クラウドプログラミング環境

プログラミング環境特論

2010年2月4日

建部修見

クラウドコンピューティング

- 必要なCPU資源，ストレージ資源をオンデマンドにネットワーク経由で利用
  - Availability, throughput, reliability
  - Manageability
- 手元にハードウェア，ソフトウェアは必要なし
  - 調達，保守，更新などから解放される

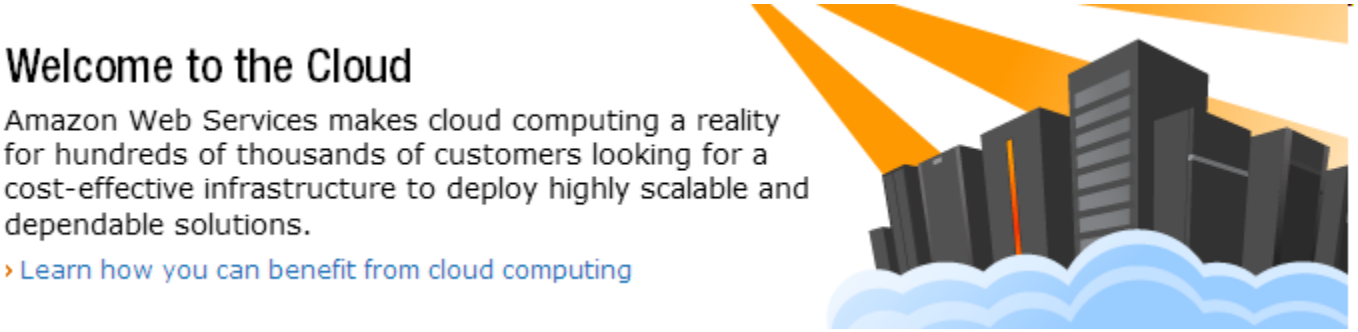
- MapReduceによる大規模分散データ処理
  - Loosely coupled data intensive computing
  - 第二のMPIとなるか？

Salesforce.com (1999)

- ネットワーク経由でCRM（顧客管理）サービスの提供
  - ソフトウェア，ハードウェアの導入不要
  - Webインターフェース
    - Outlook, Office, Notes連携，モバイル，オフライン
  - カスタマイズ
    - マウス操作，Apexコード
  - マルチテナント

Amazon Web Services (2002)

- On-demand elastic infrastructure managed by web services
  - Elastic Compute Cloud (EC2)
    - Web service that provides resizable compute capacity
  - Simple Storage Service (S3)
    - Simple web service I/F to store and retrieve data
  - Elastic Block Store (EBS)
    - Block level storage used by EC2 in the same AZ
    - Automatically replicate within the same AZ
    - Point-in-time snapshots can be persisted to S3
- Region and Availability Zone



Amazon CloudFront (2008)

- Web Service for Content Delivery
  - Low latency, high data transfer, no commitments
- Cache copies close to end users
  - US, Europe, Japan, Hong Kong
- No need to maintain web servers
- By default, support peak speeds of 1 Gbps, and peak rates of 1,000 req/sec
- Designed for delivery of “popular” objects
  - Cache poplar objects and remove less poplar objects

Introducing Amazon CloudFront

Distribute your popular content from Amazon S3 around the globe with a single API call. High-performance content delivery is now self-service and pay-as-you-go.

› Learn more

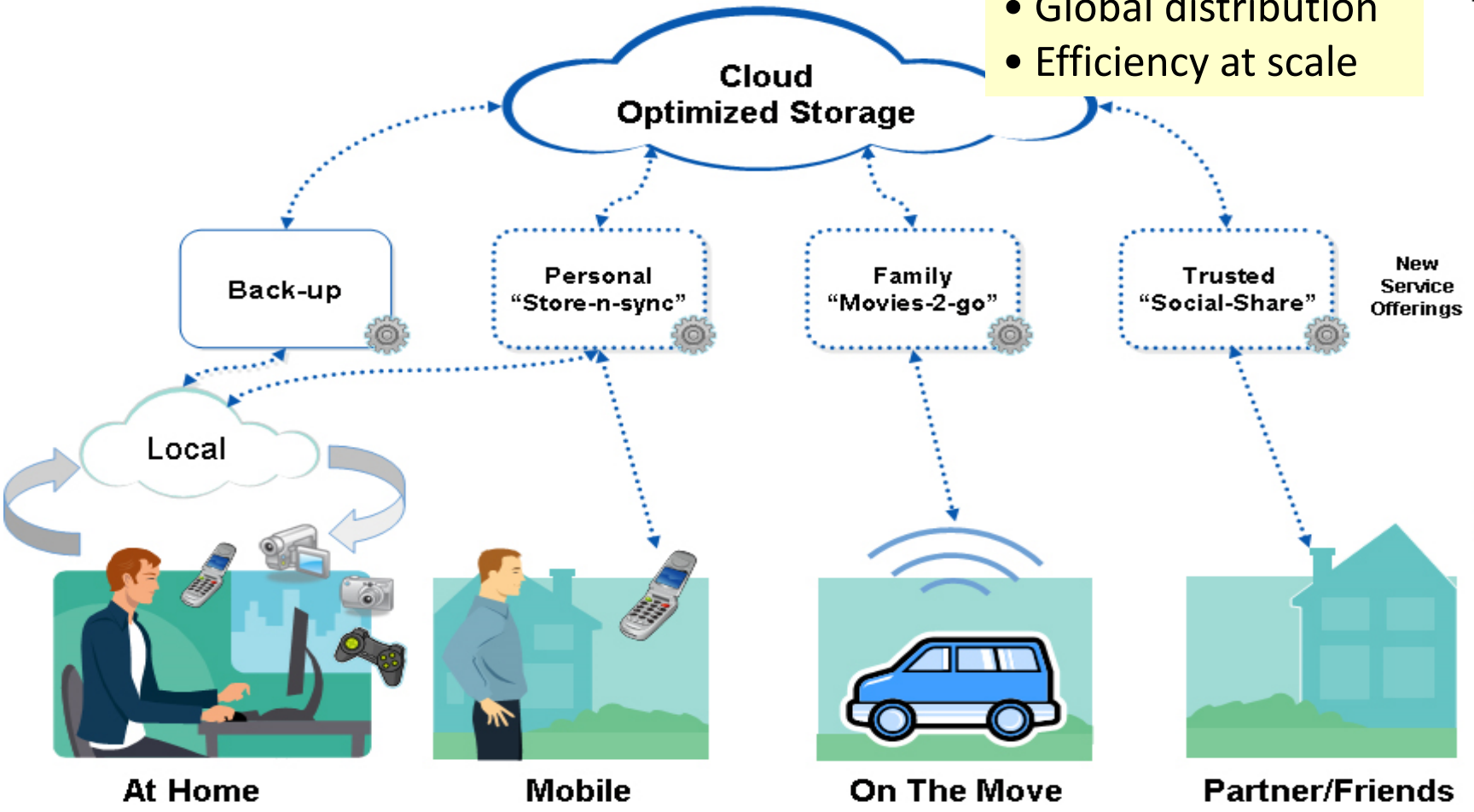
Google App Engine (2008)

- GoogleのインフラでWebアプリケーションを実行
  - Python SDK
- Datastore - Distributed data storage service
  - Data objects have a set of properties
  - Objects are retrieved by properties

- 大規模データ処理用ではない

EMC Cloud Optimized Storage (2008)

- Massive scalability
- Global distribution
- Efficiency at scale

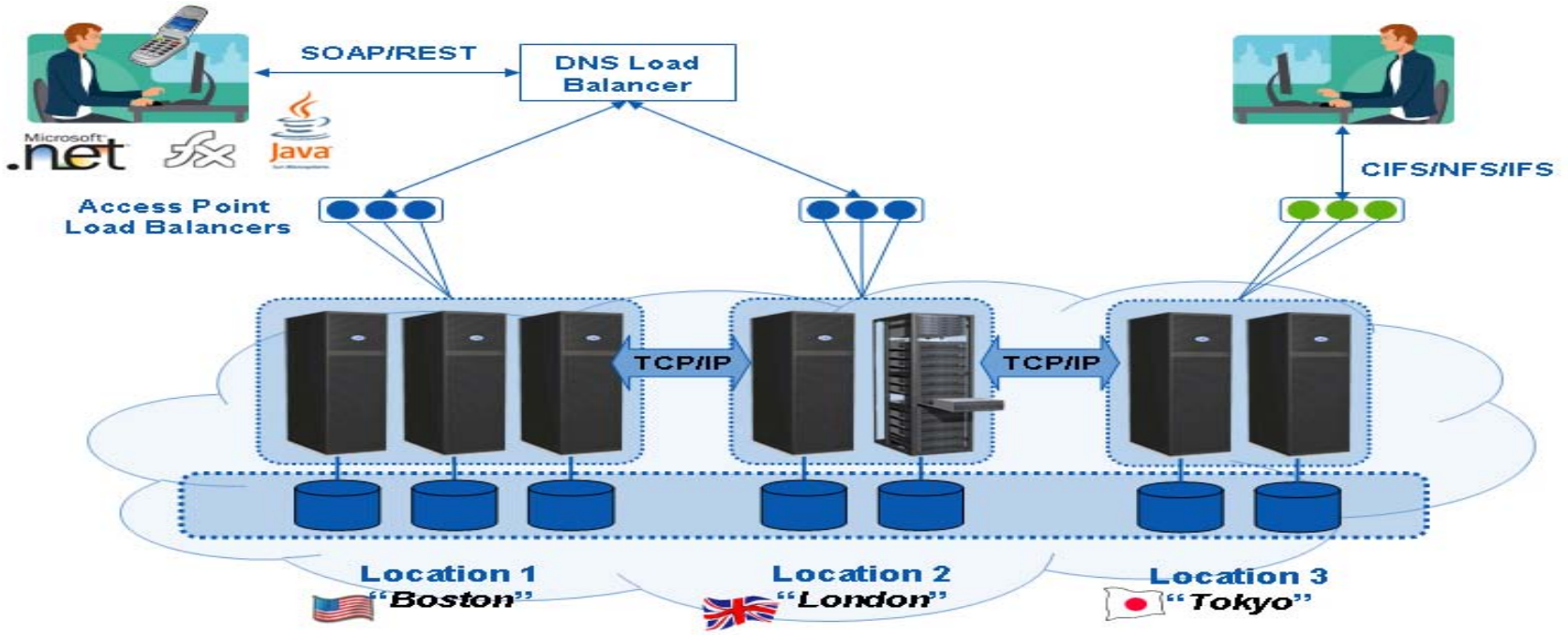


http://www.emc.com/images/about/news/atmos/ideas.jpg

EMC Atmos (2008)

Features

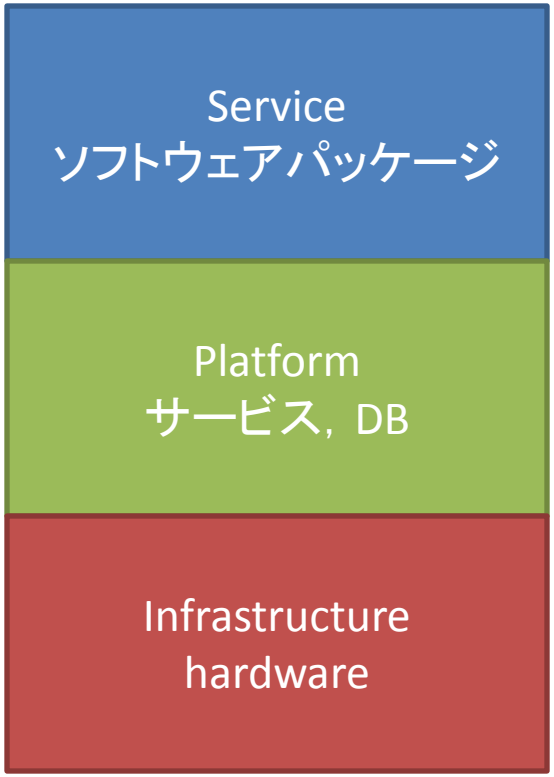
- Object metadata
- Policy-based information management
- Built-in replication, versioning, compression, deduplication, spindown
- Web Service APIs and legacy protocols
- Unified namespace



http://www.emc.com/about/news/press/2008/20081110-01.htm

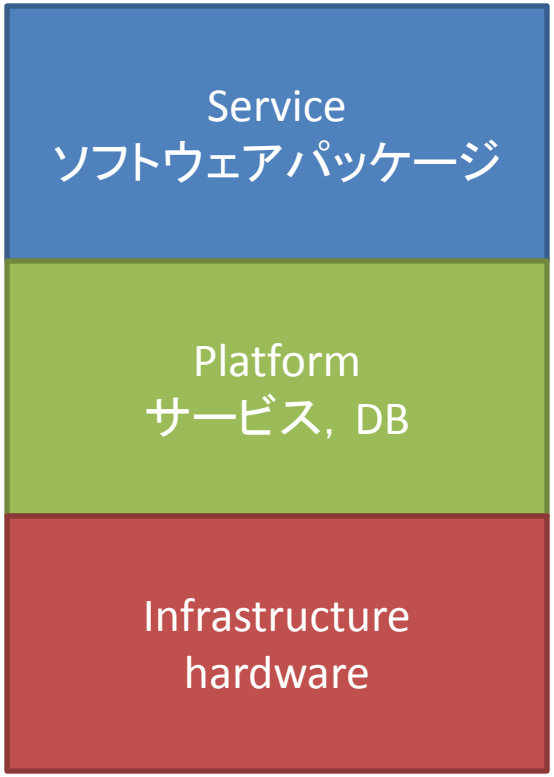
クラウドの分類

- SaaS (Software as a Service)
  - Google Apps (Gmail, …), CRM
  - Microsoft Online Services
- PaaS (Platform as a Service)
  - Webアプリ開発
    - Force.com, Google App Engine
    - Windows Azure
- IaaS (Infrastructure as a Service)
  - Amazon EC2, S3

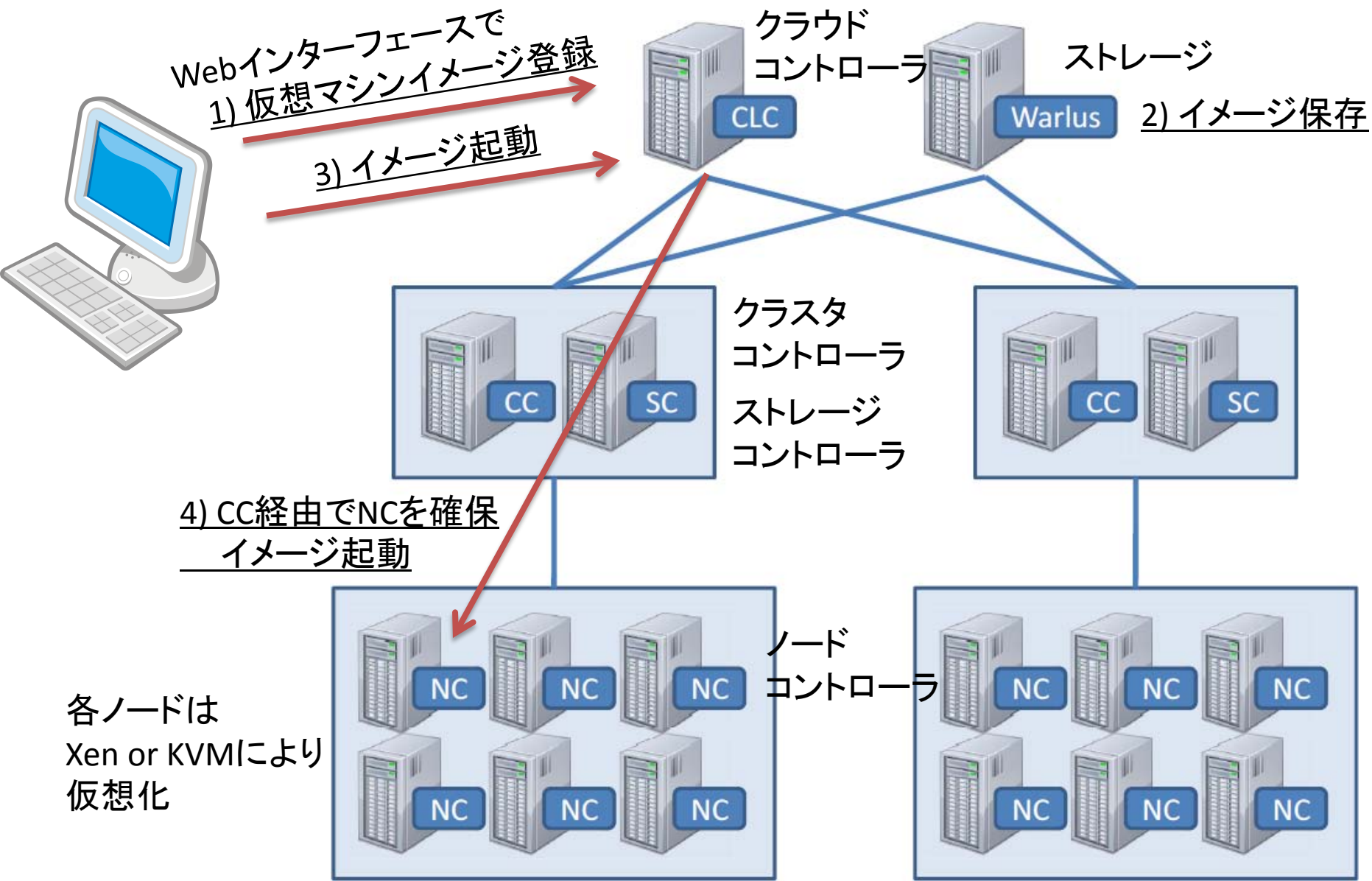


クラウドの利用技術

- SaaS (Software as a Service)
  - Web 2.0
- PaaS (Platform as a Service)
  - Web API
  - Web Service
    - XML, WSDL, SOAP/REST
- IaaS (Infrastructure as a Service)
  - 仮想マシン（Xen, KVM）
  - ディスク，ストレージ，ネットワーク仮想化



IaaSの例：Eucalyptus (2009 Nurmi)



Eucalyptus (2)

- 各ノードではXenカーネルが起動し，仮想マシンイメージを起動（EC2相当）
- ストレージコントローラはブロックデバイスを仮想化（EBS相当）
- Warlusはストレージを仮想化（S3相当）
- クラウドコントローラはWebインターフェースにより管理する
  - 仮想イメージの登録
  - ブロックデバイスの確保
  - ノードの確保，仮想イメージの起動，ブロックデバイスのマウント
  - ストレージへのアクセス

クラウドにおけるストレージシステム

- Availability, reliability
- Amazon Web Services
  - S3, EBS
  - EBSによるHDFSの実現
  - 汎用的に利用できるため理論的にはなんでもできそう
  - Availability ZoneやRegionを超えることは難しい
- Google App Engine
  - GFS, BigTableを利用
  - MapReduceは？？？
  - 地理的分散は？？？

クラウドのまとめ

- クラウドコンピューティングにおける資源
  - 安価，いつでも利用可能，高信頼，高性能
  - 管理しやすさ
- 仮想化技術，Webインターフェースにより実現
- 手元にハードウェア，ソフトウェアは必要なし
  - 調達，保守，更新などから解放される
- 資源がより必要であればさらにクラウドを利用

MapReduce (2004)

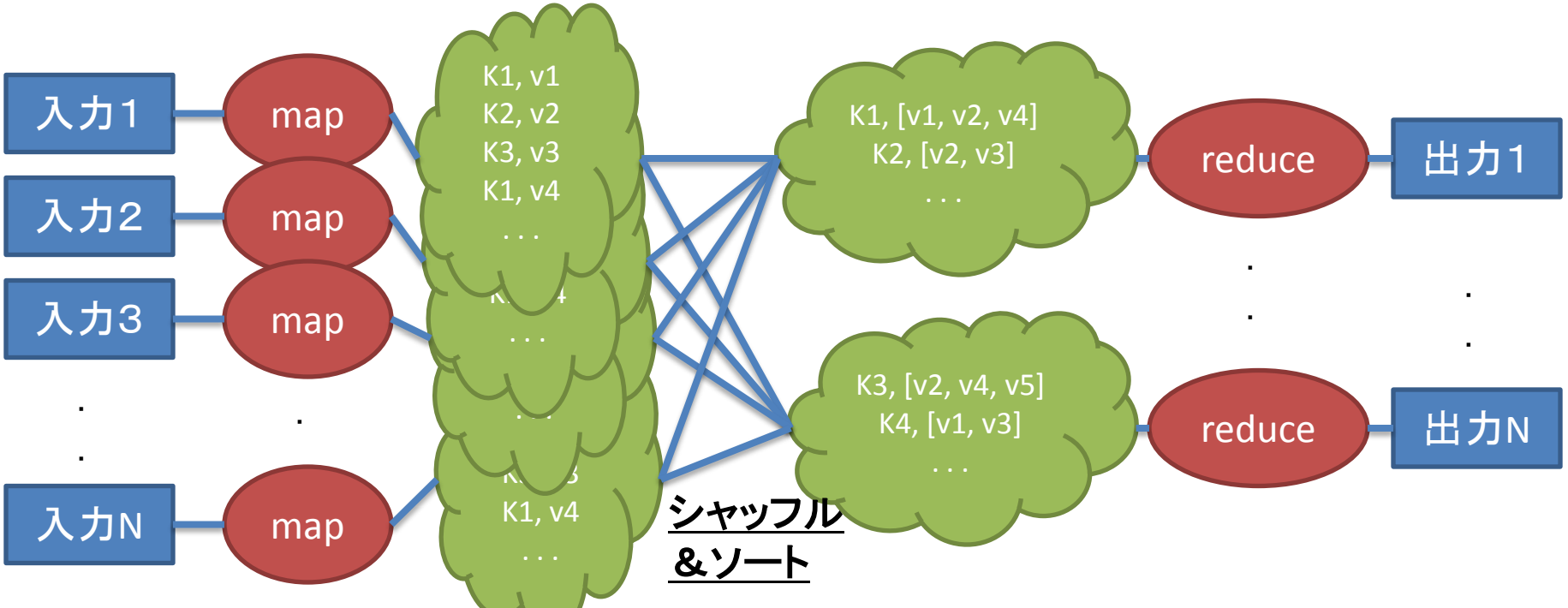
- **大規模クラスタ上でのデータ処理**のためのプログラミングモデル，実行時環境
- ユーザは**map**関数と**reduce**関数を指定
- 実行時システムが以下を行う
  - 自動的に並列化
  - マシン障害処理
  - ディスク，ネットワークの効率的な利用のためのジョブスケジューリング

背景

- Googleでは
  - 大量のクロールしたドキュメント，Webリクエストログなどから
    - 転置インデックス
    - Webドキュメントの様々なグラフ表現
    - ホストごとのクロールしたページ数
    - 1日で最も頻出したクエリのセット
  - を，数百～数千のマシンで分散処理を行っていた
- 並列計算の手法，データ分散，エラー処理のためのコードが大量に必要
- 元々の演算処理のコードがわかりにくい

新しい抽象化（1）

- 実行したい演算の記述する
- 並列処理，耐障害性処理，データ分散，負荷分散など，面倒な処理は実行時ライブラリで隠蔽
- 計算の大部分は以下の処理

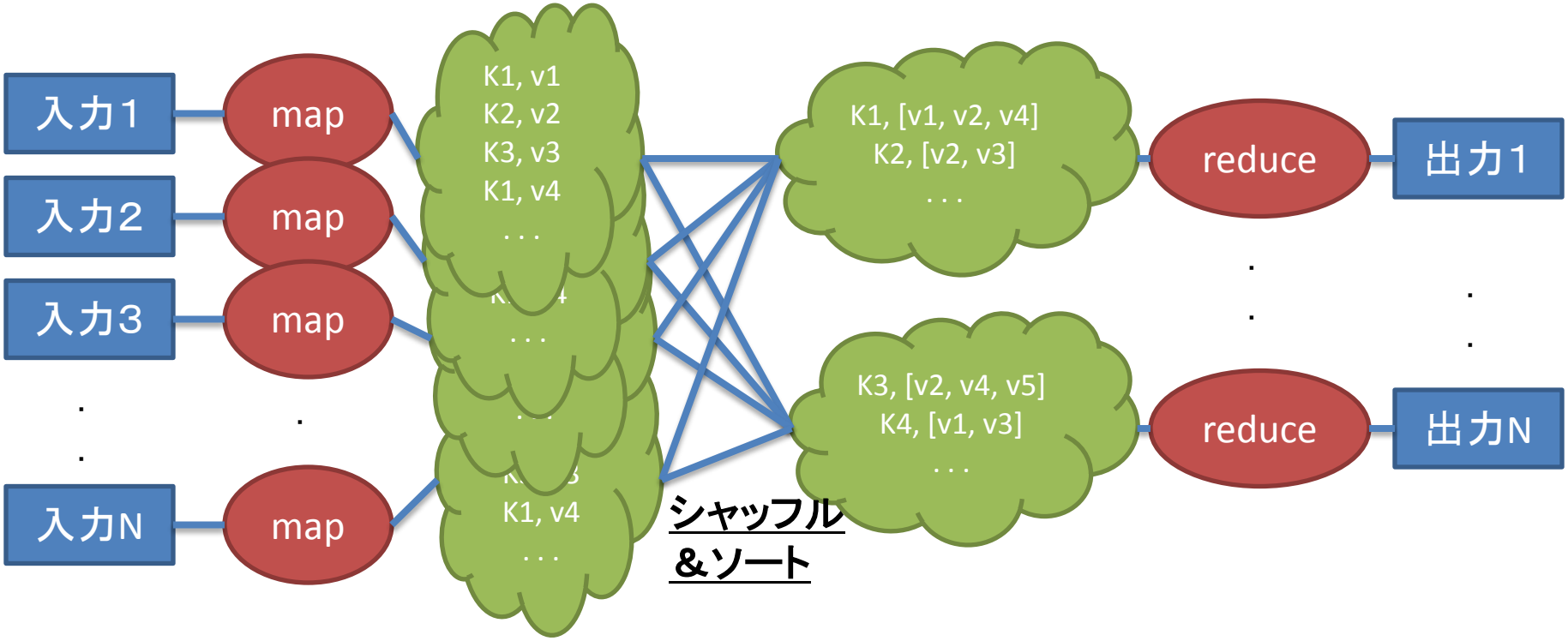


新しい抽象化（2）

- ユーザ指定のmap，reduce操作による関数型モデルにより
  - 大規模計算の並列化が容易に
  - 耐故障性実現のための再実行が可能に

- 単純で強力なインターフェース
- 自動並列化，自動分散化により大規模クラスタでの高性能実行が可能に

プログラミングモデル



- 入力，出力，中間データはすべてkey/valueペアの集合
- Mapとreduceはユーザが指定
- Mapの結果はキーごとにまとめられ，reduceに渡される

例：単語の出現頻度

- Mapではkey：単語とvalue：1を出力
  - (doc, “this is a pen”)→(this, 1), (is, 1), (a, 1), (pen, 1)
- Reduceでは，key（単語）ごとに1の列が入力となるため，総和をとる
  - (this, [1 1 1 1]), (is, [1 1 1]), . . .→(this, 4), (is, 3), . . .

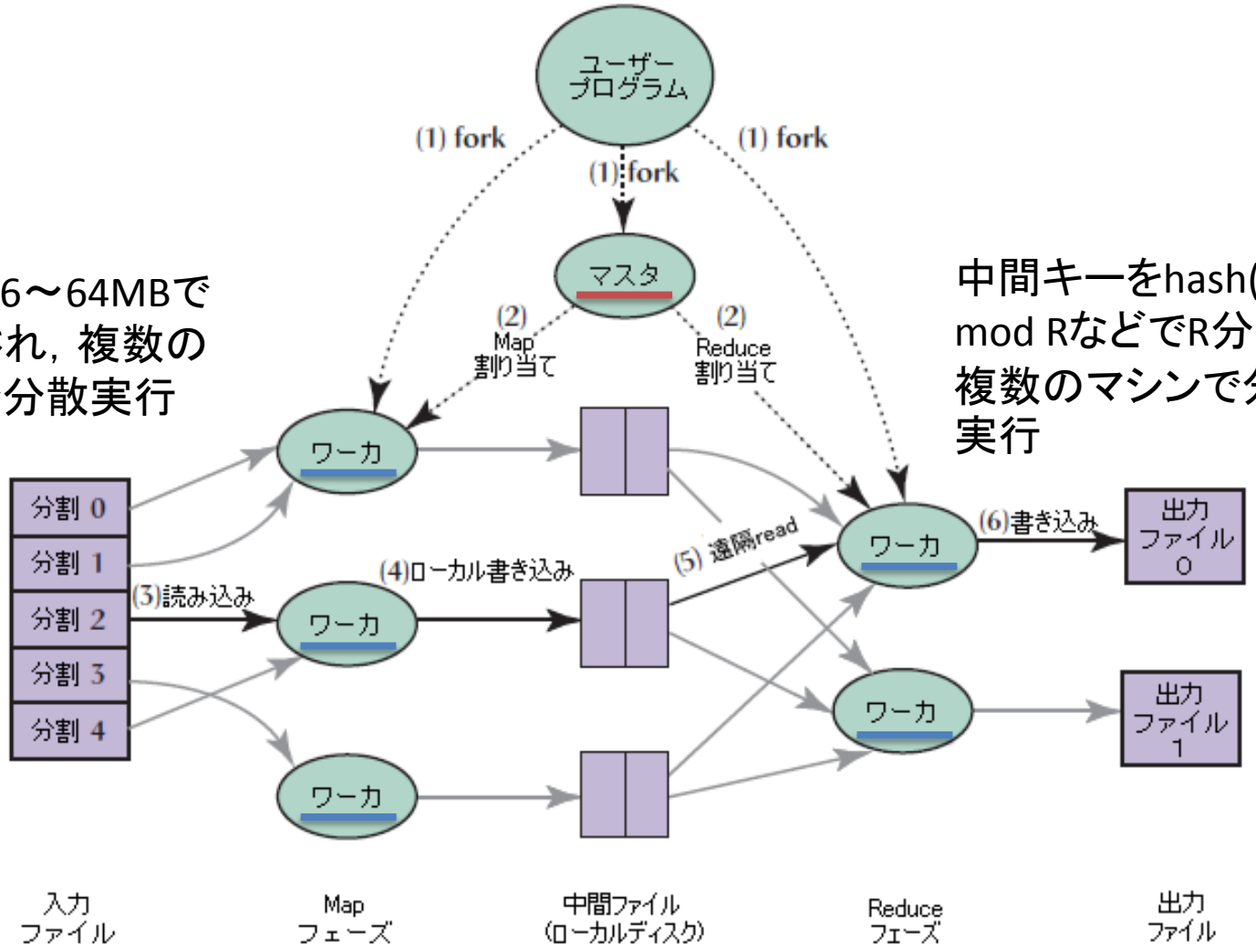
単語出現頻度の疑似コード例

```
map(String key, String value):
// key: 文書名
// value: 文書の内容
for each word w in value:          //   単語wごとに(w, “1”)を出力
    EmitIntermediate(w, “1”);

reduce(String key, Iterator values):
// key: 単語
// values: 出現カウントのリスト
int result = 0;
for each v in values:              // valuesの総和をresultに代入
    result += ParseInt(v);
Emit(AsString(result));
```

実行の概要

入力は16～64MBでM分割され，複数のマシンで分散実行



中間キーをhash(key) mod RなどでR分割し，複数のマシンで分散実行

耐故障性

- 数百～数千ノードを利用するため故障対応は必須
- ワーカ障害処理
  - マスタが全ワーカにハートビート処理（ping）
    - タイムアウトで検知
  - 障害ノードで処理済みmapタスク，処理中map，reduceタスクを全て未実行状態にし再スケジュール
    - Mapタスクの出力はローカルディスクなので実行完了でも中間データが読み出せない
    - Reduceタスクは共有ファイルシステムに出力されるためOK
- マスタ障害対応
  - チェックポインティングで可能であるが，シングルマスタなので障害はまれ

局所性

- 安価なPCクラスタの場合，ネットワークバンド幅（1Gbps）がボトルネックとなる
- 入力データはGoogleファイルシステム（GFS）に格納
  - ワーカノードのローカルディスクに格納
  - 64MBブロックに分割，3つのコピーを異なるノードに格納
- マスタは，mapタスクのスケジューリングに入力ファイルの位置情報を利用
  - 対応する入力ファイルを保持するノード
  - 同一ラックのノード
- ローカルディスクからの読み出しを多く

タスクの粒度

- mapタスク：M，reduceタスク：R
- M, R >> ワーカノード数が理想
  - 動的負荷分散のため
  - 障害時の再実行でワーカノード数に対し十分な数のタスクがあると負荷分散が可能
- M, Rの制約
  - 実装の問題：マスタはO(M+R)でスケジューリング，O(M*R)のメモリ容量
  - Mは入力データが16～64MBとするのがよい
  - Rはワーカノード数の数倍程度
    - 典型例：ワーカノード数2,000に対し，M=200,000，R=5,000

バックアップタスク

- 少数の落ちこぼれノードのために，全体処理時間が伸びてしまう
  - ディスク障害により30MB/s→1MB/sに性能低下
  - 同一ノードに複数タスクが割り当てられ，資源競合
- MapReduce処理の完了付近で，実行中タスクをバックアップタスクとして割当
  - どちらかが完了すればOK
- バックアップ処理による全体処理の増加を数%となるよう最適化
- ソートの例：バックアップタスクなしでは実行時間44%増

さらなる改善

- 中間データをR分割する関数の指定
- 中間key/valueペアのkeyにおける整列の保証
- コンバイナの指定
  - 同一mapタスクで生成される同一keyのvalueを部分的に結合
  - ネットワーク転送データ量を削減
- 入力，出力のカスタムデータ型
- デバッグのための単一マシン実行モード
- 状態表示のためのマスタのhttpサーバ機能

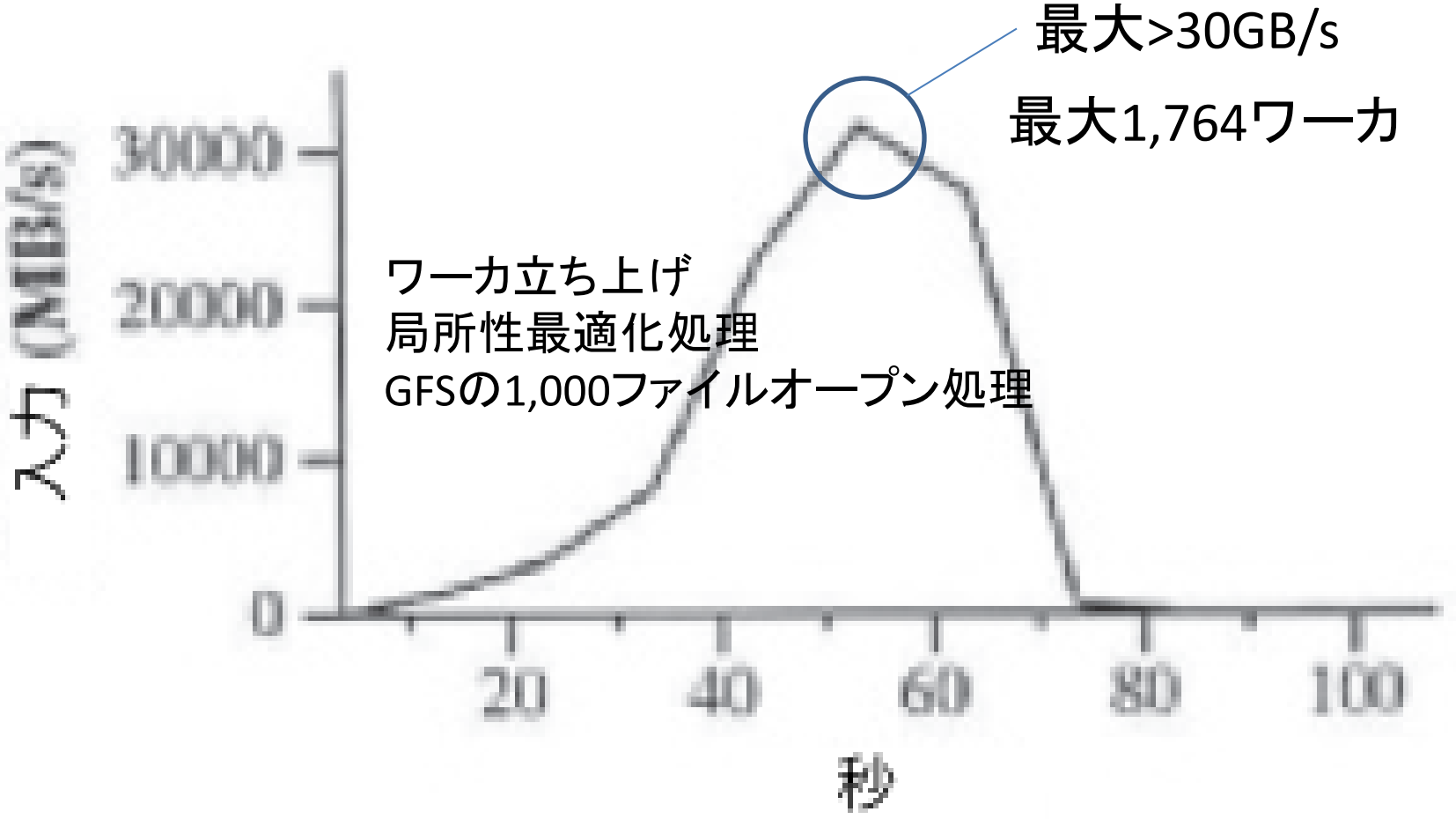
性能評価環境

- 1,800ノードのクラスタ
  - HT有効2GHzデュアルXeon
  - 4GBメモリ
  - 2x160GB IDE
  - Gigabit Ethernet
- ネットワーク構成
  - 二段スイッチ
  - ルートで利用可能なバンド幅100～200Gbps
- 同一施設内，RTT～1ミリ秒未満

Grep

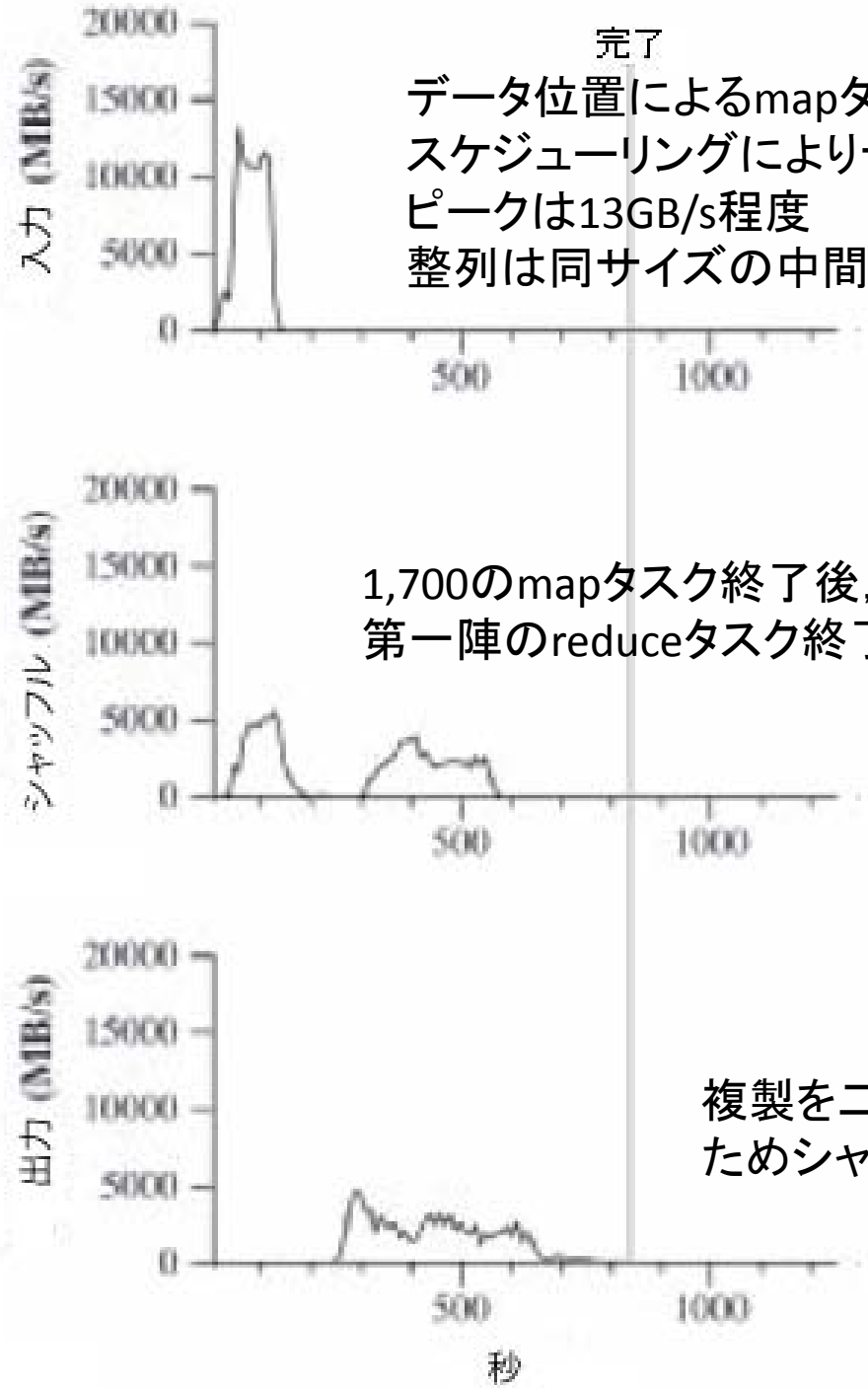
- 100B/レコード，10^{10}レコード（～1TB）
- 3Bのパターン検索
  - 出力は92,337レコード
- M=15,000（入力を約64MBに），R=1

データ転送レート


最大>30GB/s
最大1,764ワーカ
ワーカ立ち上げ
局所性最適化処理
GFSの1,000ファイルオープン処理
入力 (MiB/s)
30000
20000
10000
0
20
40
60
80
100
秒

整列

- 100B/レコード，10^{10}レコード（～1TB）の整列
  - Cf. TeraSortベンチマーク
    http://sortbenchmark.org/
- 50行以下のプログラム
- 最終結果はGFSに格納（複製数2）
- M=15,000，R=4,000
- 中間出力分割関数はkeyの先頭数B（12bit?）
  - 一般的にはkeyの分散の知識が必要
  - サンプル点で事前実行



データ位置によるmapタスクの
スケジューリングによりデータ転送レートが高い
ピークは13GB/s程度
整列は同サイズの中間データの書き出しが必要

1,700のmapタスク終了後，reduceタスクにデータ転送
第一陣のreduceタスク終了後第二陣を開始

複製を二つ作成（可用性のため）している
ためシャッフルより転送レートが低い

大規模インデックスの例

- Googleでは，ウェブ検索用インデックス処理を完全にMapReduceに移行
  - コードがシンプルに．C++で3,800行→700行
  - インデックス処理の変更が簡単に
  - 障害が実行時ライブラリで解決することにより，オペレータ介入の必要がなくなる
  - 動的なマシン追加により簡単に性能向上が可能

MapReduceのまとめ

- MapReduceは広く利用されるようになった
  - プログラミングモデルが簡単
  - 並列化，データ分散，耐障害性，局所性，負荷分散を意識しなくてよい
  - 多くの問題が表現可能
  - 数千台までスケール
- プログラミングモデルを制限することにより，上記の多くの利点を得た