies
  - Secondary Sorting
  - Relational Joins





# Computing Relative Frequencies

- 条件付き確率 $P(X|C_i)$を計算する

# Naïve Bayesian Classifier: Training Dataset

Class:
C1:buys_computer = 'yes'
C2:buys_computer = 'no'

Data sample
X = (age <=30,
Income = medium,
Student = yes
Credit_rating = Fair)

| age | income | student | credit_rating | comp |
|---|---|---|---|---|
| <=30 | high | no | fair | no |
| <=30 | high | no | excellent | no |
| 31…40 | high | no | fair | yes |
| >40 | medium | no | fair | yes |
| >40 | low | yes | fair | yes |
| >40 | low | yes | excellent | no |
| 31…40 | low | yes | excellent | yes |
| <=30 | medium | no | fair | no |
| <=30 | low | yes | fair | yes |
| >40 | medium | yes | fair | yes |
| <=30 | medium | yes | excellent | yes |
| 31…40 | medium | no | excellent | yes |
| 31…40 | high | yes | fair | yes |
| >40 | medium | no | excellent | no |



# Naïve Bayesian Classifier: An Example

- $P(C_i)$: P(buys_computer = "yes") = 9/14 = 0.643
  P(buys_computer = "no") = 5/14= 0.357

- Compute $P(X|C_i)$ for each class
  P(age = "<=30" | buys_computer = "yes") = 2/9 = 0.222
  P(age = "<= 30" | buys_computer = "no") = 3/5 = 0.6
  P(income = "medium" | buys_computer = "yes") = 4/9 = 0.444
  P(income = "medium" | buys_computer = "no") = 2/5 = 0.4
  P(student = "yes" | buys_computer = "yes) = 6/9 = 0.667
  P(student = "yes" | buys_computer = "no") = 1/5 = 0.2
  P(credit_rating = "fair" | buys_computer = "yes") = 6/9 = 0.667
  P(credit_rating = "fair" | buys_computer = "no") = 2/5 = 0.4

- **X = (age <= 30 , income = medium, student = yes, credit_rating = fair)**

**$P(X|C_i)$ :** P(X|buys_computer = "yes") = 0.222 x 0.444 x 0.667 x 0.667 = 0.044
P(X|buys_computer = "no") = 0.6 x 0.4 x 0.2 x 0.4 = 0.019
**$P(X|C_i)*P(C_i)$ :** P(X|buys_computer = "yes") * P(buys_computer = "yes") = 0.028
P(X|buys_computer = "no") * P(buys_computer = "no") = 0.007

**Therefore, X belongs to class ("buys_computer = yes")**

# Computing Relative Frequencies

条件付き確率 $P(X \mid C_i)$を計算する

$$P(X \mid C_i) = P(X \cap C_i) / P(C_i)$$

$P(C_1)$: コンピュータ購入する確率
$P(age = \text{“<=30”} \cap C_1)$: 30歳以下で購入

どうやって MapReduce 上に実装すれば良いか？

1. 直観的な方法: iteration を2回実行する
   - 1st iteration: $P(C_i)$を計算する
   - 2nd iteration: $P(X \cap C_i)$ を計算して $P(X \mid C_i)$ を得る

2. デザインパターンでの方法: iteration 1回で実行
   - map 関数で 2種類の値を emit : $P(C_i)$用, $P(X \cap C_i)$ 用
   - $C_i$ をキーとして partition 分け
   - $P(C_i)$を先に計算，次に $P(X \cap C_i)$ を計算，$P(X \mid C_i)$ を得る



# Computing Relative Frequencies

```
map (String key, String value){
  // value is an input record
  for each class in value:
     EmitIntermediate(“_C1_”+class, 1);
        for each attribute in value:
           EmitIntermediate(“_C2_”+class+”:”+attribute, 1);
}
```

map 関数で 2種類の値を emit

$P(C_i)$ を計算後に$P(X \cap C_i)$を計算

```
Private HashMap<Integer> PC = new HashMap<Integer>();
reduce (String key, Iterator values){
  if prefix(key) == _C1_ then PC[getKey(key)] = compute P(Ci);
  else if prefix(key) == _C2_ then
        compute P(X∩Ci)
        P(X | Ci) = P(X∩Ci) / PC[getKey(key)];
}
```

# デザインパターンから学べること

- MapReduce のハッキングテクニック
  - 状況に応じて利用することで処理を高速化可能
- 実装ノウハウを蓄積することは重要
- しかし，パズルみたいで易しくはない
  - GoF のデザインパターンも普通には思いつかない
  - 複雑なノウハウを利用者から隠ぺい化したい

ある種の最適化に関わる研究課題
1. 新たなデザインパターンを見つける
2. システムが自動的に適切なデザインパターンを選択する



# 少し余談: MapReduce って結局何？

以下の要素からなるシステム
- 関数型言語のインタフェース
  - map (f) [r1,...,rn] = [f(r1),...., f(rn)]
  - reduce (⊕) [r1,...,rn] = r1 ⊕ .... ⊕ rn
- 分散環境への適用
  - map:マシン内データ処理，reduce:マシン間データ処理
    - 論理プラン = 物理プラン
    - (key, value) + hash によるシャッフル処理
  - エラー処理などの分散特有の処理を隠ぺい
  - ファイルを介してプロセス間通信 (高速なリカバリ)

# 少し余談: Dryad もあるよね？

- Dryad: マイクロソフトの分散処理システム
- MapReduce の課題を解消している

| | プログラミングスタイル | 実行プラン | プロセス間通信 | データ処理言語 |
|---|---|---|---|---|
| MapReduce | map/reduce | 論理プランと物理プランは同じ | ファイル経由 | Pig Latin, Hive |
| Dryad | DAG による柔軟なワークフロー記述 | 論理プランを最適化し，物理プランを導出（多段 reduce 処理他） | TCP-pipe and in-memory FIFO, ファイル経由 | LINQ |

- しかし，多様な部品を組み合わせるプログラミング，物理プランの性能チューニングが複雑そう
- Hadoop MapReduce のコミュニティ規模の差



# MapReduce の高速化の研究動向

# 例1: 標準偏差の計算

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}.$$

where $\mu$ is the mean value of
$X = x_1, \ldots, x_i, \ldots, x_N = \boldsymbol{ab} = \boldsymbol{a} \cup \boldsymbol{b}$

どうやって MapReduce 上に実装すれば良いか？

1. 直観的な方法: 2回の iteration
   - 1st iteration: $\mu$ = *sum/count* と *N (count)* を計算
   - 2nd iteration: $\mu$ を使って定義式に従い $\sigma$ を計算
2. インクリメンタルな方法: 1回の iteration
   (summation form, 準同型)

**標準偏差** $\sigma_{ab} = \frac{n_a\sigma_a + n_b\sigma_b}{n_a + n_b} + n_a n_b \left(\frac{\mu_b - \mu_a}{n_a + n_b}\right)^2$

**平均値** $\mu_{ab} = \frac{n_a\mu_a + n_b\mu_b}{n_a + n_b}$

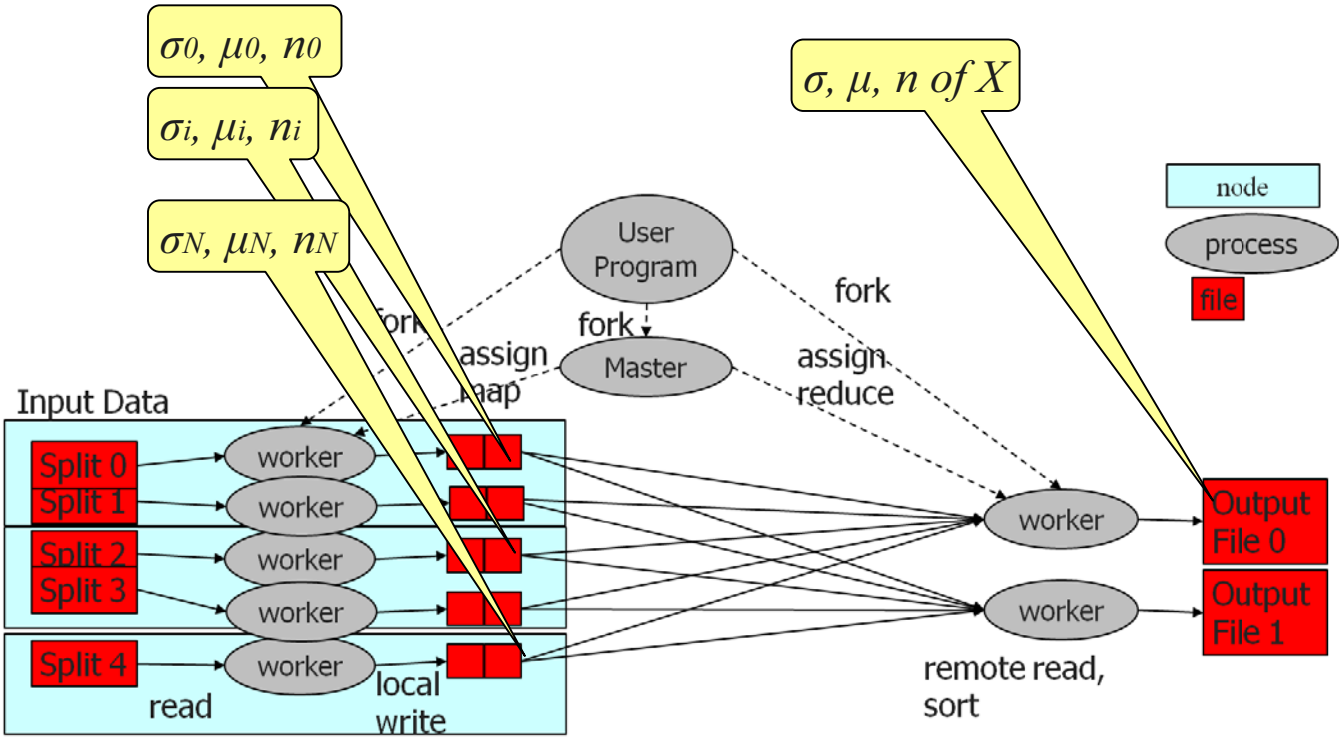
**データ件数** $n_{ab} = n_a + n_b$

Calculating variance and mean with MapReduce (Python)
http://blog.cordiner.net/?p=764



# 例1: 標準偏差の計算



# Summary form の応用

- ジョイン演算
  - インクリメンタルビュー更新 [Gupta et al., IEEE 1995]
    $R \bowtie (S1 \cup S2) = R \bowtie S1 \cup R \bowtie S2$
- 機会学習 [Chu et al, NIPS 2006]
  - 10 種類の代表的な機会学習は (naive bayse, k-means, SVM,…) summation form で表現可能
  - summation form は MapReduce への変換が容易
  - 評価実験の結果コア数に良くスケールする結果が得られた

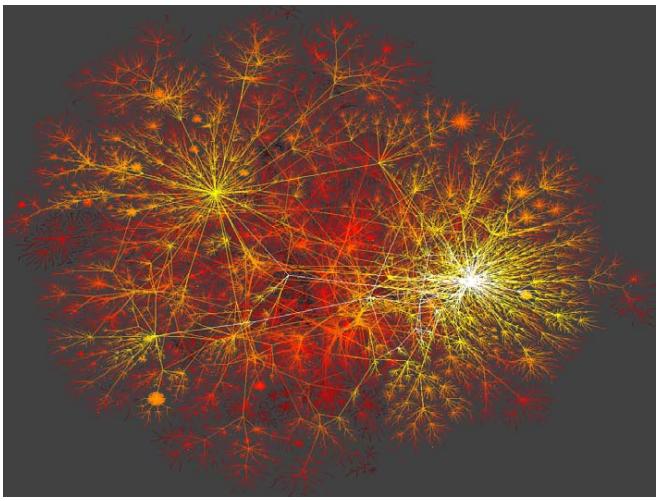


# 例 2: PageRank

page $p$ の PageRank $\boldsymbol{v}$ は利用者によって p がアクセスされる確率.

$$\boldsymbol{v} = (1-c)\boldsymbol{A}\boldsymbol{v} + c\boldsymbol{u}$$

但し，$\boldsymbol{A}$ はwebグラフの隣接行列，$c$ はランダムジャンプの確率，$u$ はランダムなページのベクトル



出展: The architecture of complexity, ASIS Keynote 2006

# 例 2: PageRank

page $p$ の PageRank $v$ は利用者によって p がアクセスされる確率.

$$\boldsymbol{v} = (1-c)\boldsymbol{A}\boldsymbol{v} + c\boldsymbol{u}$$

但し，$A$ はwebグラフの隣接行列，$c$ はランダムジャンプの確率，$u$ はランダムなページのベクトル

$v$ はランダムサーファーモデルによる収束値として得られる

$$\begin{bmatrix} \\ v \\ \\ \end{bmatrix} = (1-c)\begin{bmatrix} \\ A \\ \\ \end{bmatrix}\begin{bmatrix} \\ v \\ \\ \end{bmatrix} + c\begin{bmatrix} \\ u \\ \\ \end{bmatrix}$$



# 例 2: PageRank

どうやって MapReduce 上に実装すれば良いか？

$$\boldsymbol{v} = (1-c)\boldsymbol{A}\boldsymbol{v} + c\boldsymbol{u}$$

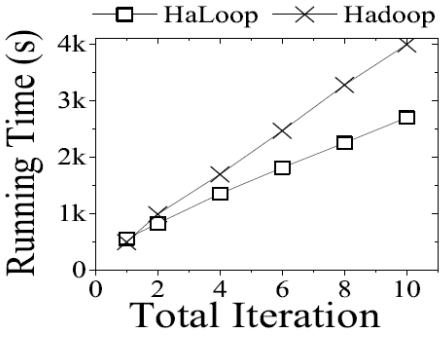
但し，$A$ はwebグラフの隣接行列，$c$ はランダムジャンプの確率，$u$ はランダムなページのベクトル

1. 直観的な方法: 1 iteration → 1 MapReduce job
   - 収束するまで，あるいは一定回数で計算を停止
2. もっと高速に処理するには？
   - iteration 数の削減: 複数 iteration → 1 MR job
     - PEGASUS [Kang et al., ICDM 2009]
       隣接行列 $A$ およびベクトル $v$ をブロック化
   - 複数 job において変化しないデータをキャッシュする
     - HaLoop [Bu et al, PVLDB 2010]

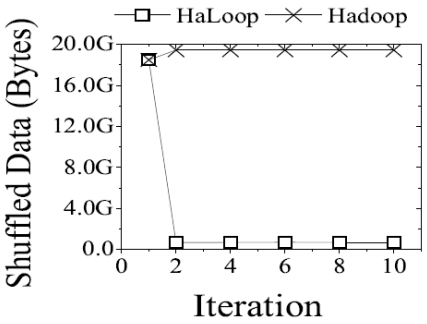
 

# HaLoop [Bu et al, PVLDB 2010]

- キャッシュ+インデックス化による多段 iteration 高速化
  - mapper input cache: HDFS read 量を削減
    k-means: 入力データを全 mapper node にキャッシュ
  - reducer input cache: shuffle 量を削減
    PageRank: web隣接行列の各 partition を，担当 reducer node にキャッシュ + インデックス構築




(a) Overall Performance




(d) Shuffled Bytes

**HaLoop vs. Hadoop (Livejournal Dataset, 50 nodes)**



# MapReduce に関する研究の動向 (2/2)

- 高速化
  5. ジョイン処理:
     - [Dittrich et al., PVLDB 2010]: co-partitioning join tables
     - [Afrati et al., EDBT 2010]: shuffle コスト最適化
  6. 同期処理の回避:
     - MapReduce online [Condie et al., NSDI 2010]
  7. スケジューラ: LATE [Zaharia et al., OSDI 2008]
  8. 最新ハードウェア: Mars [He et al., PACT 2008]: GPUs
- 分散処理モデル
  - Pregel [Malewicz, SIGMOD 2010]: グラフモデル
  - Dryad [Isard et al., EuroSys 2007]: データフローモデル

# 研究動向までのまとめ

- MapReduce
  - スケーラブル (1万台, ペタバイトスケール)
  - シンプルなAPI (map/reduce)
- 研究が盛んに行われている
  - データベース系会議 (SIGMOD, VLDB,…)
  - OS系会議 (OSDI, SOSP, NSDI…)
  - NL/マイニング系会議 (NIPS, ICDM…)

SOCC



# MapReduce 高速化の取り組み

## 1. PJoin:
## Efficient join processing with MapReduce for OLAP applications

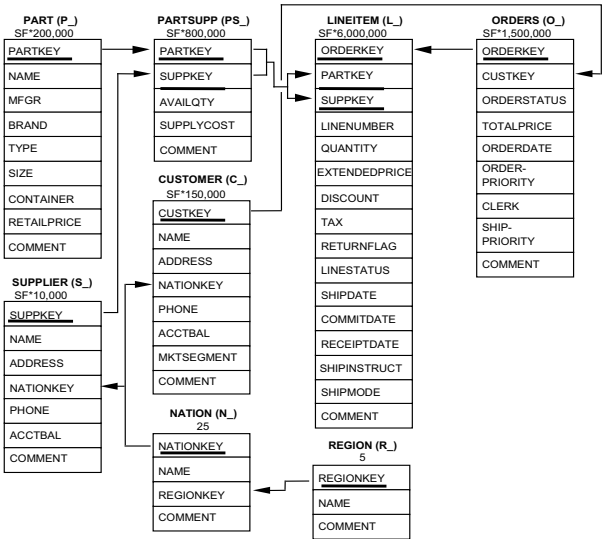
# 技術の概要

- PJoin (pre-partition-based join)
  - **[到達点]** 多次元データ分析(OLAP)処理において，シャッフル量を 1/3 に削減することで処理時間を，従来技術より 42.8% 高速化
  - **[戦略]** 複数の分析処理において共通するシャッフル処理を前もって実行(事前処理)することで，分析処理時のコストを削減



# 背景: 多次元データ分析(OLAP)とは？

- 統計的な分析処理の典型的な手法
  - 履歴・トランザクションデータを格納する factテーブル(大)
  - fact テーブルを多角的な次元で集約演算を実施するための，複数の dimension テーブル




スター型スキーマ (TPC-H)

```
SELECT   nation,
         o_year,
         sum(amount) as sum_profit
FROM  (SELECT    n_name as nation,
                 extract(year from o_orderdate) as o_year,
                 l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity as amount
         FROM    part, supplier, lineitem, partsupp, orders, nation
         WHERE   s_suppkey = l_suppkey
                 and ps_suppkey = l_suppkey
                 and ps_partkey = l_partkey
                 and p_partkey = l_partkey
                 and o_orderkey = l_orderkey
                 and s_nationkey = n_nationkey
                 and p_name like '%[COLOR]%'
      ) as profit
GROUP BY nation, o_year
ORDER BY nation, o_year desc;
```

集約演算

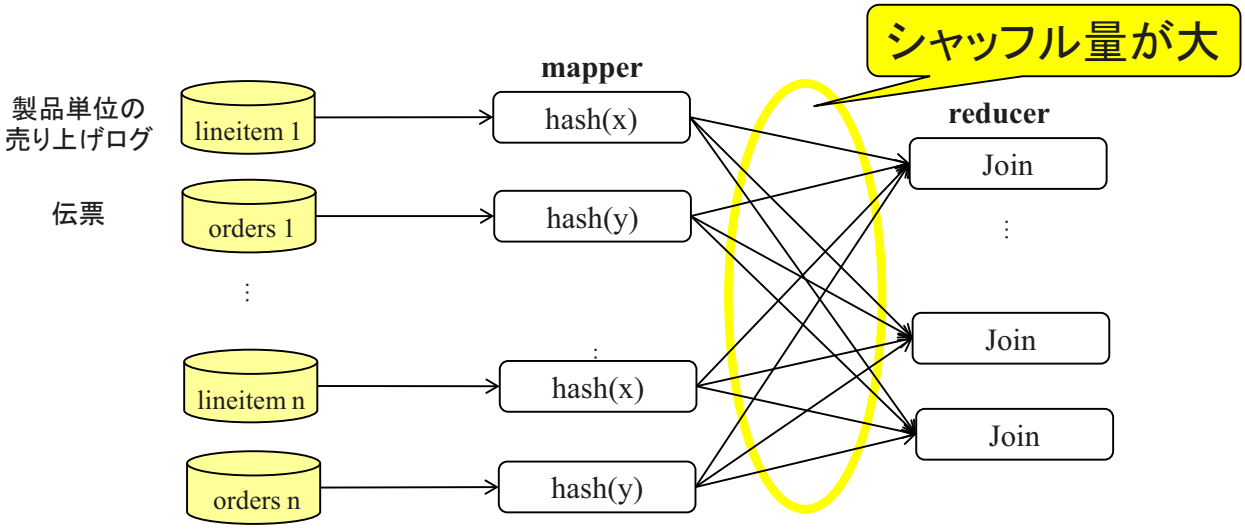
多テーブル結合処理

グループ化

クエリ 9 (TPC-H)

# 背景: 従来の MapReduce 結合処理の課題

- ハッシュ結合による方法:
  - シャッフル量に起因する通信コスト・IOコストが大
- 通信コスト・IOコストの削減が最重要課題



# PJoin の概要

**[方針]** テーブル結合処理時のシャッフル量を削減する
**[前提]** OLAP分析では，更新よりも参照処理の性能が重要
1対多の関係でテーブル結合処理する
**[戦略]**

- 複数の分析処理において共通的なシャッフル処理を事前処理することで，分析処理時のコストを削減
  - 結合条件(主キー)によるテーブルの**事前シャッフル**
  - 準結合の活用 + 中間データの**事前生成**
- ③ 複数MapReduce 間でシャッフル量を削減する結合計画

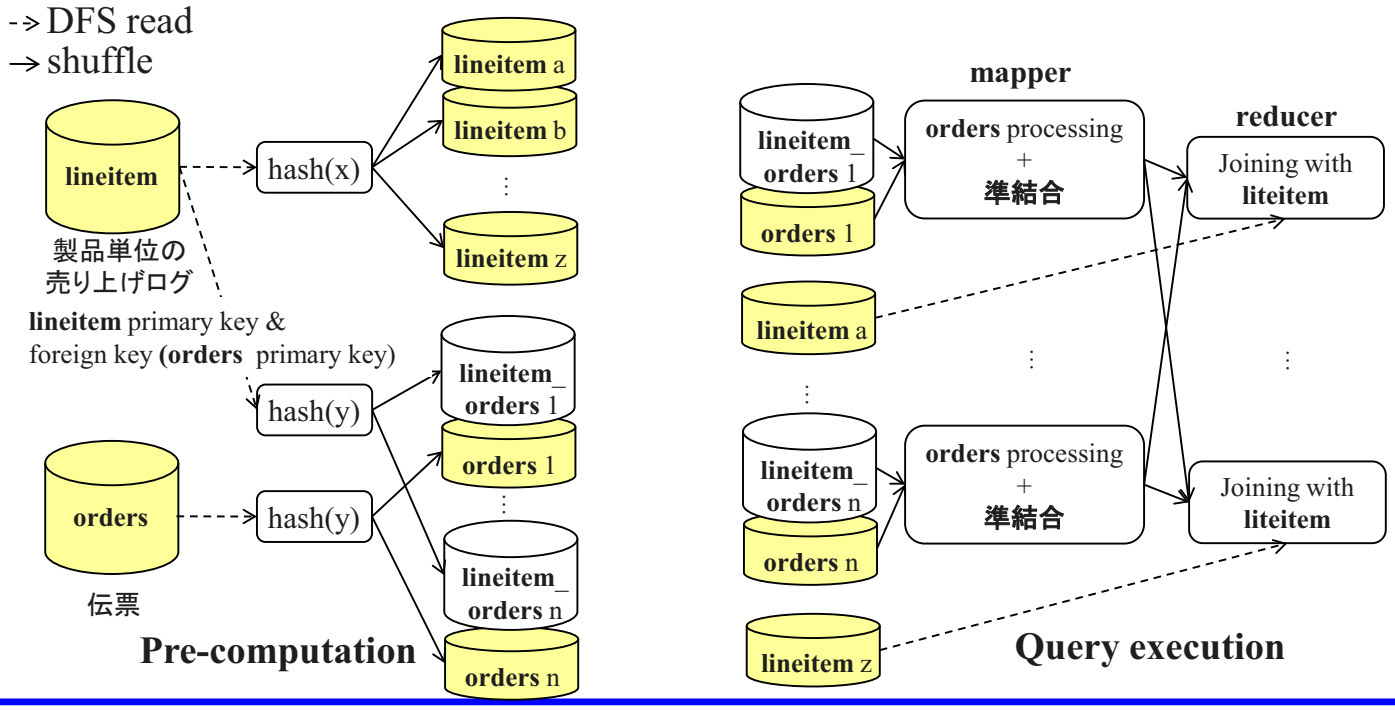
**[効果]**

- 分析処理時のシャッフル量を削減
- ① Nテーブル結合演算によりMapReduce job数を削減
- ②

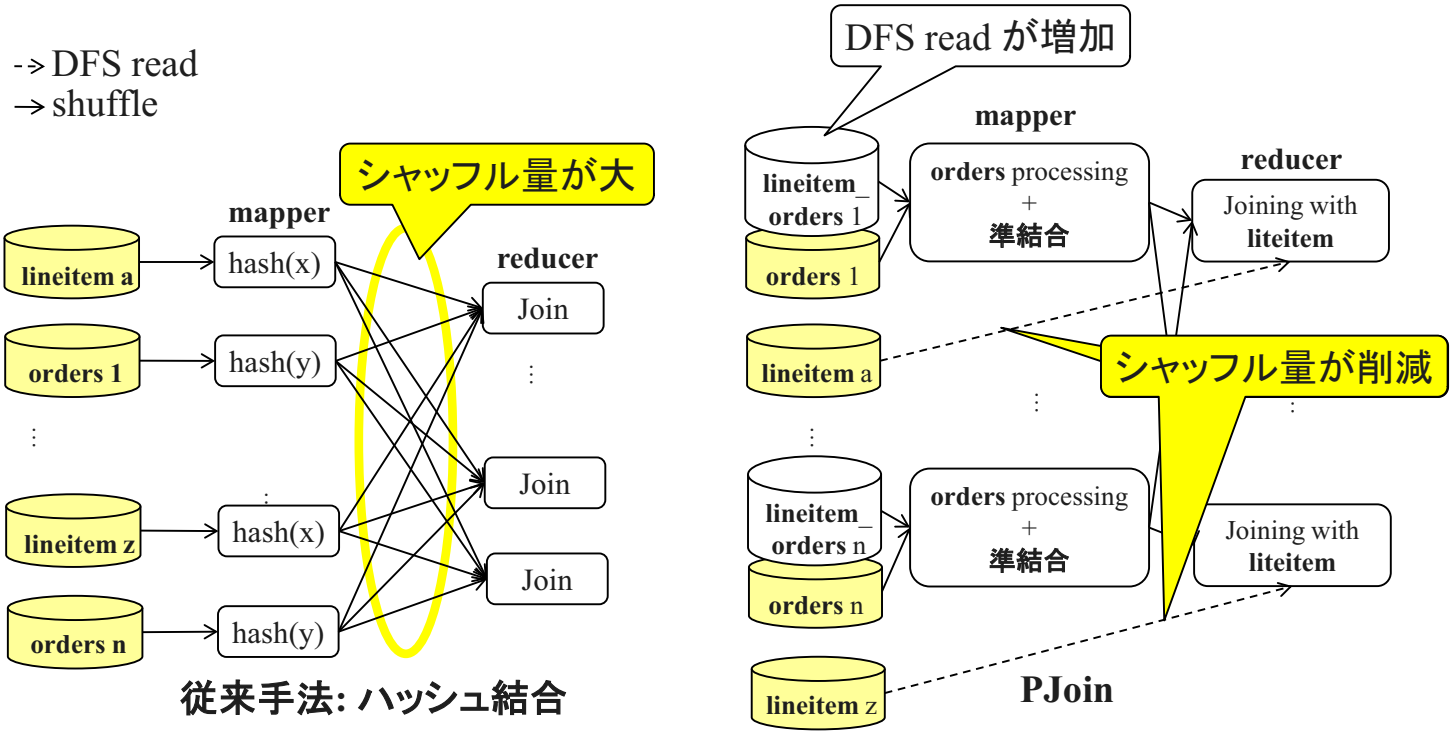
# PJoin の特徴①: シャッフル量削減

- テーブルの事前シャッフル実行，準結合中間データの事前生成
- mapper で準結合処理後に，reducer で残処理を実行






# 従来手法と PJoin の比較



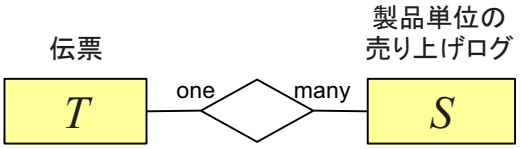
# PJoin の特徴②: Nテーブル結合

- PJoin

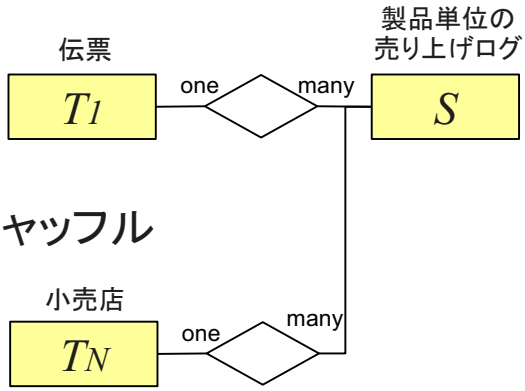
$$T \bowtie S = (T \ltimes S) \bowtie S$$
$$= (T \bowtie F_S^T) \bowtie S$$

T の主キーで結合処理

mapper（T の主キーで結合処理）　reducer (S の主キーで結合処理)




- スター型スキーマの場合
  - 複数 mapper を実行
  - 全 mapper 結果は S の主キーでシャッフル
  - reducer は1つで処理可能

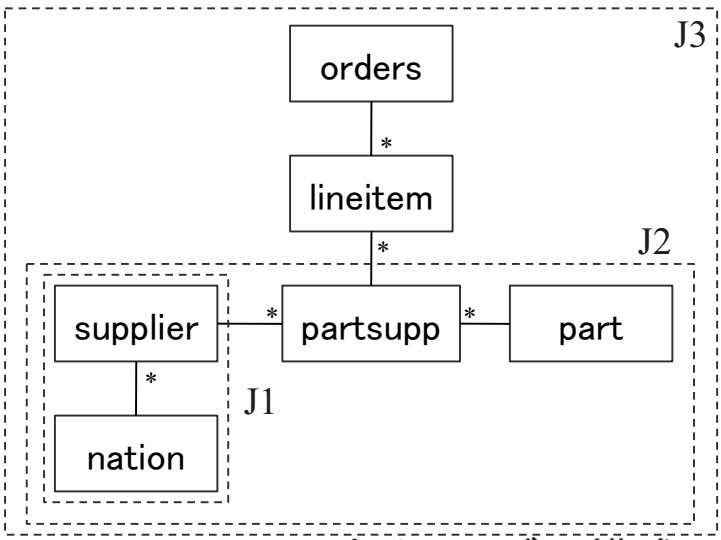



Nテーブル結合が 1つの
MapReduce job で実現可能



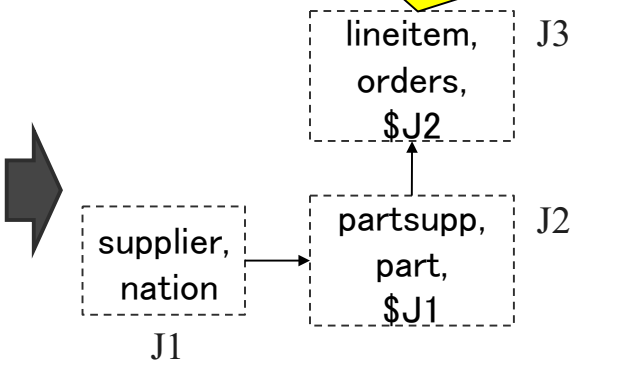
# PJoin の特徴③: スター型クエリ結合計画

- 多次元データ分析のスター型スキーマ
  - dimension テーブルから処理を開始し，fact テーブルを最後に実行することで，中間データを削減する




Q9 でアクセスするテーブル構成

fact テーブルの結合を最後に実行




Q9 結合計画

# 評価実験

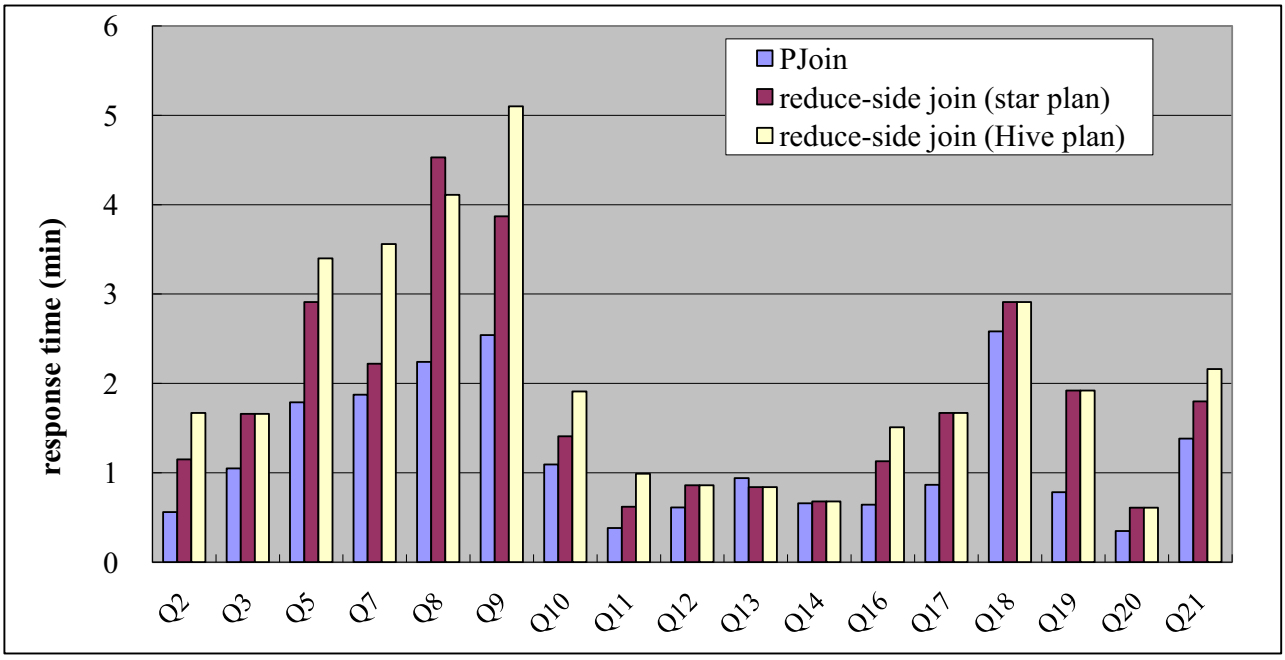
- **評価環境**
  Linux (CentOS5.4, 2.8GHz, 主記憶 8GB) × 50台
  Java1.6, Hadoop 0.19.2
- **ベンチマーク: TPC-H ベンチマーク**
  **[データ]** 104GB, 207GB, 311GB
  準結合中間データ: 83GB, 167GB, 250GB
  **[分析クエリ]** TPC-Hにおける結合演算を利用する 17クエリ

- **評価観点: PJoin と従来の Join 手法の比較**
  - **全体**: PJoin と，従来の2つの Join 手法の応答性能比較
    - PJoin vs reduce-side join (スター型結合計画, Hive 計画)
  - **詳細**
    - PJoin による シャッフル量，HDFS read/write 量の影響
    - スケール性評価

# 応答時間 (104GB, 50台)

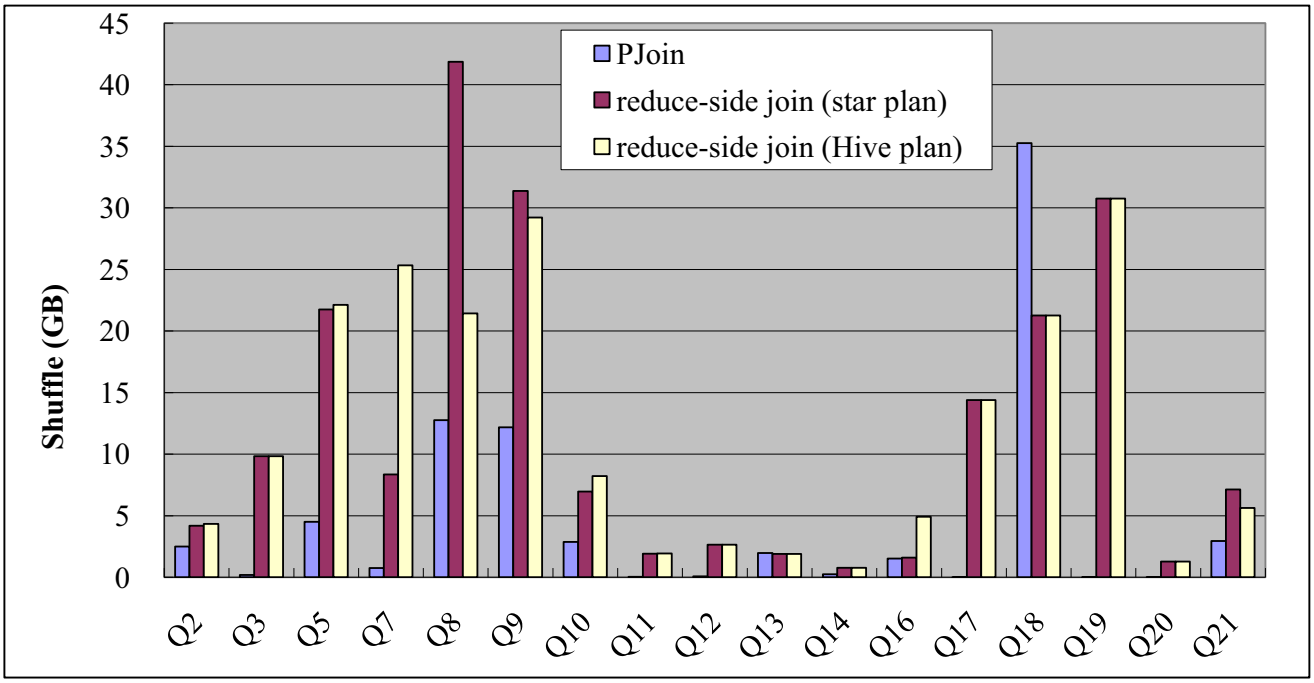
- 従来手法より33.4% (star plan比), 42.8% (Hive plan比)
- 改善の効果: MapReduce job 数削減，HDFS+シャッフル量削減
  性能 = {(HDFS read) + (HDFS write) + 2 × (shuffle)} × 係数



# シャッフル量 (104GB, 50台)

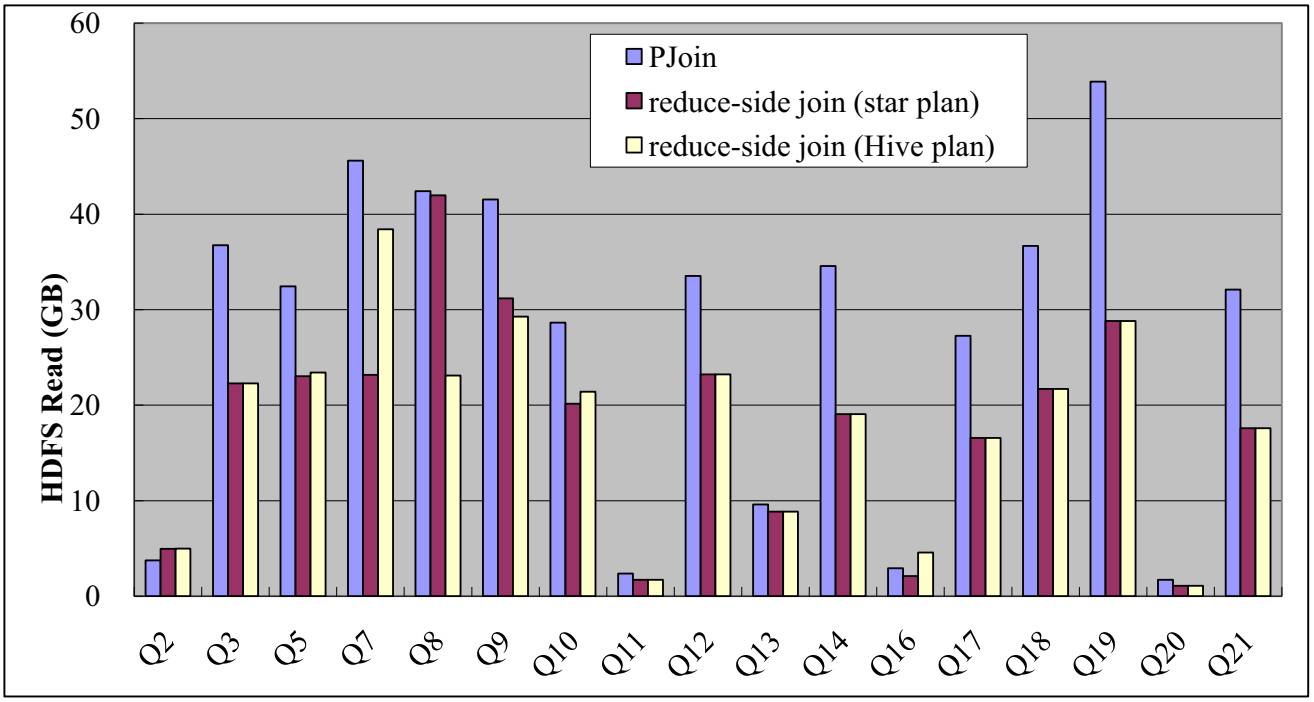
- 従来手法より62.6% (star plan), 62.2% (Hive plan)改善
- Q18 は悪化: WHERE条件の選択効果の低下が原因



# HDFS read量 (104GB, 50台)

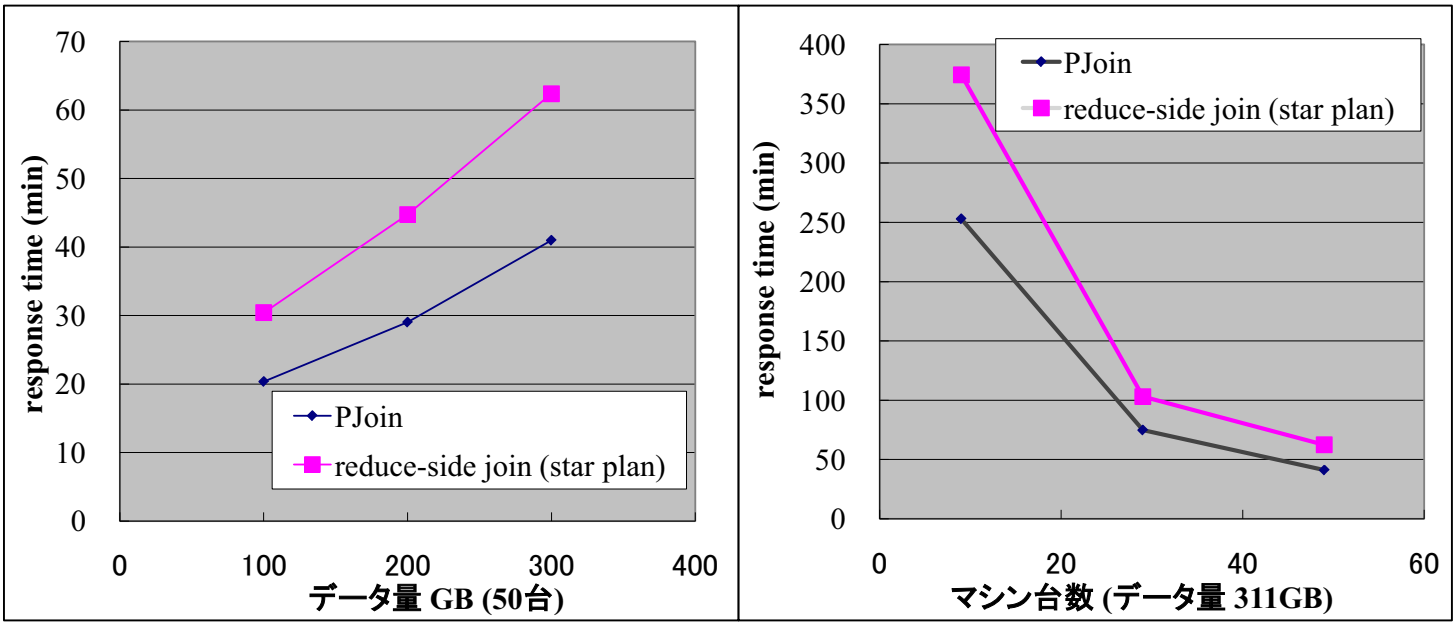
- 従来手法より51.4% (star plan), 52.2 % (Hive plan)増加
- PJoin では準結合中間データを参照するため



# スケール性

- データ量: PJoin の方がスケール性が若干優れる
- マシン数: PJoinも従来手法も台数効果が出ている



# PJoin まとめ

**特徴:**

- シャッフル処理の事前実行(pre-partitioning)，準結合中間データの事前生成
- 準結合を mapper で実行，残りの結合処理を reducer で実行

**効果:**

- TPC-H において 30-40%応答性能を改善
- シャッフル量は1/3 に削減

**今後の課題:**

- 最適プラン選択
- MapReduce の足回りの課題 (HDFS read/write)

back

# 研究の方向性



# 研究の方向性

- MapReduce のメリット
  - 低コストで簡単に(大規模な)分散処理が可能
  - Hadoop コミュニティに参加できる
  - Amazon を使えば大規模環境も利用しやすい
- クラウド系研究の難しさ
  - Hadoop は1000台規模ぐらいまではスケールする
  - 大規模なほどスケールメリットが生きてくる
  - 大規模な環境での技術的な課題に手が届かない
    - The Next Generation of Apache Hadoop MapReduce

# 研究の方向性

- 研究の方向性
  - MRの改善の余地がありDB技術の適用は続く
    - 取り組みやすいが大きい貢献は難しい
  - MapReduce 独自の観点
    - shuffle 量の削減，多段 job 処理
    - 投機実行，障害回復
  - 分散処理における生産性向上
    - ドメイン特化型の分散モデル (Pregel, Parcolator)
    - SQLや関数型言語の最適化技術を活用
      - 宣言的言語から実行プランを導出
      - 関数型言語: 組化，融合



# おわりに

- HadoopのOSSをベースに取り組めるメリット
  - コミュニティへの貢献
- 多くの利用者は Hadoopが万能だと考えている
  - PB規模のデータはマシンが多くても容易ではない
- NTT 研究所での取り組み
  - R&D クラウド 1500台環境
  - 応用
    - 大規模 webデータ, クエリログ，アクセスログ
    - トラフィックデータ解析
    - 音楽療法データ・マイニング

興味のある方，是非議論しましょう！

# 文献


- **[Abouzeid et al., PVLDB 2009]** HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads, A. Abouzeid et al., In PVLDB 2(1), 2009
- **[Afrati et al., EDBT 2010]** Optimizing joins in a map-reduce environment, F. N. Afrati, J. D. Ullman, In EDBT 2010
- **[Agrawal et al., VLDB 2010]** Big Data and Cloud Computing: New Wine or just New Bottles?, D. Agrawal, S. Das, A. Abbadi, In VLDB tutorial 2010
- **[Blanas et al., SIGMOD 2010]** A Comparison of Join Algorithms for Log Processing in MapReduce, S. Blanas, J.M. Patel, V.Ercegovac, J. Rao, In SIGMOD 2010
- **[Bu et al., PVLDB 2010]** HaLoop: Efficient Iterative Data Processing on Large Clusters, S. Blanas, J.M. Patel, V.Ercegovac, J. Rao, In SIGMOD 2010
- **[Chen, PVLDB 2010]** Cheetah: A High Performance, Custom Data Warehouse on Top of MapReduce, S. Chen, In PVLDB 3(2), 2010
- **[Chu et al., NIPS 2006]** Map-reduce for machine learning on multicore, C. Chu, S. Kim, Y. Lin, Y. Yu, G. Bradski, A. Ng, K. Olukotun, In NIPS 2006
- **[Condie et al., NSDI 2010]** MapReduce Online**,** T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, In NSDI2010
- **[Dean et al., ODSI 2004]** MapReduce: Simplified Data Processing on Large Clusters, J. Dean, S. Ghemawat, In OSDI 2004
- **[Dean et al., CACM 2008]** MapReduce: Simplified Data Processing on Large Clusters, J. Dean, S. Ghemawat, In CACM Jan 2008
- **[Dean et al., CACM 2010]** MapReduce: a flexible data processing tool, J. Dean, S. Ghemawat, In CACM Jan 2010
- **[DeWitt, 2009]** Mapreduce: A major step backwards, D. DeWitt, M. Stonebraker, The Database column 2009

# 文献


- **[Dittrich et al., PVLDB 2010]** Hadoop++: Making a Yellow Elephant Run Like a Cheetah (Without It Even Noticing), J. Dittrich, J. Quiane-Ruiz, A. Jindal, Y. Kargin, V. Setty. J. Schad, In PVLDB 3(1), 2010
- **[Gupta et al., IEEE 1995]** Maintenance of materialized views: Problems, techniques and applications, UA. Gupta, I.S. Mumick, IEEE Data Eng. 18(2) 1995
- **[Isard et al., EuroSys 2007]** Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks, M. Budiu, Y. Yu, A. Birrell, D.Fetterly, In EuroSys 2007
- **[He et al., PACT 2008]** Mars: A MapReduce framework on graphics processors, B. He, W. Fang, Q. Luo, N.K. Govindaraju, T. Wang, In PACT 2008
- **[Kang et al., ICDM 2009]** PEGASUS: A Peta-Scale Graph Mining System Implementation and Observations, U. Kang, C. Tsourakakis, C. Faloutsos, In ICDM 2009
- **[Lang et al., PVLDB 2010]** Energy Management for MapReduce Clusters, W. Lang, J. M. Patel, In VLDB 3(1), 2010
- **[Lin et al., 2010]** Data intensive text processing with MapReduce, J. Lin and C. Dyer, Morgan & Claypool 2010
- **[Malewicz, SIGMOD 2010]** Pregel: a system for large-scale graph processing, G. Malewicz, M. Austern, A. Bik, J. Dehnert, I. Horn, N. Leiser, G. Czajkowski, In SIGMOD 2010
- **[Nykiel et al., PVLDB 2010]** MRShare: Sharing across multiple queries in MapReduce, T. Nykiel, M. Potamias, C. Mishra, G. Kollios, N. Kodas, In VLDB 3(1), 2010
- **[Panda et al., PVLDB 2009]** PLANET: Massively parallel learning of tree ensembles with MapReduce, B. Panda, J.S. Herbach, S. Basu, R.J. Bayardo, In PVLDB 2(2), 2009
- **[Zaharia et al. OSDI 2008]** Improving MapReduce performance in heterogeneous environments, M. Zaharia, A. Konwinski, A. Joseph, In OSDI 2008