GPUを考慮したMapReduceのタスクスケジューリング

白幡 晃一†1　佐藤 仁†1　松岡 聡†1†2†3

†1 東京工業大学
†2 科学技術振興機構
† 3 国立情報学研究所

情報爆発時代における大規模データ処理

- 大規模データ処理
  - 気象、生物学、天文学、物理学など様々な科学技術計算での利用
- MapReduce
  - 大規模データ処理のためのプログラミングモデル
  - スケーラブルな並列データ処理
- GPGPU
  - 汎用CPU に比べ高い性能
  - CUDA をはじめとする開発環境の整備

  → CPU と GPU の混在したスパコン・クラウドの出現

  ex) TSUBAME 2.0 : NVIDIA の Fermi「Tesla M2050」を計算ノードに3台搭載

GPU を活用した MapReduce処理の高速化

CPU と GPUの混在環境での MapReduce処理における問題点

- CPUとGPUへタスクを割り振る方法は自明ではない
  - 計算資源への依存
    - CPUコア数、GPU台数、メモリ容量、メモリバンド幅
      ストレージへの I/O バンド幅など
  - アプリケーションへの依存
    - GPU処理の特性
      - 長所：ピーク性能、メモリバンド幅
      - 短所：複雑な命令は CPU に劣る

    → GPU利用による性能向上があるものの、CPU・GPU処理には向き・不向きがある

GPUとCPUのハイブリッド実行のスケジューリングを行い，
それぞれの長所を生かす → 計算資源を有効活用

目的と成果

- 目的
  - CPUとGPUの混在環境でのMapReduce処理の高速化
- 成果
  - Mapタスクのハイブリッド実行
    - MapReduce の OSS実装である Hadoop に実現
  - Mapタスク割り振りのスケジューリング手法の提案
    - ジョブ全体の実行時間を最小化
  - K-meansアプリケーションによる評価
    - ジョブ実行時間: 複数GPUと提案スケジューリング手法により、CPUのみの場合に対し1.02-1.93倍となる高速化

発表の流れ

- 研究背景
- MapReduceとGPGPU
- 提案手法
- 設計と実装
- 実験
- 関連研究
- まとめと今後の課題

MapReduce と GPGPU

- MapReduce
  - Map, Shuffle, Reduce の 3つのフェーズからなる
  - 様々な機械学習アルゴリズムへの適用例
  - 主な実装
    - Hadoop: MapReduce, HDFS, HBase などの OSS
    - Mars: GPU用のフレームワーク

  → 企業・研究機関での導入事例の多い Hadoop を対象
- GPGPU
  - グラフィックプロセッサをSIMD演算機として利用
  - CPUに比べ安価で高いピーク性能
  - 主な統合開発環境
    - NVIDIA: CUDA, AMD: ATI Stream

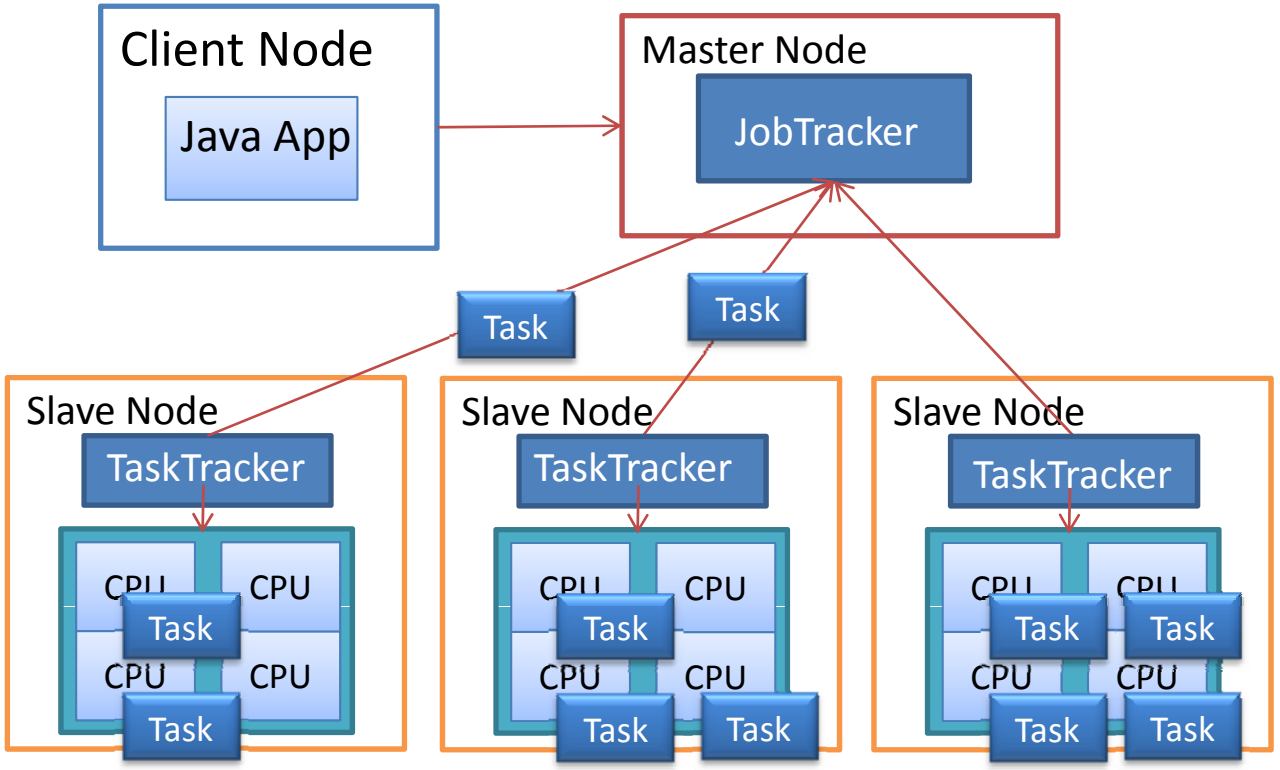
  → CUDAを対象

発表の流れ

- 研究背景
- MapReduceとGPGPU
- 提案手法
- 設計と実装
- 実験
- 関連研究
- まとめと今後の課題

Hadoop の構造

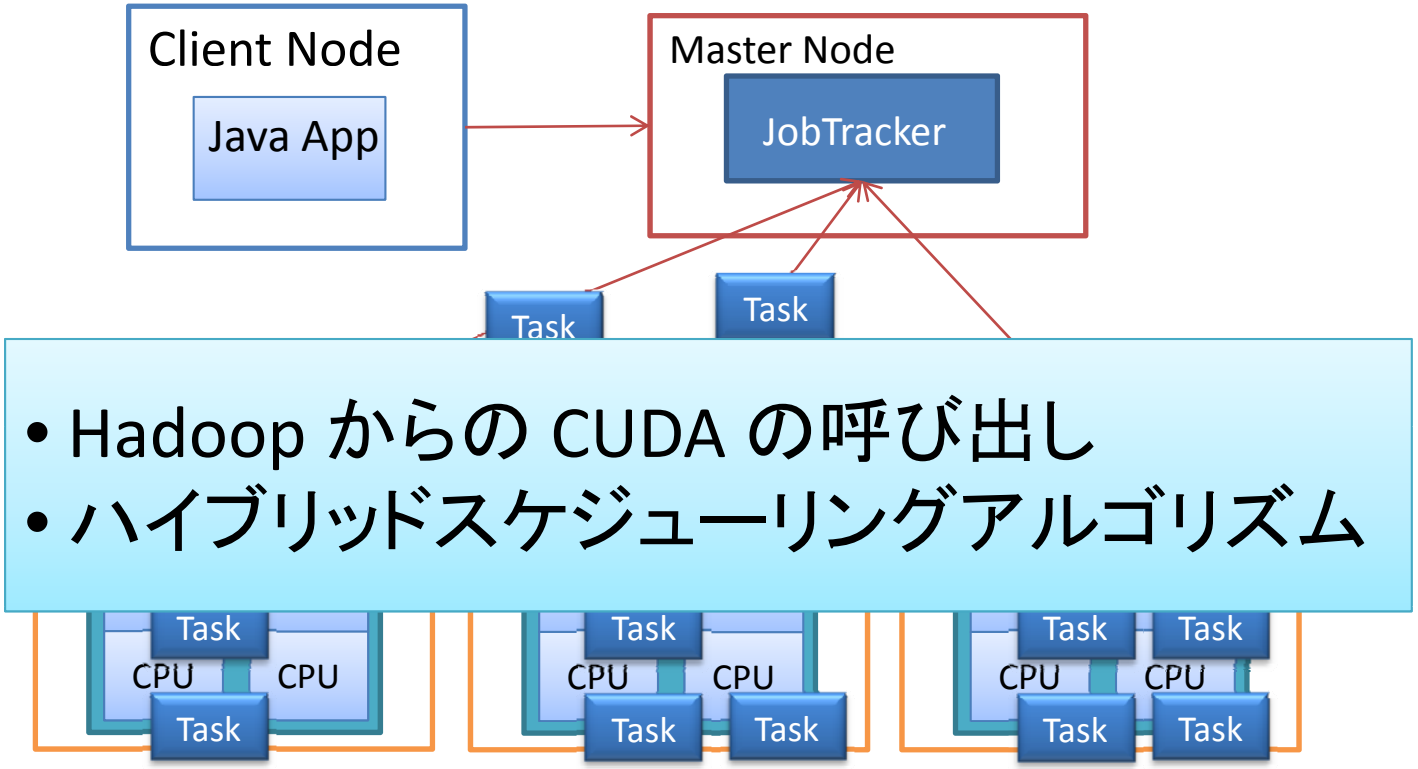
- マスタ・ワーカモデル
- ジョブの動作
  - Client: ジョブの投入，JobTracker: タスクの割り振り，TaskTracker: タスクの実行，Heartbeat による送受信



提案：CPU と GPUの Mapタスクの ハイブリッドスケジューリング

- CPU と GPU への割り振りを自動決定
  - 実行環境、計算資源
  - アプリケーションの特性

→ ジョブ実行時間を最小化



Hadoop からの CUDA の呼び出し

- 実装の違いの吸収
  - Hadoop → Java実装 (ミドルウェア，アプリケーション)
  - CUDA → C++ によるライブラリ
- 主な手法
  - Hadoop Streaming: 標準入出力
  - Hadoop Pipes: ソケット接続，C++ライブラリ
  - JNI，JNIベースの CUDAラッパー (JCUDA)

→ Hadoop Pipes を対象

- C++ で MapReduceアプリケーション, CUDAカーネル関数を記述可能
- CUDAカーネルを起動可能

Hadoop からの CUDA の呼び出し (cont'd)

- Mapタスク・実行スロットの管理
  - Mapタスクを CPU, GPU のどちらのスロットで実行するか？
  - 空きスロット (CPU, GPU) の認識
- GPUへのMapタスクの競合
  - 1ノード上に複数GPU を搭載時に発生
  - Mapタスクを実行する GPUデバイスを識別する必要有
    - cudaSetDevice で設定

Mapタスクの
ハイブリッドスケジューリング

- 設定
  - CPU nコア と GPU m台を搭載
- スケジューリング方法
  - ジョブ実行時間を最小化
    - CPU と GPU の性能比 (加速倍率) に応じた Mapタスクの割り振り
  - 動的なモニタリング
    - CPU, GPU で実行が終了した Mapタスクのプロファイル (実行時間など) を Heartbeat 毎に繰り返し取得
    - 加速倍率を計算
    - ジョブの進捗状況の把握

スケジューリングアルゴリズム

- 目標
  - CPU, GPU に割り当てられた Mapタスクが共に全て終了するまでにかかる時間の最小化
  - CPU と GPU に割り当てる Mapタスク数を決定
- 加速倍率:

$$\alpha = (\text{平均}CPU\text{タスク実行時間})/(\text{平均}GPU\text{タスク実行時間})$$

- スケジューリングアルゴリズム

$$\text{minimize } f(x,y)$$

$$\text{subject to } f(x,y) = max\{\lceil\frac{x}{n}\rceil\ \alpha\ t, \lceil\frac{y}{m}\rceil t\}$$

$$x + y = N$$

$$x, y \geqq 0$$

既知　CPU コア数：n, GPU 台数：m
モニタリング　残りタスク数：N,
1GPU タスク実行時間：t, 加速倍率：α
出力　CPU 上で実行する合計タスク数：x,
GPU 上で実行する合計タスク数：y

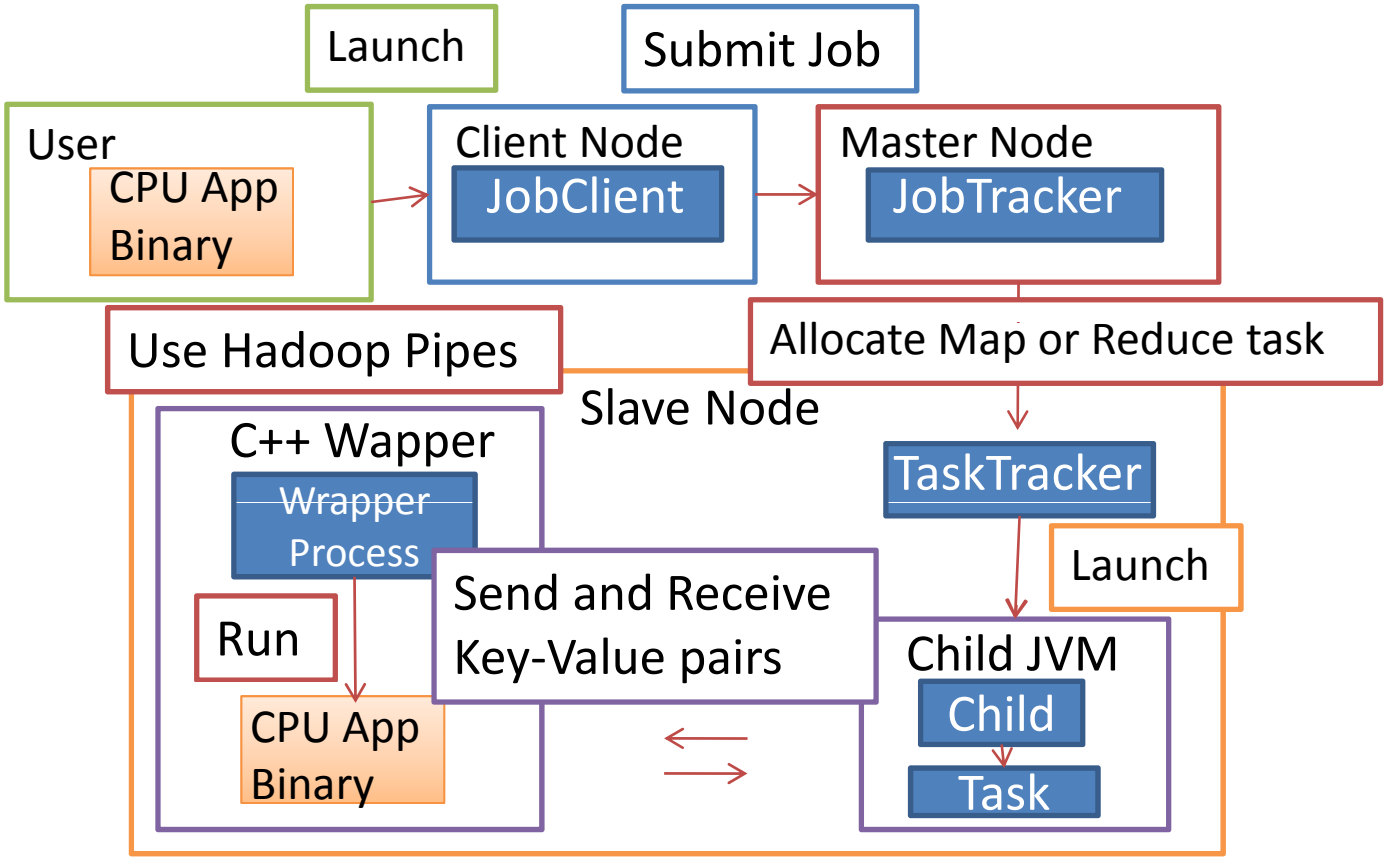
発表の流れ

- 研究背景
- MapReduceとGPGPU
- 提案手法
- 設計と実装
- 実験
- 関連研究
- まとめと今後の課題

Hadoop Pipes ユーザからの実行方法

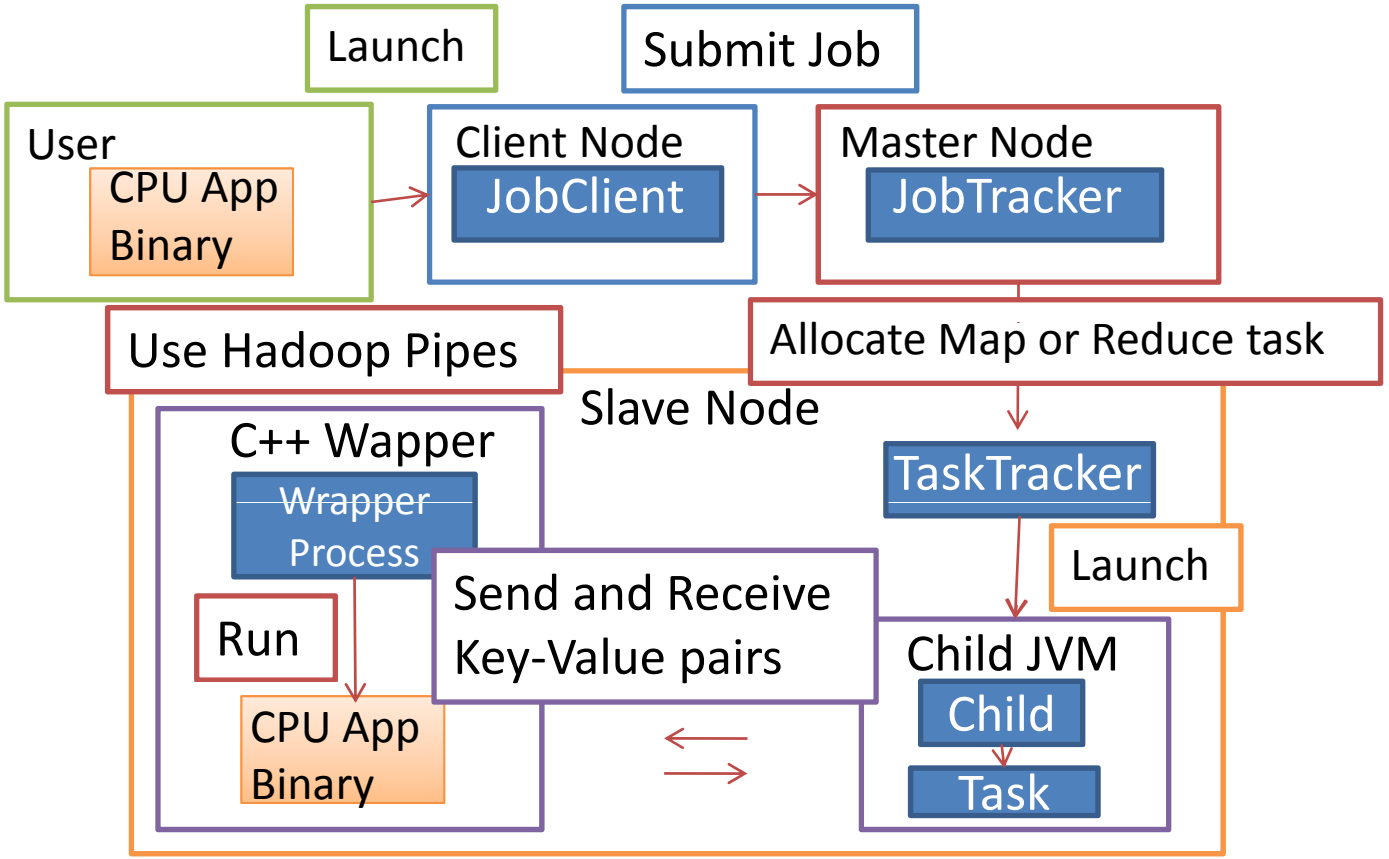
- ユーザは map関数と reduce関数を記述 (C++ラッパーライブラリを使用)
- コンパイル済のバイナリと入力・出力ファイルを指定してジョブを実行

```
bin/hadoop pipes¥
 -D  hadoop.pipes.java.recordreader=true¥
 -D  hadoop.pipes.java.recordwriter=true¥
 -bin  cpu-kmeans2D¥
 -input  input¥
 -output  output
```



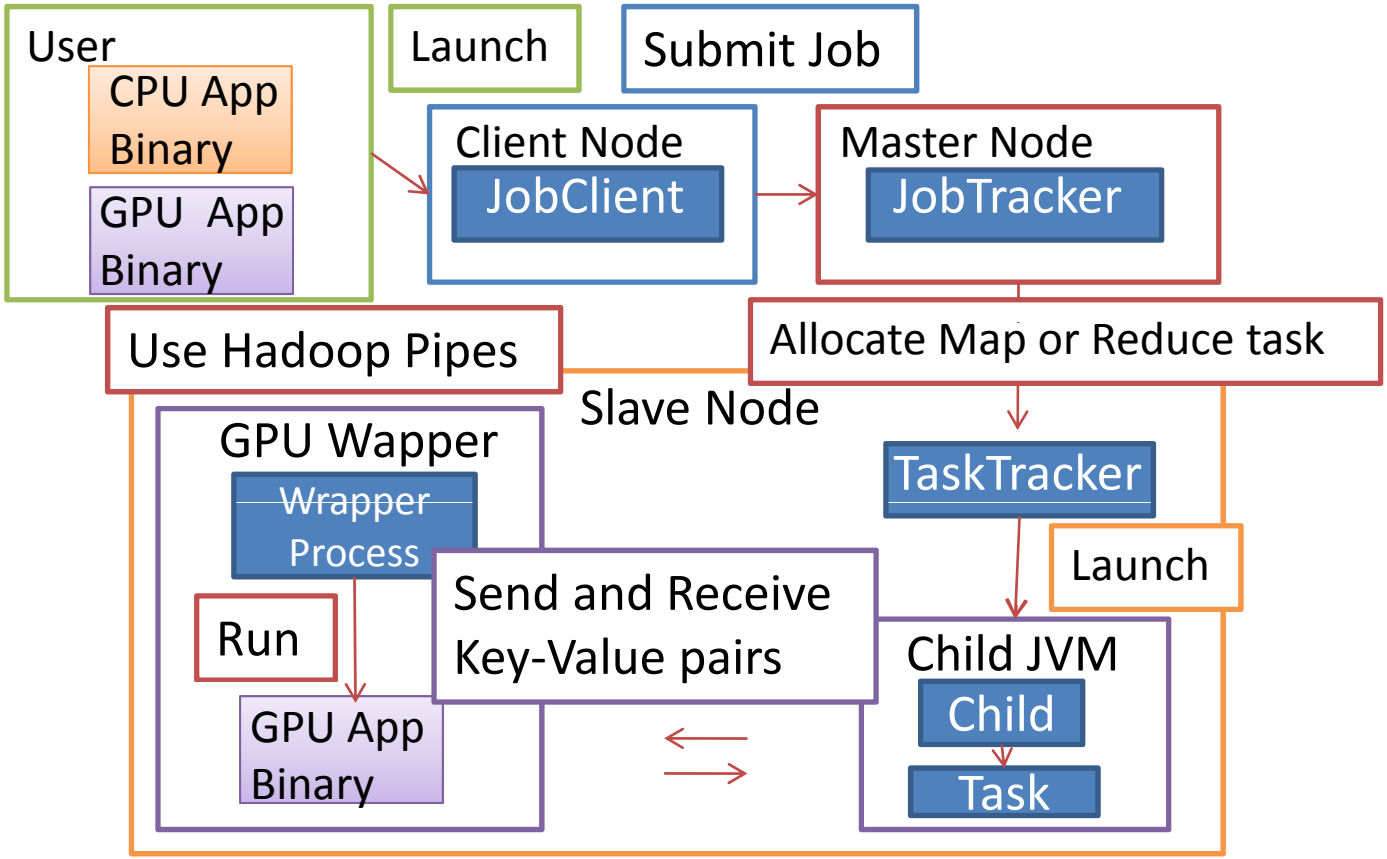
Hadoop Pipes 実行フロー

- TaskTracker が Childプロセスを起動
- Child プロセスがタスクを起動
- C++ Wrapper プロセスが C++バイナリを実行
  - ソケット接続による Key-Value ペアの送受信



ハイブリッド実行の場合
incl. Pipes による CUDA の呼び出し

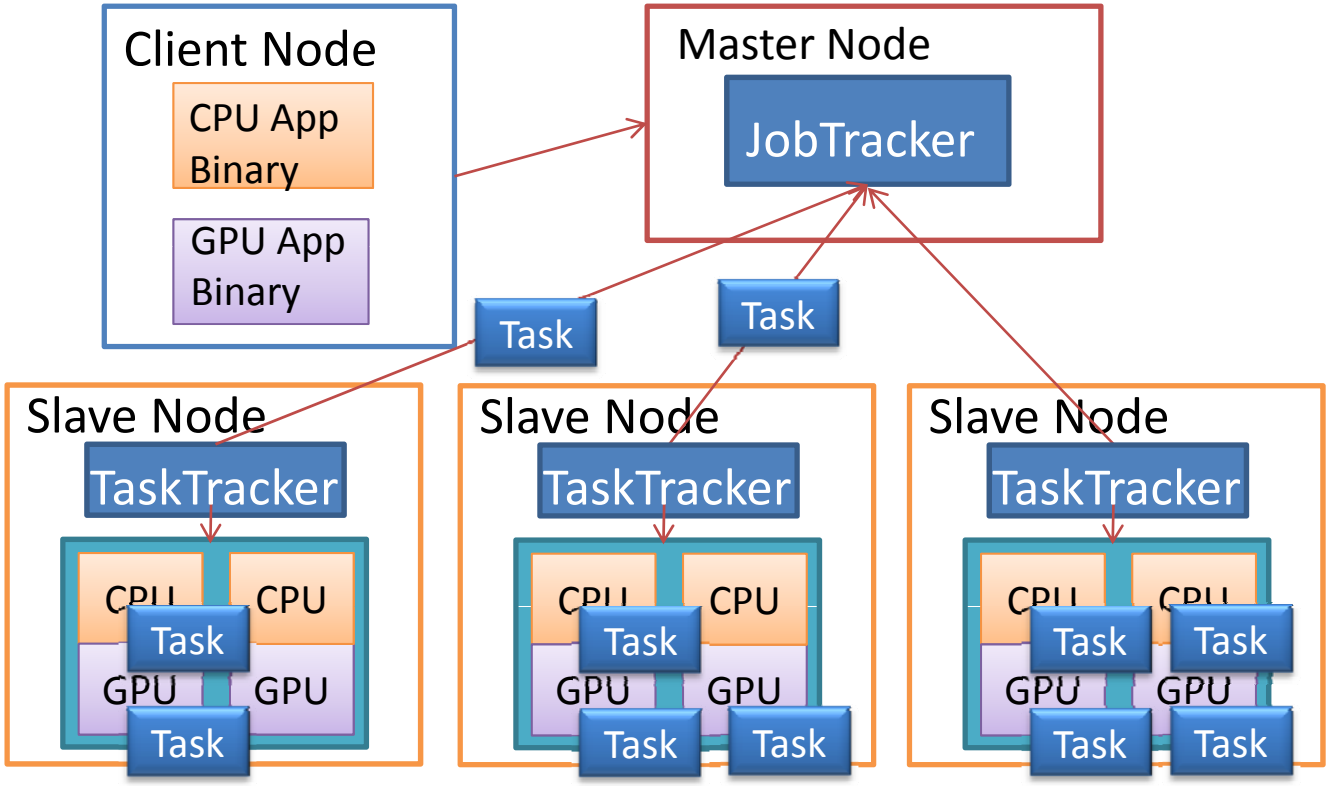
- ジョブ実行時に CPU・GPUバイナリを指定
- Mapタスクの識別 (CPU・GPU, GPUデバイスID)
- 空きスロットの識別 (CPU・GPU, 空きGPUデバイスID)



Mapタスクの割り振り

JobTracker
- プロファイルのモニタリング
- スケジューリングアルゴリズムの計算 (Heartbeat毎)
- 各ノードへのタスクの割り振り (CPU・GPU, GPUデバイスID の指定)

TaskTracker
- 空きスロット, プロファイル (実行時間) の通知
- タスクの割り当て要求
- タスクを識別して実行 (CPU・GPU, GPUデバイスID の識別)



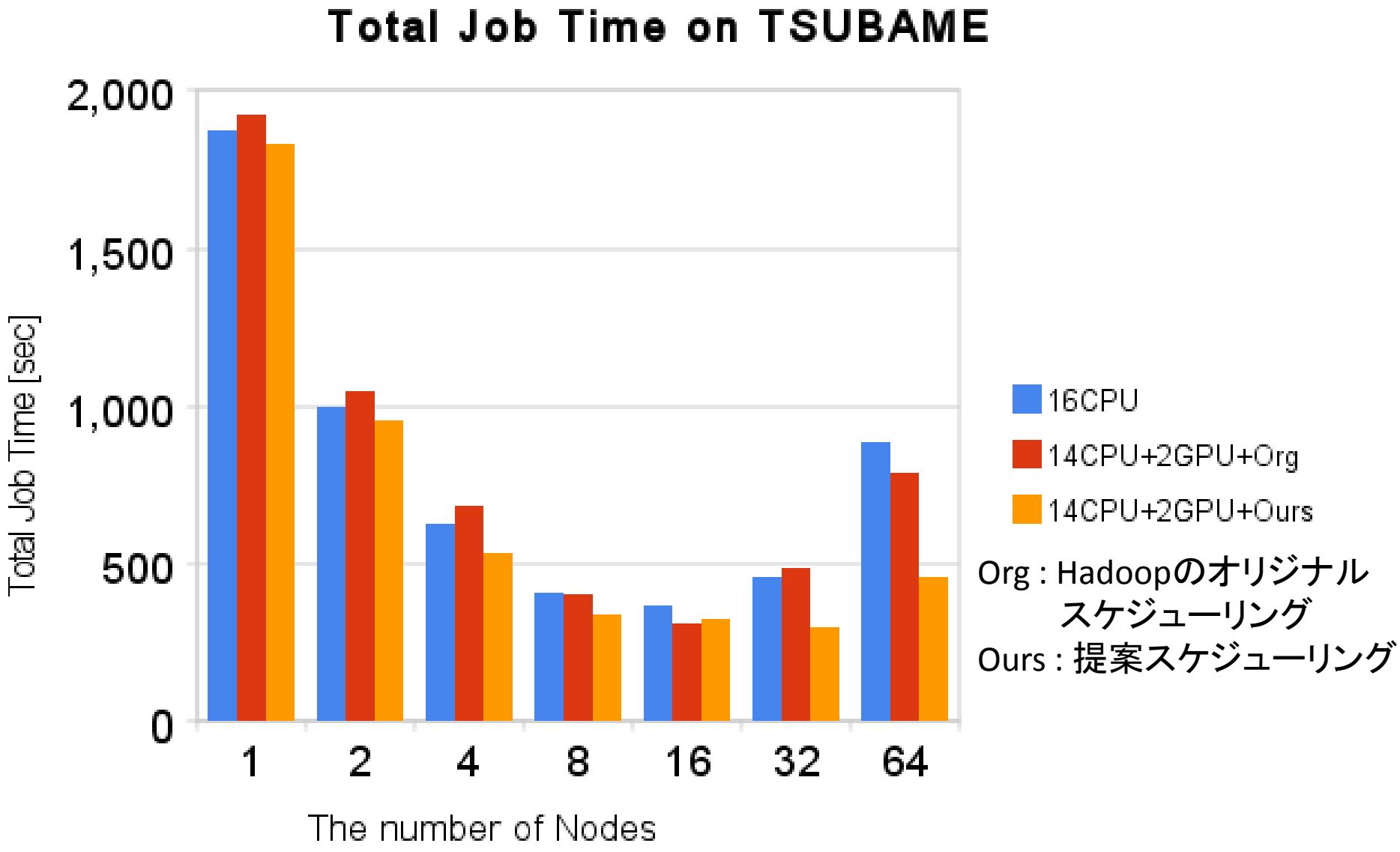
発表の流れ

- 研究背景
- MapReduceとGPGPU
- 提案手法
- 設計と実装
- 実験
- 関連研究
- まとめと今後の課題

実験

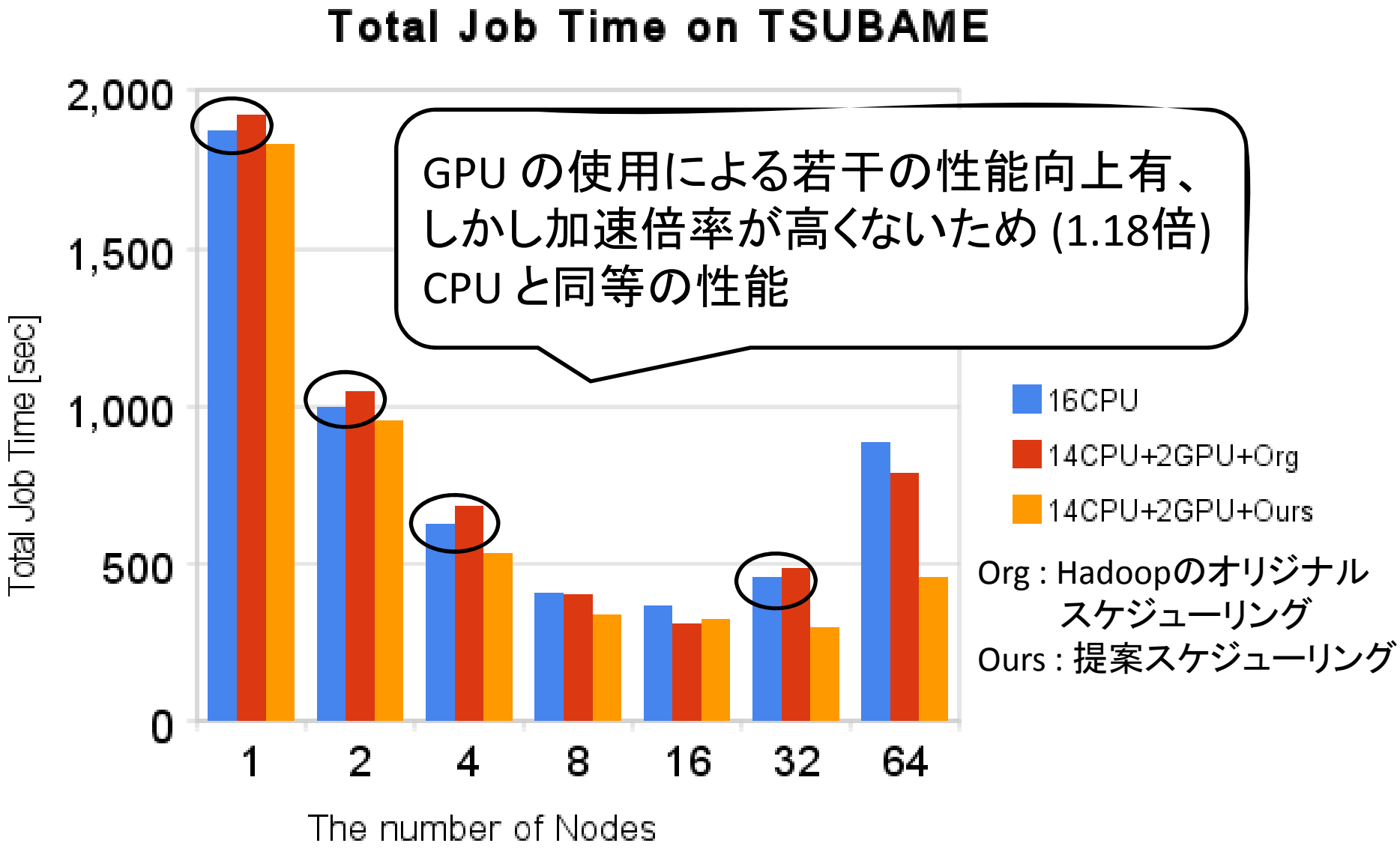
- Hadoop環境でジョブ実行時間、Mapタスク実行時間を測定
  - CPUのみ と CPU + GPU の場合を比較
  - 提案スケジューリングの適用と非適用の場合を比較
- 実験環境： TSUBAME
  - CPU 16コア + GPU 2台 / Node
  - Mapスロット数，Reduceスロット数: 16 スロット / Node
  - 1～64ノードと変化
  - DFS は Lustre を使用（ストライプ数: 4，チャンクサイズ: 32 MB）
    - I/O 性能: 32 MBのファイルに対し、Write 180MB/s，Read 610MB/s
- K-meansアプリケーション
  - Mapタスクを C++バイナリ、CUDAバイナリで実行
  - 入力 : 262,144 個の二次元座標データ（浮動小数点）と 128個のクラスタを持つファイルを 1000個連結 (20GB)
    - Map: 各ファイルごとに K-means を実行
    - Reduce: K-means の結果を収集

ジョブ実行時間の比較

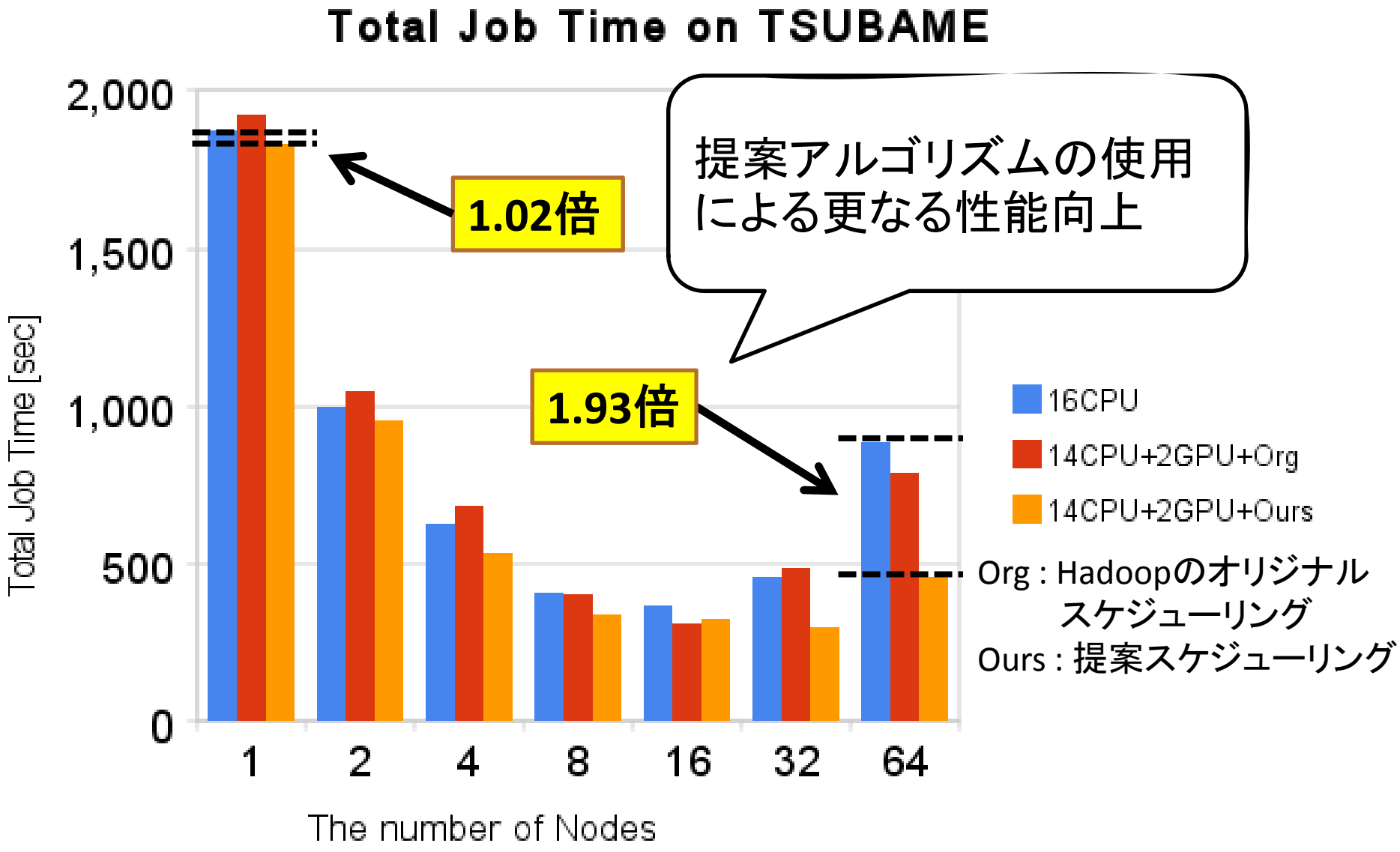


Org : Hadoopのオリジナル
スケジューリング
Ours : 提案スケジューリング

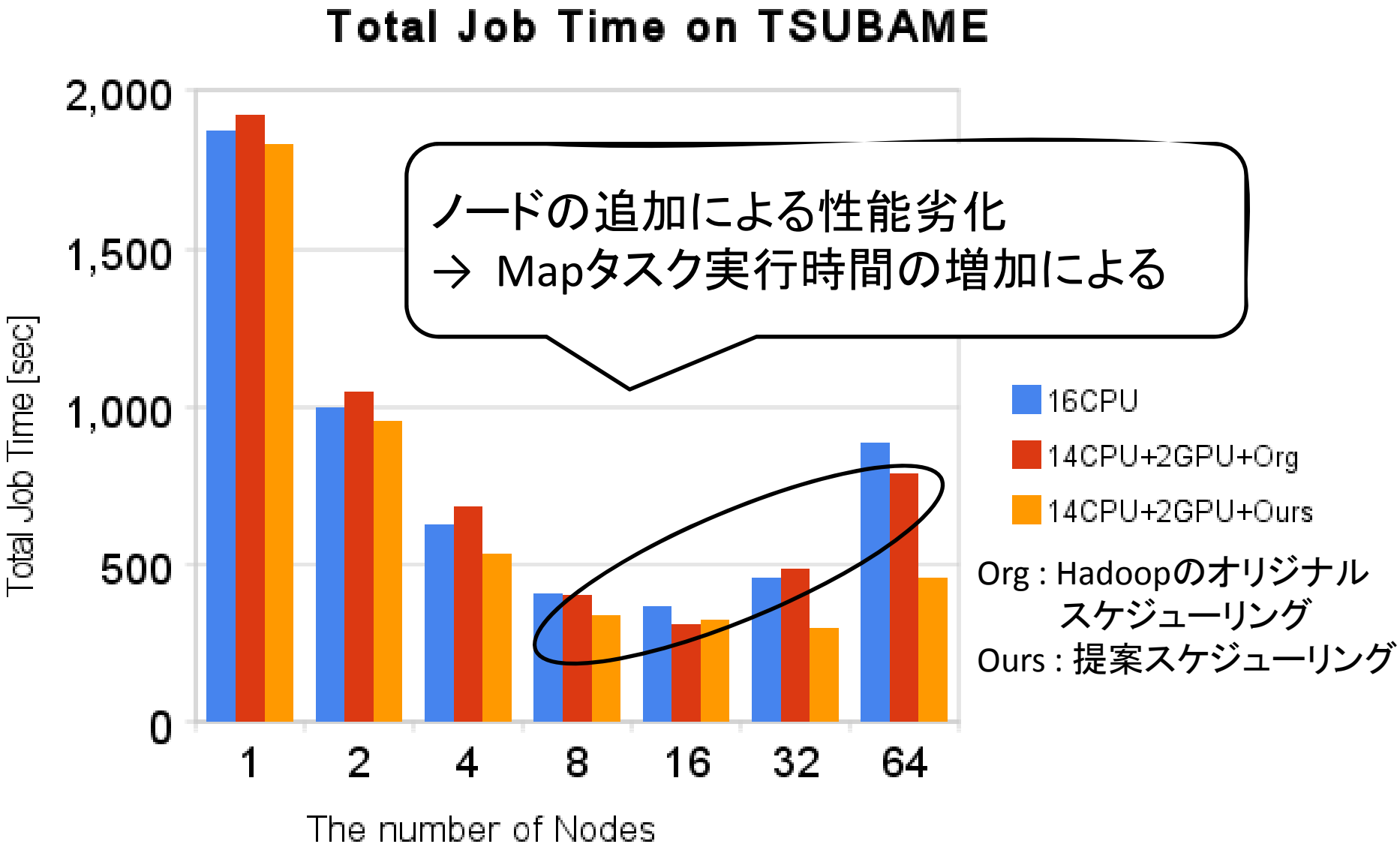
ジョブ実行時間の比較


Total Job Time on TSUBAME
2,000
1,500
1,000
500
0
Total Job Time [sec]
1
2
4
8
16
32
64
The number of Nodes
GPU の使用による若干の性能向上有、
しかし加速倍率が高くないため (1.18倍)
CPU と同等の性能
16CPU
14CPU+2GPU+Org
14CPU+2GPU+Ours
Org : Hadoopのオリジナル
スケジューリング
Ours : 提案スケジューリング

ジョブ実行時間の比較

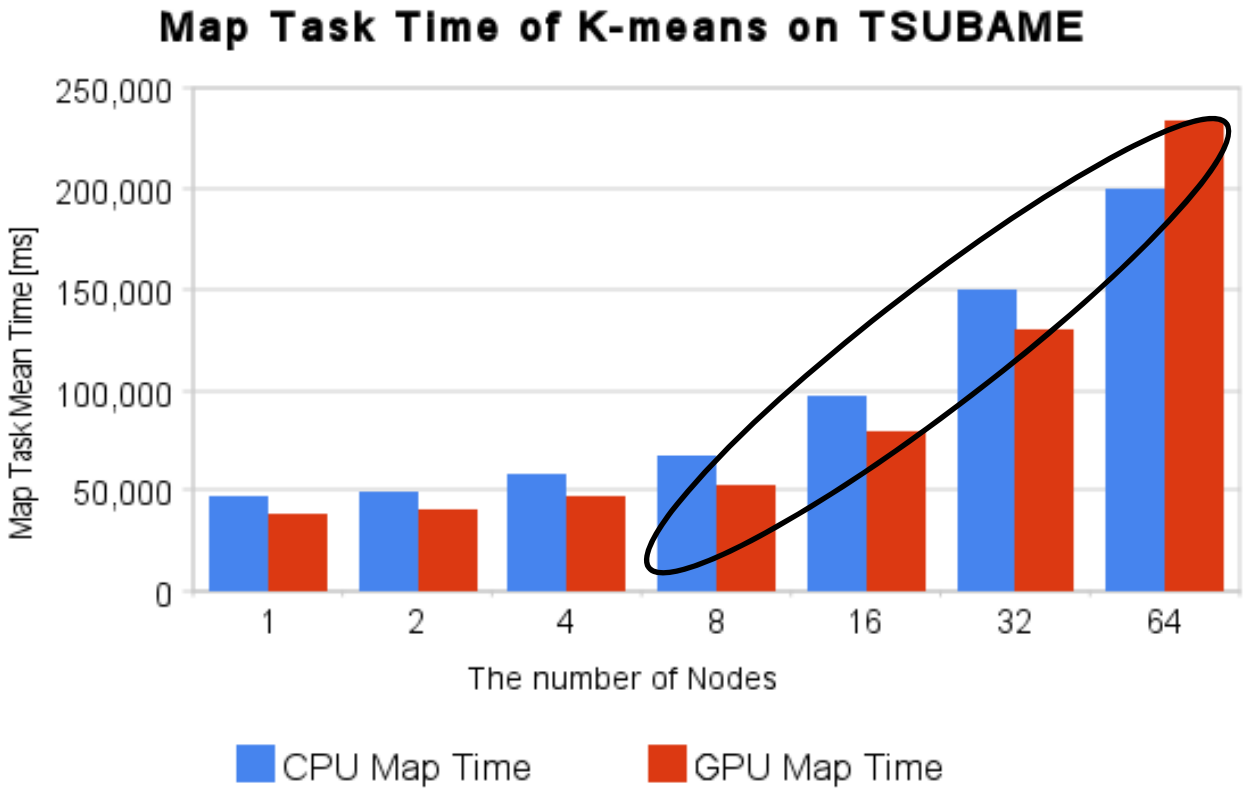

Total Job Time on TSUBAME
2,000
1,500
1,000
500
0
Total Job Time [sec]
1.02倍
1.93倍
提案アルゴリズムの使用
による更なる性能向上
16CPU
14CPU+2GPU+Org
14CPU+2GPU+Ours
Org : Hadoopのオリジナル
スケジューリング
Ours : 提案スケジューリング
1
2
4
8
16
32
64
The number of Nodes

ジョブ実行時間の比較

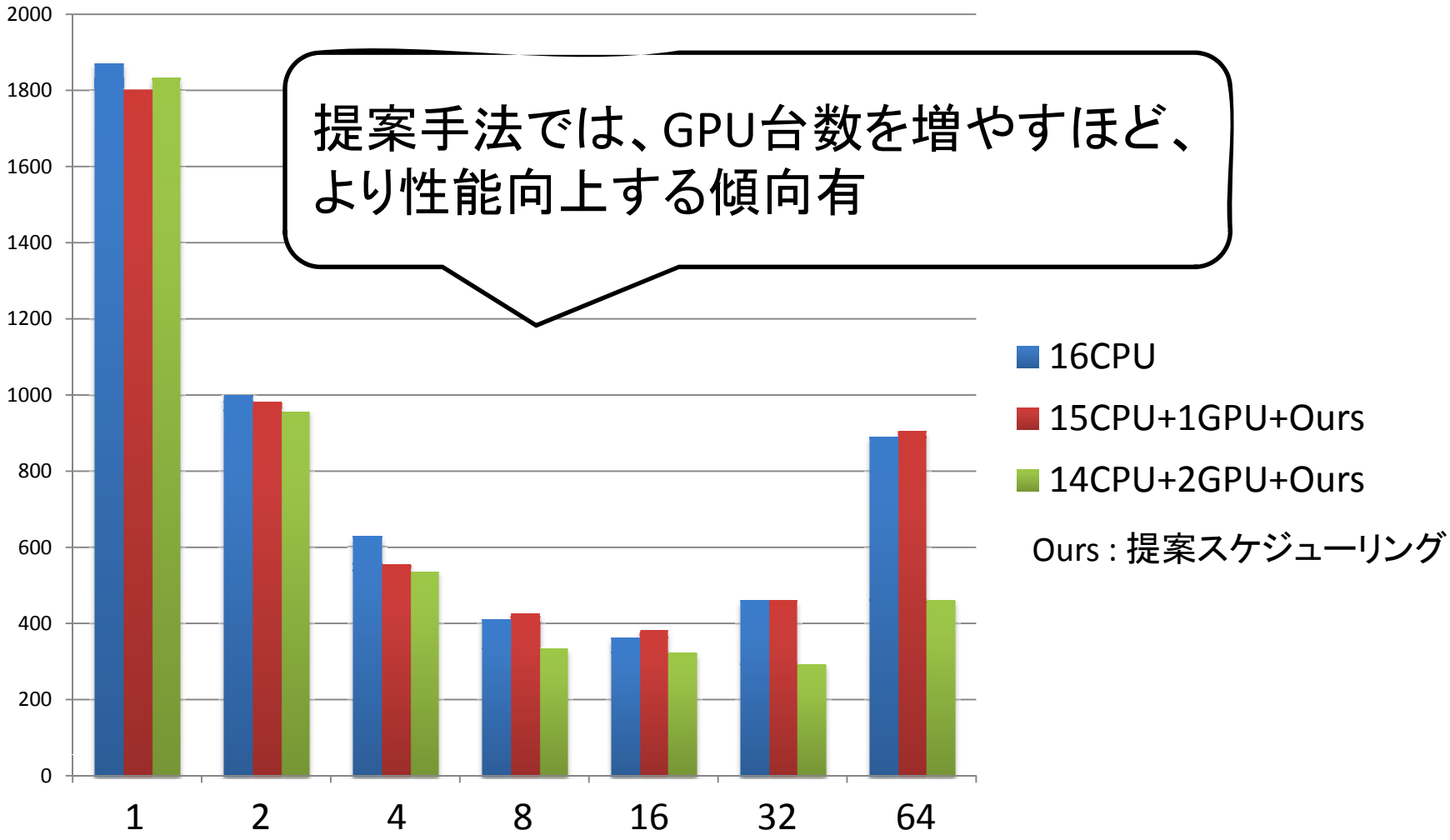

Total Job Time on TSUBAME
2,000
1,500
1,000
500
0
Total Job Time [sec]
1
2
4
8
16
32
64
The number of Nodes
ノードの追加による性能劣化
→ Mapタスク実行時間の増加による
16CPU
14CPU+2GPU+Org
14CPU+2GPU+Ours
Org : Hadoopのオリジナル
スケジューリング
Ours : 提案スケジューリング

Mapタスク実行時間の増加

- ノード数に応じてMapタスク時間が増加
  - I/O性能の低下
  - ジョブ実行時間の増加の一因
  - その他の原因として、スロット数の過剰が冗長

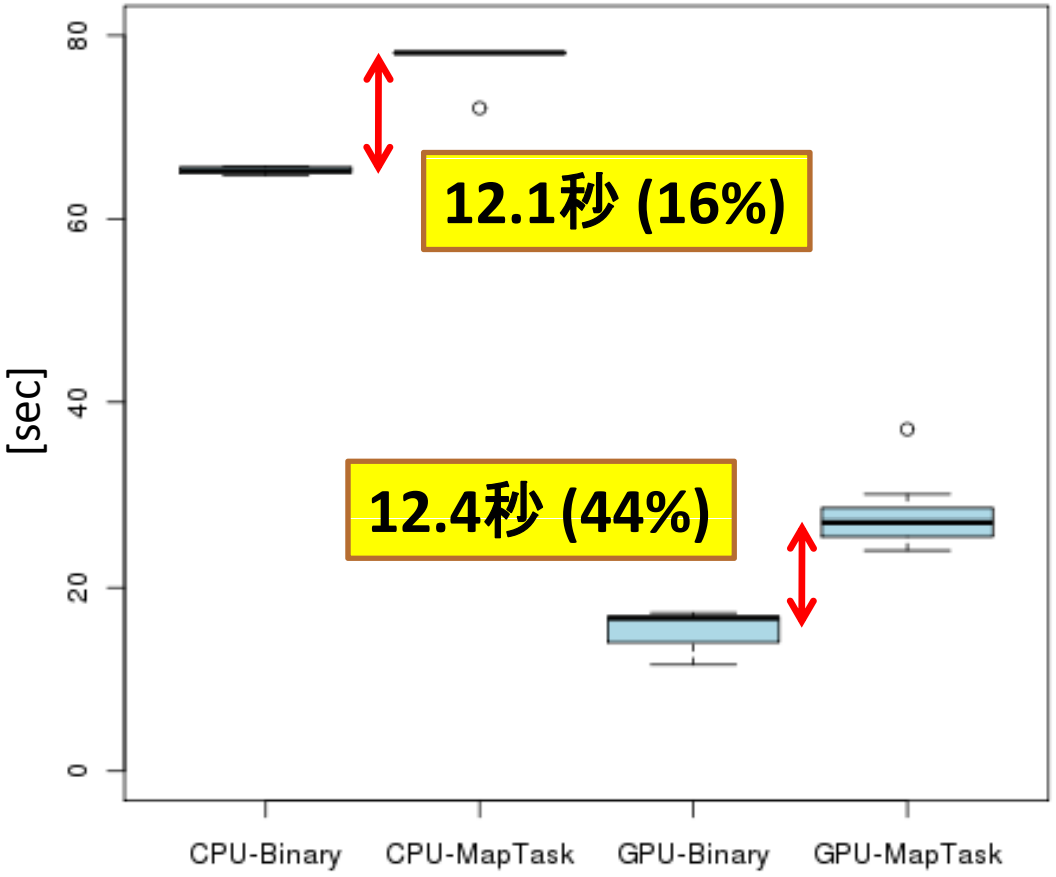


ジョブ実行時間の比較
1GPU と 2GPU



プロセス起動によるオーバーヘッド
１台の計算機での実験

- バイナリのみとMapタスクの実行時間を比較
  - バイナリ実行時間 : C++ または CUDA の Map関数
  - Mapタスク実行時間 : Mapタスクの割り当てから終了まで



→ 実装依存のオーバーヘッドは無視できない

CPU-Binary, GPU-Binary :
バイナリ実行時間
CPU-MapTask, GPU-MapTask :
Mapタスク実行時間

発表の流れ

- 研究背景
- MapReduceとGPGPU
- 提案手法
- 設計と実装
- 実験
- 関連研究
- まとめと今後の課題

関連研究

- 学習メカニズムによるCPU・GPUタスクスケジューリング
  [Chi-Keung Lu et al. `09]
- チャンクサイズの変化によるCPU・GPUハイブリッド実行の高速化
  [T. Ravi Vifnesh et al. `10]
- 不均質な環境でのMapReduceのタスクスケジューリング
  [Matei et al. `08]
- 既存の不均質環境下のタスクスケジューリング手法
  [Suda `06]

→ 計算資源・アプリの特性に応じた CPU・GPU同時実行による大規模データ処理に特化

- 資源競合 (メモリ、ストレージ) を考慮
- オンラインで自動スケジューリング

まとめと今後の課題

- まとめ
  - Mapタスクを GPU から呼び出し
    - MapReduce実装の Hadoop環境で実現
  - CPU と GPU の混在環境を考慮したタスクスケジューリング手法の提案
  - K-meansアプリケーションでの実験
    - ジョブ実行時間: 2GPUと提案手法により 1.02-1.93倍の高速化
    - Hadoopの実装に依存するオーバーヘッドは無視できない
- 今後の課題
  - スケジューリングモデルの改良
    - メモリ・ディスクアクセス等を考慮
    - GPU上でのタスク実行時における CPUとの資源の競合によるオーバーヘッドを考慮

ご清聴ありがとうございました

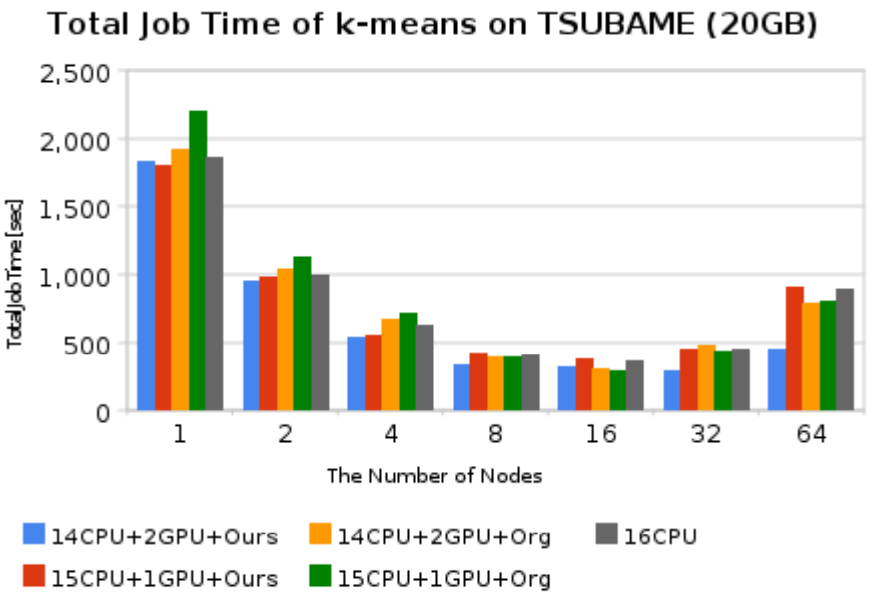
実験結果

•Mapタスク実行時間
- GPUではCPUに対し1.0-1.25倍となる高速化
（プロセス起動時間等を含む）

•ジョブ実行時間
-2GPUの使用と提案スケジューリングアルゴリズムの適用により1.02-1.93倍の高速化

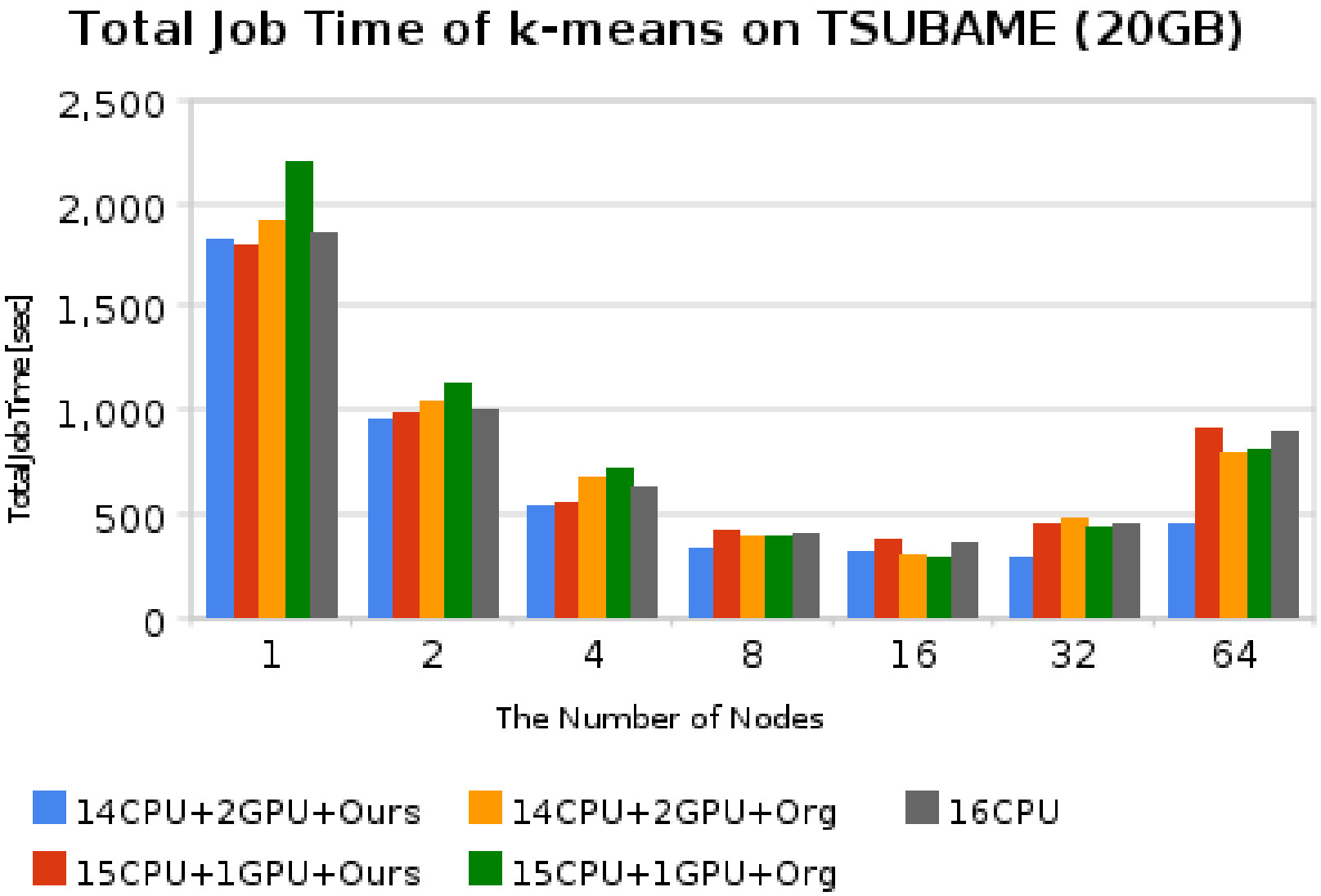
-14CPU+2GPUでは15CPU+1GPUに対し
1.02-1.29倍の高速化 (複数GPUによる効果)

- 提案アルゴリズムの使用により
14CPU+2GPUの場合に平均1.28倍、
15CPU+1GPUの場合に平均1.08倍の高速化



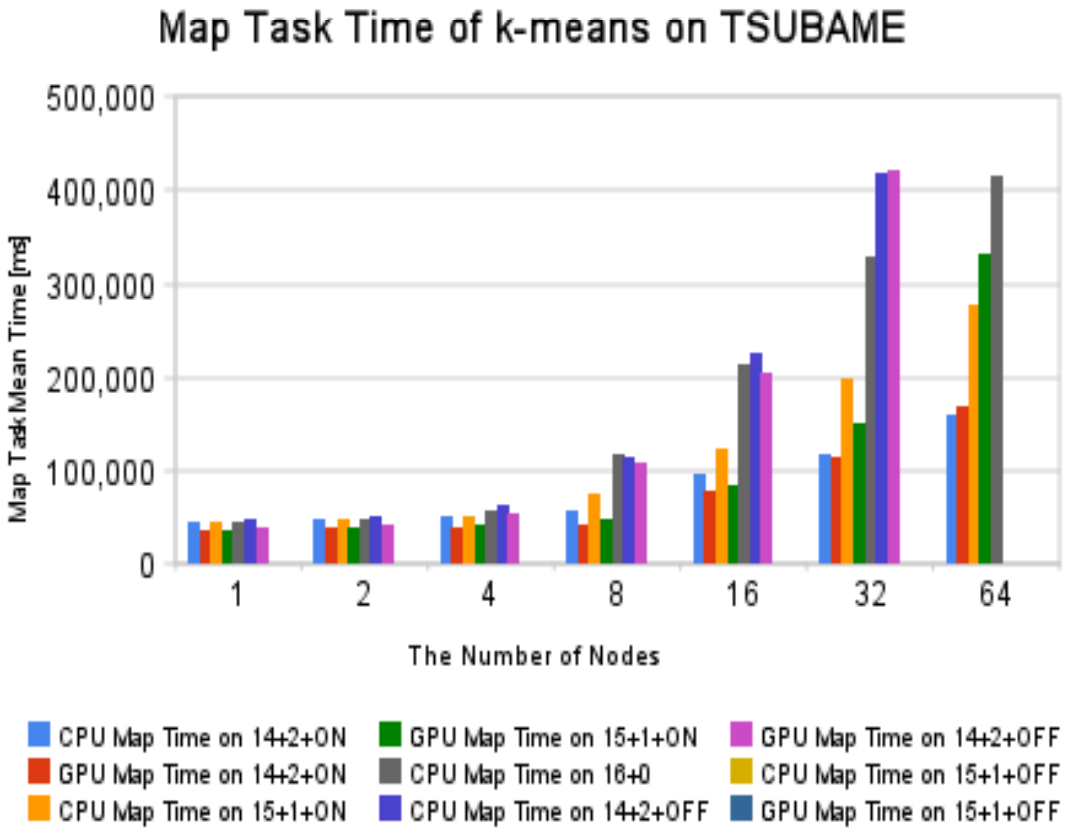
Ours : 提案スケジューリング
Org : Hadoopのオリジナルスケジューリング

- GPUの使用、提案タスクスケジューリングの適用による性能向上を確認


Total Job Time of k-means on TSUBAME (20GB)
2,500
2,000
1,500
1,000
500
0
Total Job Time [sec]
1
2
4
8
16
32
64
The Number of Nodes
14CPU+2GPU+Ours
14CPU+2GPU+Org
16CPU
15CPU+1GPU+Ours
15CPU+1GPU+Org

Mapタスク実行時間の増加

- ノードを増やすにつれてMapタスク時間が増加
  - I/Oの増加が原因か



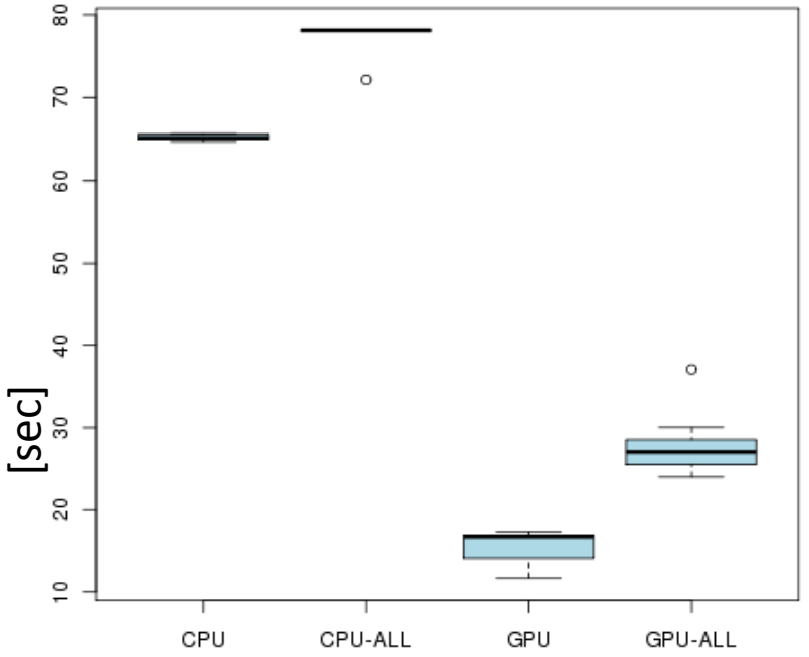
プロセス起動等のオーバーヘッド

- ローカル1ノードでの実験（7CPU+1GPU）
  - バイナリ実行時間とMapタスク実行時間を比較
    - バイナリ実行時間：C++またはCUDAのMap関数の実行時間
    - Mapタスク実行時間：Mapタスクが割り当てされてから終了するまでの時間
  - オーバーヘッド
    - CPU上では約16%
    - GPU上では約44%

→ 実装依存のオーバーヘッドは無視できない

	CPU	GPU
バイナリ実行時間	64.93～65.68 秒	11.66～16.86 秒
Mapタスク実行時間	72.13～78.14 秒	24.04～37.06 秒

バイナリ実行時間とMapTask実行時間の比較



GPUを呼び出す手法の比較

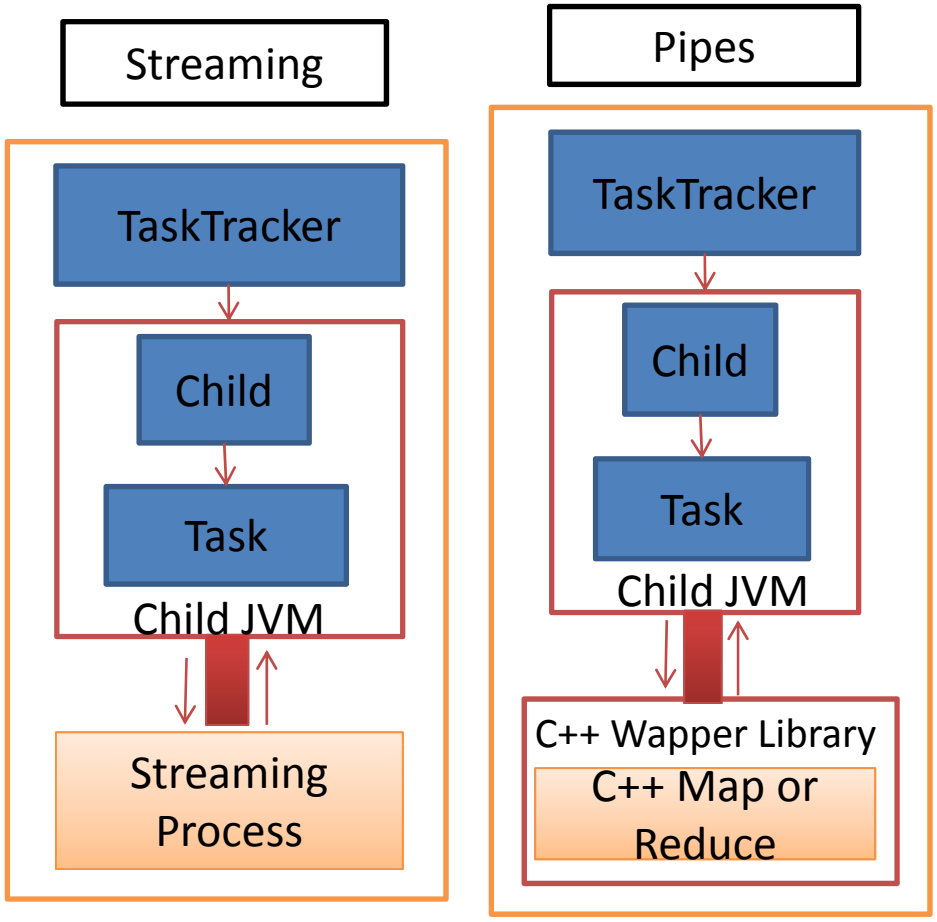
- Hadoop Streaming
  - Hadoopとユーザプログラムの間にUnix標準ストリームを使用
  - 標準入出力の解析が必要　×
- Hadoop Pipes
  - ソケット接続を確立し、スレーブノードとの通信のチャネルを生成するC++インターフェース
  - C++とCUDAの親和性　〇
- JNI（Java Native Interface）
  - 関数呼び出しのオーバーヘッド大、実行環境依存のライブラリが必要　×
- jCUDA（Java for CUDA）
  - 現状では明示的なメモリアラインメント不可、非同期の命令がメモリ破損を引き起こす可能性有　×

→ C++との親和性を考慮しHadoop Pipesを使用

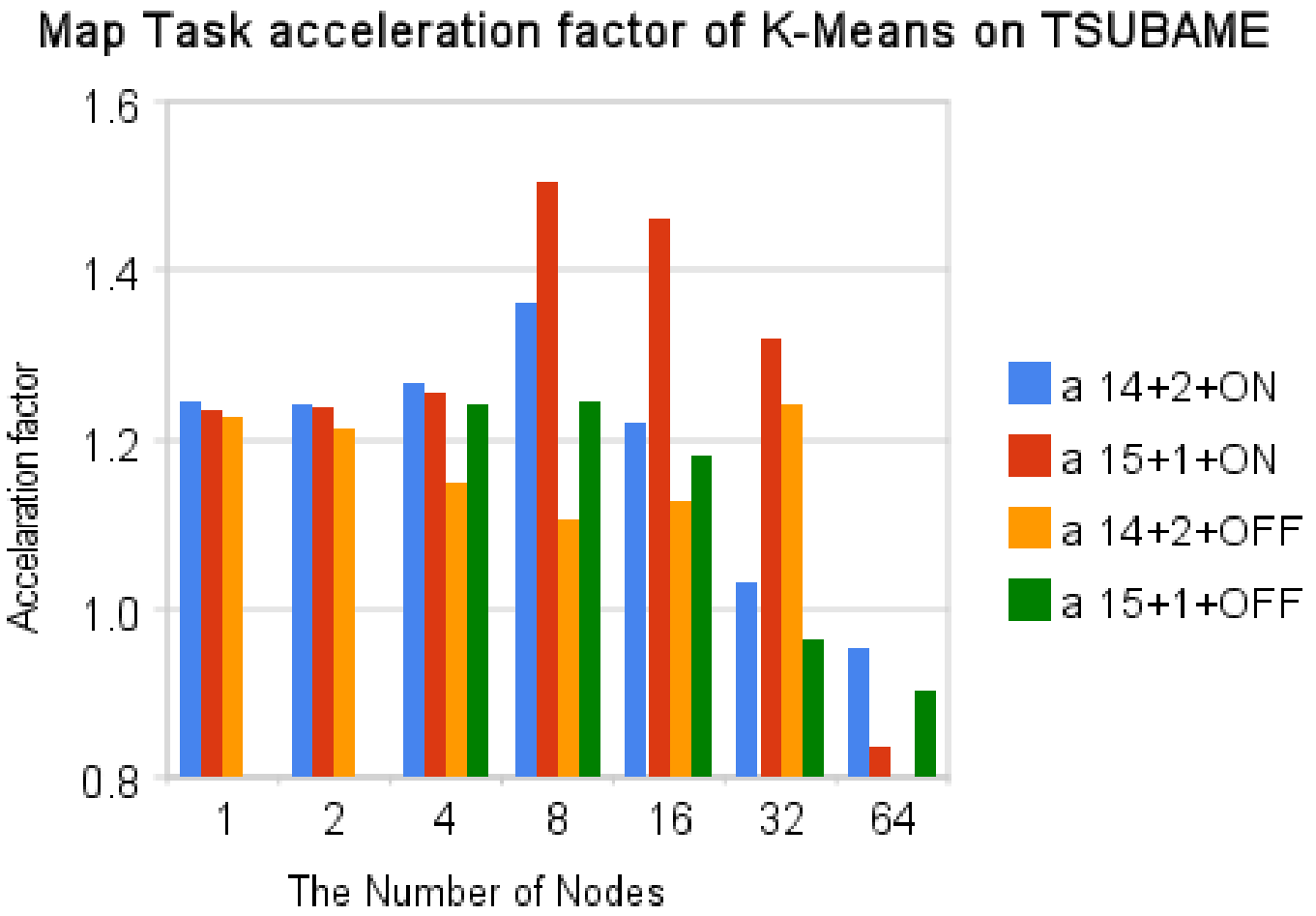
GPUを呼び出す手法の比較

- Hadoop Streaming
  - HadoopとユーザプログラムのインターフェースにUnix標準ストリームを使用
  - 任意の言語でユーザプログラムを記述可能
  - 標準入出力の解析が必要
- Hadoop Pipes
  - C++ インターフェース
  - ソケット接続を確立し、スレーブノードとの通信のチャネルを生成
- JNI（Java Native Interface）
  - JVMで実行されるJavaコードが他のプログラミング言語で記述されたネイティブコードを利用するためのAPI
  - JVMで動作させるには不利なプログラムをネイティブコードに置き換えて高速化することが可能
  - 関数呼び出しのオーバーヘッド大、実行環境依存のライブラリが必要
- jCUDA（Java for CUDA）
  - GPU上のメモリの割り当て・転送のためのメソッドを提供
  - Jcublas, Jcufft, Jcudppなどのライブラリと相互運用が可能
  - 現状では明示的なメモリアラインメント不可、非同期の命令がメモリ破損を引き起こす可能性有

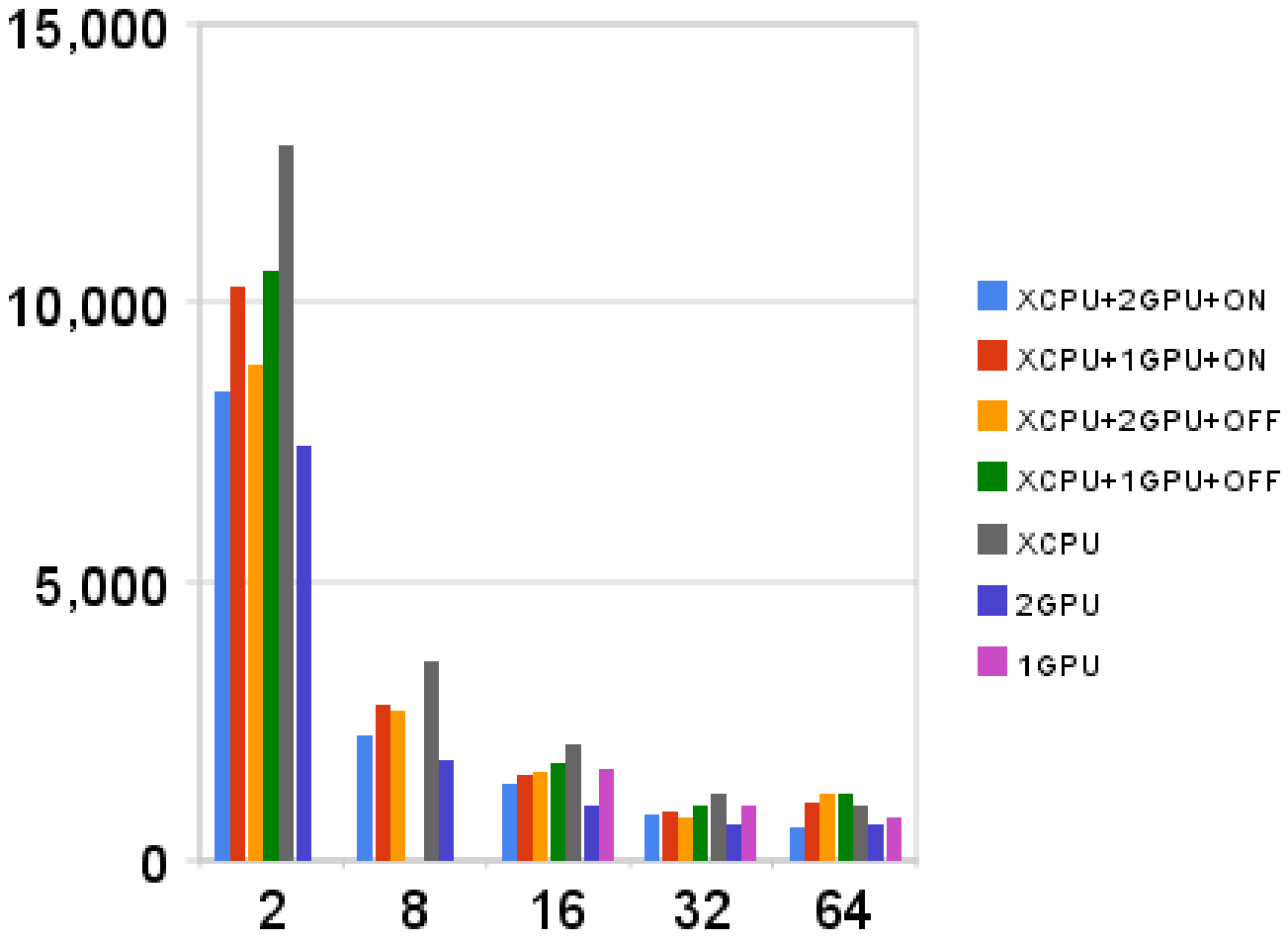
→ 今回はHadoop Pipesを使用

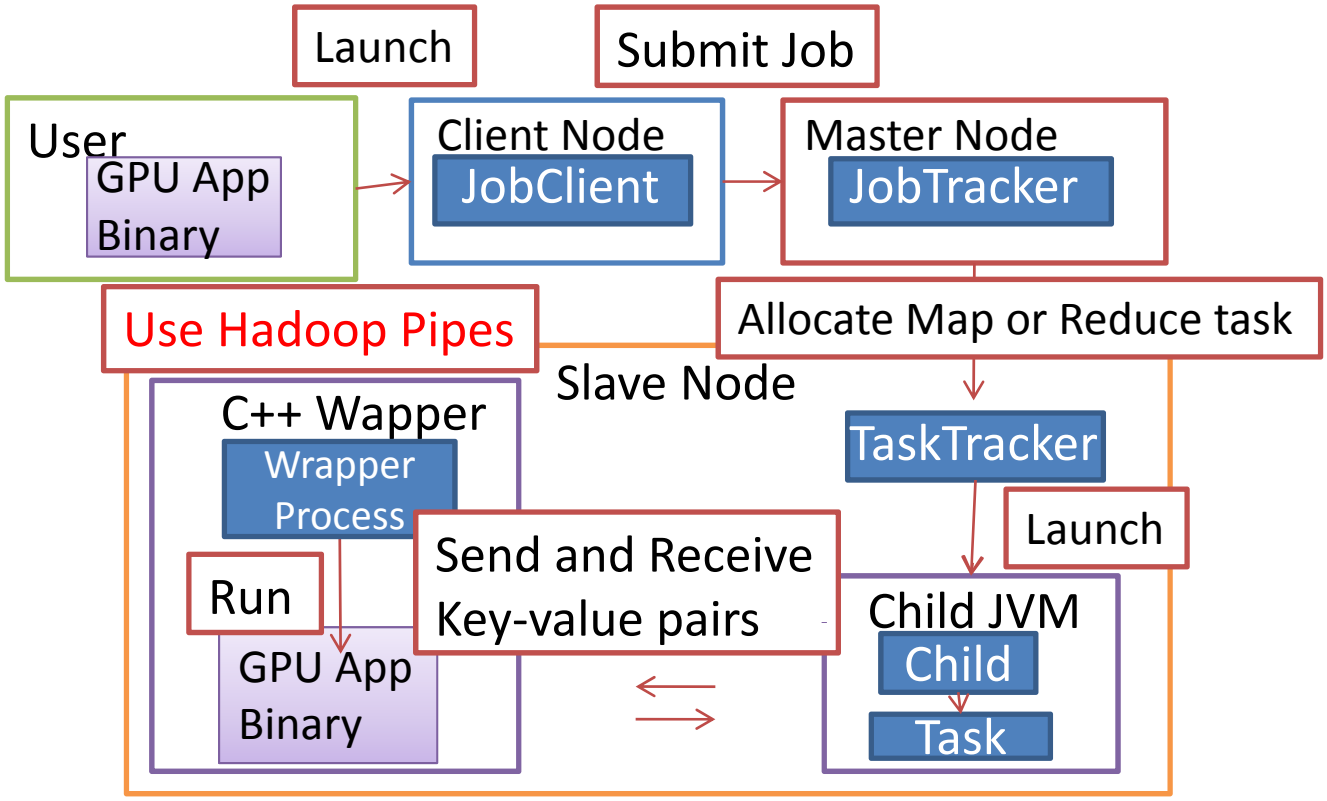

Streaming
TaskTracker
Child
Task
Child JVM
Streaming Process
Pipes
TaskTracker
Child
Task
Child JVM
C++ Wapper Library
C++ Map or Reduce

K-meansにおける Mapタスクの加速倍率



PCAでのジョブ時間


15,000
10,000
5,000
0
2
8
16
32
64
XCPU+2GPU+ON
XCPU+1GPU+ON
XCPU+2GPU+OFF
XCPU+1GPU+OFF
XCPU
2GPU
1GPU


Launch
Submit Job
User
GPU App
Binary
Client Node
JobClient
Master Node
JobTracker
Use Hadoop Pipes
Allocate Map or Reduce task
Slave Node
C++ Wapper
Wrapper
Process
TaskTracker
Launch
Run
Send and Receive
Key-value pairs
Child JVM
Child
GPU App
Binary
Task