

# 金融分野へのグリッド導入に向けた ベンチマークテスト

---

2009年3月12日

産業技術総合研究所 伊藤 智

グリッド協議会 金融分科会 ベンチマークWG

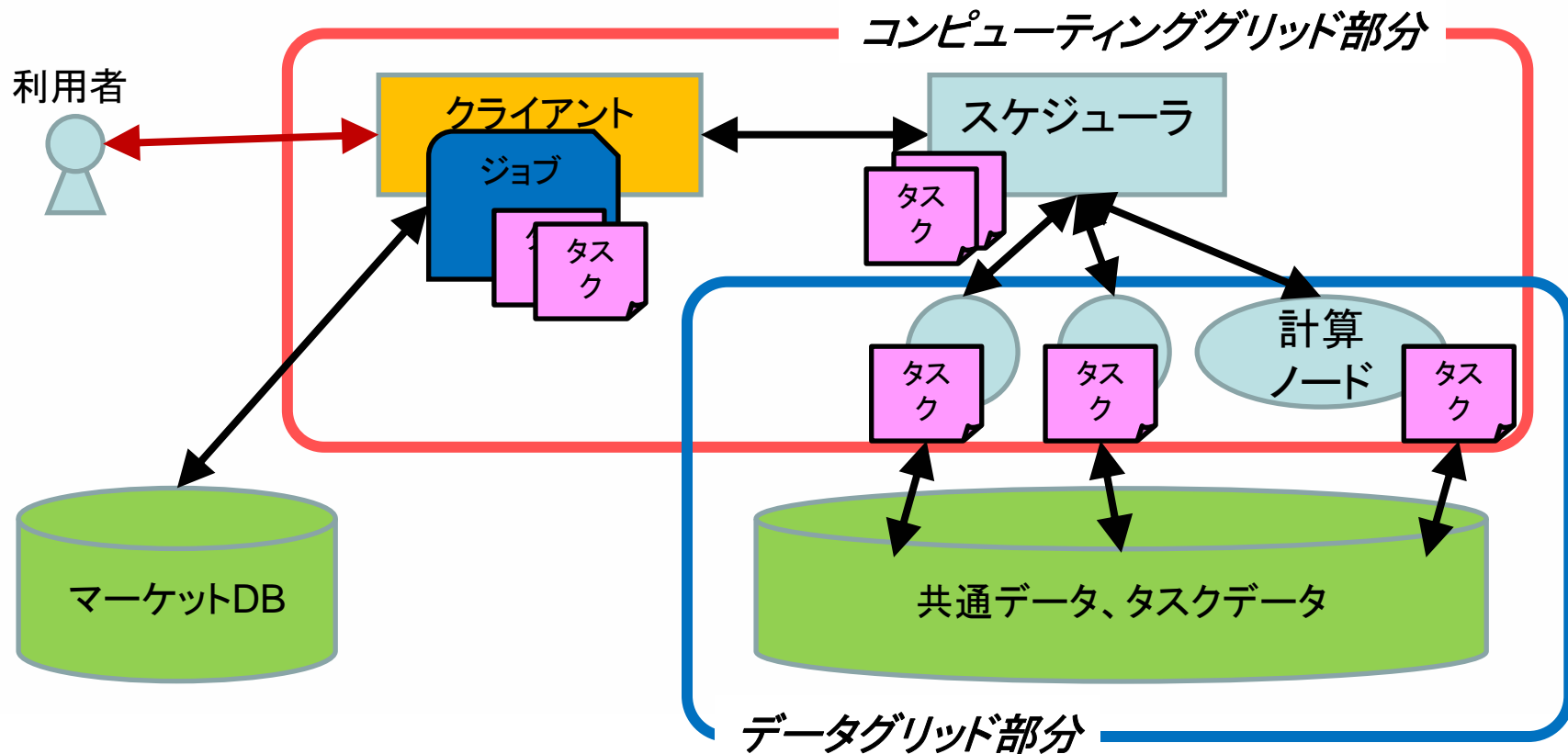
# ベンチマークの目的

---

- 典型的な金融グリッドシステムのベンチマークプログラム作成
- 実環境を用いた実測評価
  
- (分科会参加メンバーで) 共有可能な尺度を持つ
- 金融向けグリッド環境を評価
- 製品の性能を比較するのではなく、どのような使い方(パラメータの値)が適しているかを分析するツールとする
  - システムをグリッド化した際に工夫が必要なポイントを知見として得る
  - 自分の業務を当てはめるとどこに時間がかかるか、どう工夫すればいいかの指針
  - 製品のチューニング、特にデータ周りの扱いが難しいが、それらの経験が知見になること

# ベンチマークの対象業務

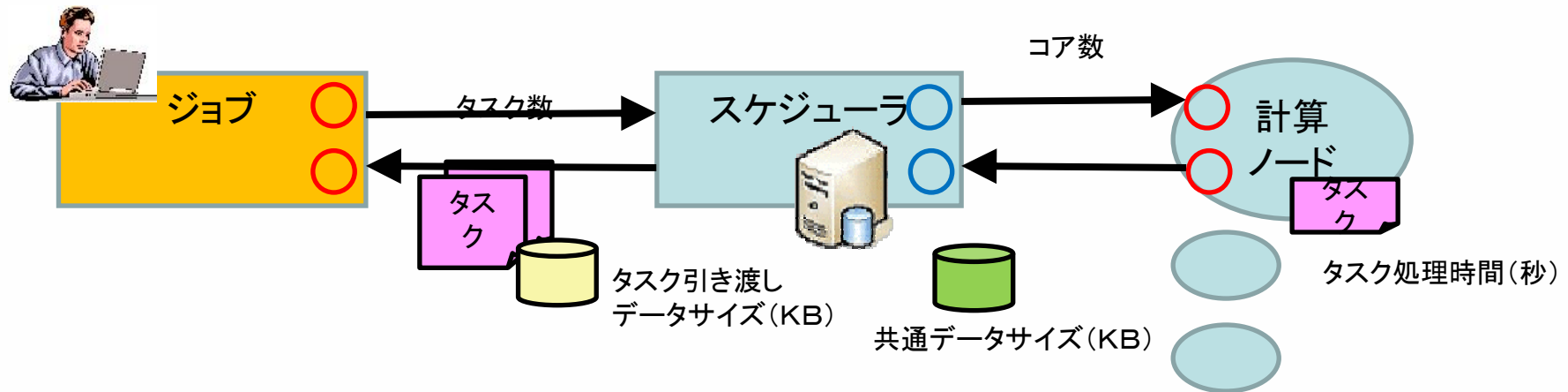
- ターゲット業務はリスク分析とする
- コンピューティンググリッド部分とデータグリッド部分に分離
- コンピューティンググリッド部分
  - タスクの配布とデータの分配・収集を含むが、データアクセスは含まない
- データグリッド部分
  - データが分散された状態で、データへのアクセスを評価する



# コンピューティンググリッド概要

- コンピューティンググリッド部分

- 一つのジョブが複数のタスクを生成し、スケジューラに対してAPIから投入
- データの分配・収集の時間も評価に含める
- 赤丸のポイントでタイムスタンプを取得(青丸は可能なら取得)



ベンチマーク評価として、可変なパラメータ

ノード数: タスク処理に使用するノード(コア)数

タスク数: 生成するタスクの数

タスク処理時間: タスク処理に要する総実行時間(秒: 平均時間と標準偏差)

タスクデータサイズ: タスクに引き渡すデータサイズ(KB)

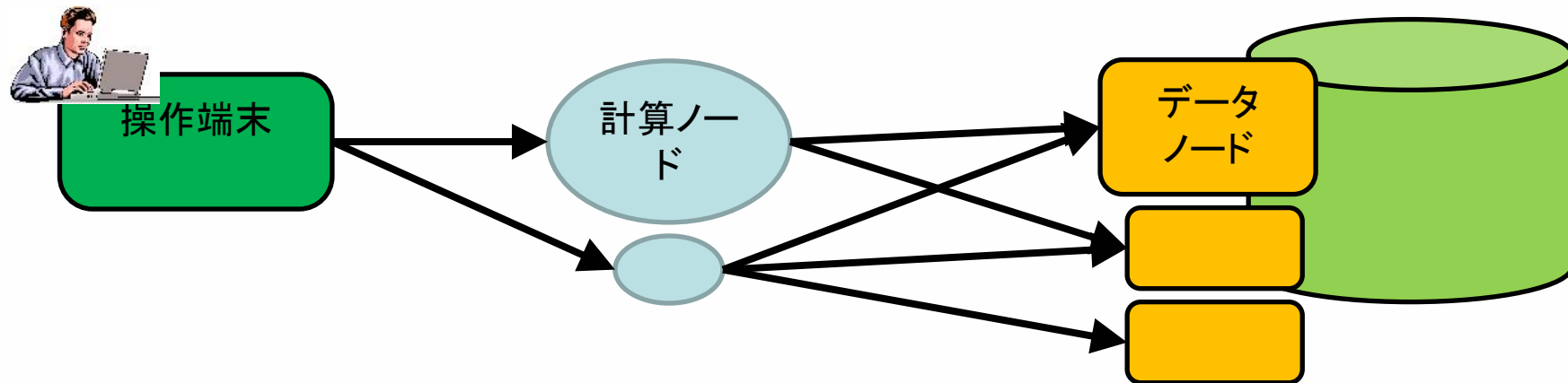
共通データサイズ: 共通データとして引き渡すデータサイズ(KB)

出力データサイズ: タスクが出力するデータサイズ(KB)

# データグリッド概要

- データグリッド部分

- 計算ノードとデータノードをそれぞれ異なるサーバで複数使用
- データノードに予め測定用データを分散配置
- 計算ノード上でデータアクセスのタスクを起動、タスクの実行時間を計測



ベンチマーク評価として、可変なパラメータ

ノード数: タスク処理に使用するノード(コア)数

データノード数: データを分散配置するノード数

タスク数: 生成するタスクの数

データサイズ: 読み/書き/更新するデータの規模

タスク処理時間: ダミーのCPU負荷時間、CPU負荷を与えないSleep時間

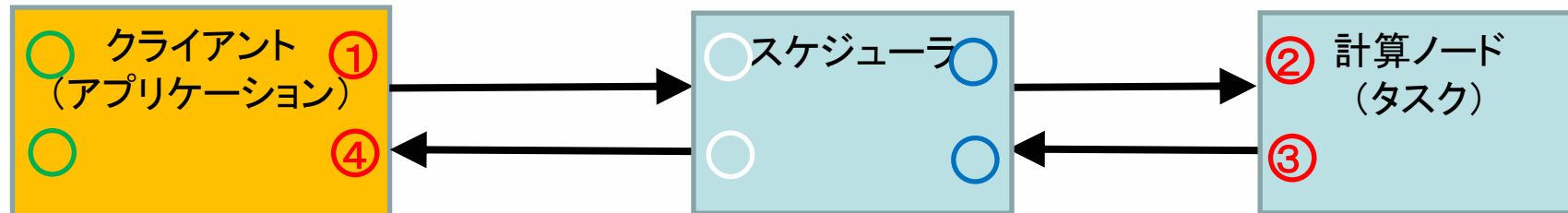
# 評価環境

---

- **実行環境 (2008.10~11月 4週間)**
  - 新日鉄ソリューションズ株式会社、  
NS Solutions Grid/Utility Computing Center (NSGUC)
  - 最大400超コア
  - CentOSによるLinux環境
- **コンピューティンググリッド**
  - Symphony (Platform Computing)
  - Condor batch & Master/Worker (University of Wisconsin)
  - Ninf-G + Sun Grid Engine (産総研) & (SUN)
  - GridServer (DataSynapse) : 未実施
  - Windows HPC Server 2008 (Microsoft) : 未実施
- **データグリッド**
  - Coherence (Oracle)
  - Caché (Intersystems)
  - WebSphere eXtream Scale/ solidDB (IBM) : 未実施
  - GigaSpaces XAP (GigaSpaces) : 未実施
  - GemFire (GenStone) : 交渉中

# コンピューティンググリッド詳細

- 以下の場所でタスク毎にタイムスタンプを取得する
- タスクには個別IDを付加する



- タイムスタンプの場所
- クライアントアプリケーション
  - データの転送開始時刻と終了時刻が取れるなら取る
  - タスクの投入時刻①、タスクの結果の受け取り時刻④
- スケジューラ
  - キュー受付時刻
  - タスク実行開始時刻、終了時刻
- 計算ノード
  - タスクの実行開始時刻②、終了時刻③

○ 必須

○ できれば

どちらかで  
OK

# コンピューティンググリッド パラメータ

- **パラメータ設定**

- ファイルから読み込み (xml)

```
<BenchmarkSetup>
```

```
  <NTasks> 10      </NTasks>
```

```
  <NNodes> 1       </NNodes>
```

```
  <InputFile> input.10.30.5 </InputFile>
```

```
  <TaskDataSize> 5   </TaskDataSize>
```

```
  <SharedDataSize> 5 </SharedDataSize>
```

```
  <WriteDataSize> 5  </WriteDataSize>
```

```
</BenchmarkSetup>
```

- NTasks 生成するタスクの数 (整数)
- NNodes タスク処理に使用するノード(コア)数 (整数)
- InputFile タスク実行時間リストをおさめたファイルの名称 (文字列)
- TaskDataSize タスクに引き渡すデータサイズ(KB) (実数)
- SharedDataSize 共通データとして引き渡すデータサイズ(KB) (実数)
- WriteDataSize タスクが出力するデータサイズ(KB) (実数)

- **タスク実行時間リスト(上記例 input.10.30.5 )**

- 改行で区切られた浮動小数点値(数値は実行に掛かる\*秒数\*)
- ただし, #で始まる行はコメントとして無視する.

- **タスク生成**

- 実行時間とダミーのデータを引き渡してタスクを呼び出す
- 指定された時間sleepして戻る

# コンピュータグリッド評価シナリオ

---

- 1) コア数と共通データサイズの関係
- 2) タスク実行時間と入出力データサイズの関係
- 3) 最初のタスクの立ち上げコスト
- 4) タスク毎データによる遅延時間の測定
- 5) タスク毎戻りデータによる遅延時間の測定
- 6) 総データ量をタスクで分割

# 1) コア数と共通データサイズの関係

---

## ○観点

- ・データ量が増える、コア数が増えると共通データの配布に負荷がかかってくるはずどの程度までが適当か
- ・データグリッドとの比較としてある程度以上のデータはデータグリッドの方がいいのではないか

## ○可変のパラメータ

ノード数: 1, 10, 50, 100, 200, 400

共通データサイズ: 0KB, 10KB, 100KB, 1MB(1,024KB), 10MB, 100MB, 1GB(1,048,576KB)

## ○固定のパラメータ

タスク数: = ノード(コア)数

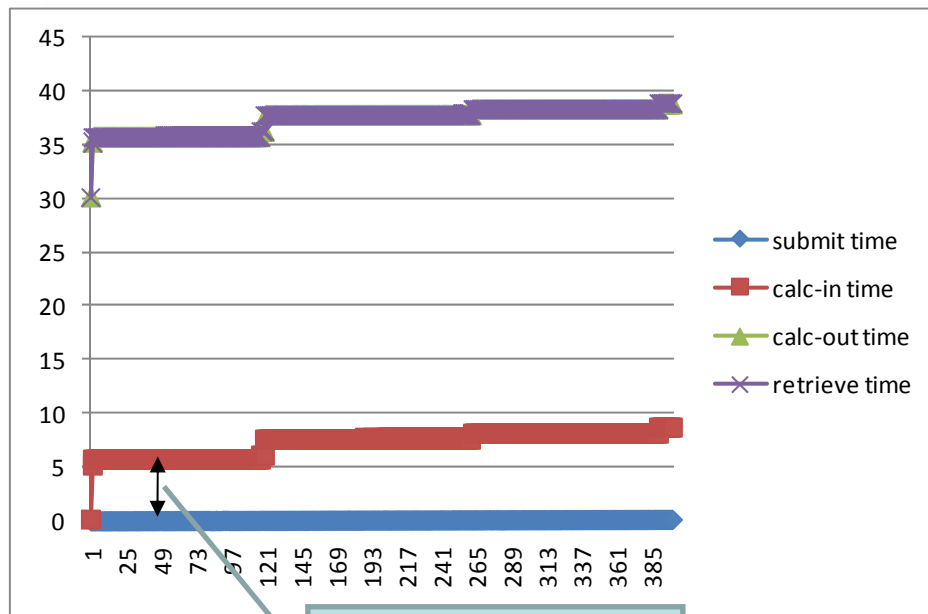
タスク時間: 30秒均一

入力データサイズ: 0KB

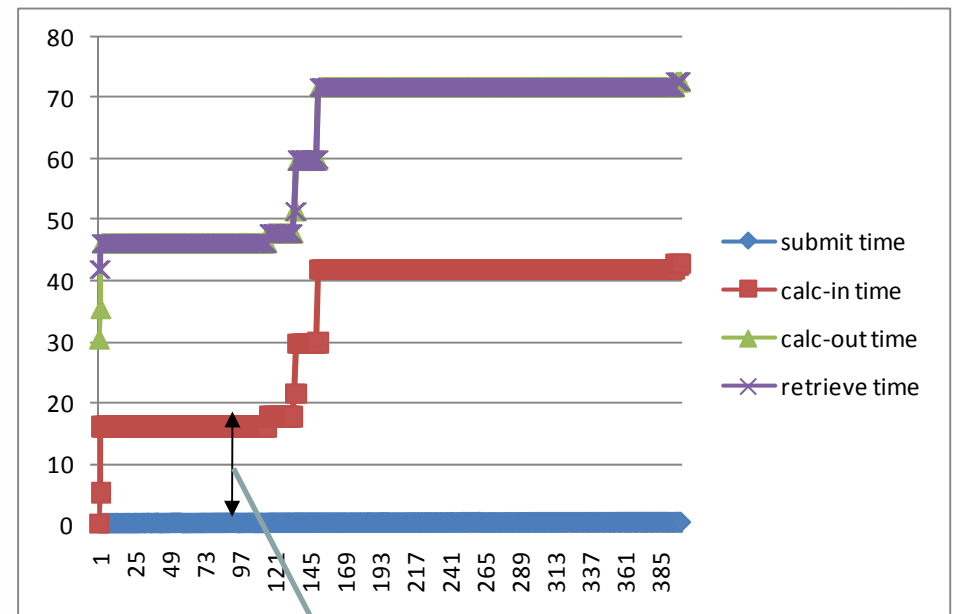
出力データサイズ: 0KB

# シナリオ1 結果例(Symphony)

- ノード(コア)数 = 400, タスク数=400, sleep=30秒, 共通データサイズ= 0KB(左), 10MB(右)
- 共有データは、青から茶色の間に送信ノード(コア)により、タイムラグが発生



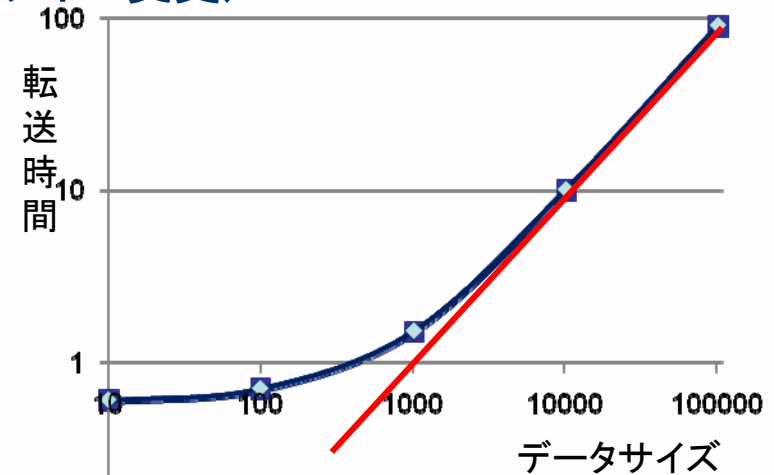
立ち上がり時間



立ち上がり時間＋  
共有データ転送時間

# シナリオ1 データ転送

- 共通データの転送時間を抽出  
(ノード数100 タスク数100 共通データサイズ変更)
- 最大転送速度1.1MB/s (8.8Mbps)
- 100個分なので 最大880Mbps  
ほぼGb Ethernet の性能
- 1MBのデータのあたりでは560Mbps  
(これは、シナリオ2の数字とほぼ一致)



データサイズ	Symphony				Ninf-G			
	計算開始	立ち上げ	転送時間	転送速度	計算開始	立ち上げ	転送時間	転送速度
0K	5.6	5	0.6		21.75	21.6	0.15	
10K	5.6	5	0.6	0.02	20	19.89	0.11	0.1
100K	5.7	5	0.7	0.14	18.8	18.5	0.3	0.3
1M	6.51	5	1.51	0.7	22.3	21.2	1.1	0.9
10M	15	5	10	1.0	32.6	23.1	9.5	1.05
100M	95	5	90	1.1	110.3	20.2	90.1	1.1

## 2) タスク実行時間と入出力データサイズの関係

---

### ○観点

- ・データサイズがタスク実行時間に比して大きすぎると、非効率になるはず  
どの程度までが適当か
- ・極端に遅くなるデータサイズ(閾値)があるのではないか

### ○可変のパラメータ

タスク時間: 0s, 0.1s, 1s, 10s, 100s

**入出力**データサイズ: 0KB, 10KB, 100KB, 1MB(1,024KB), 10MB, 100MB,  
1GB(1,048,576KB)

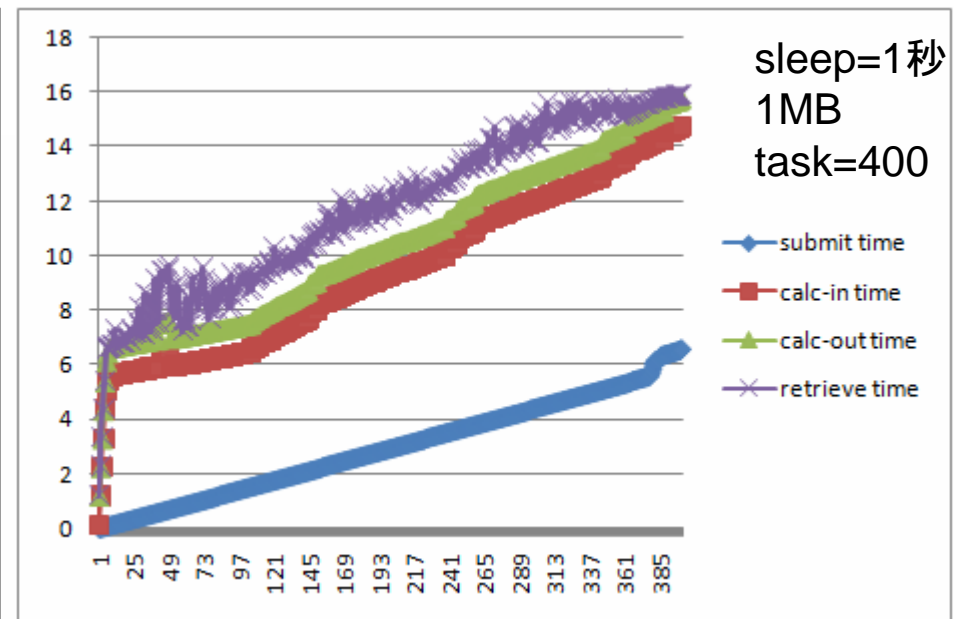
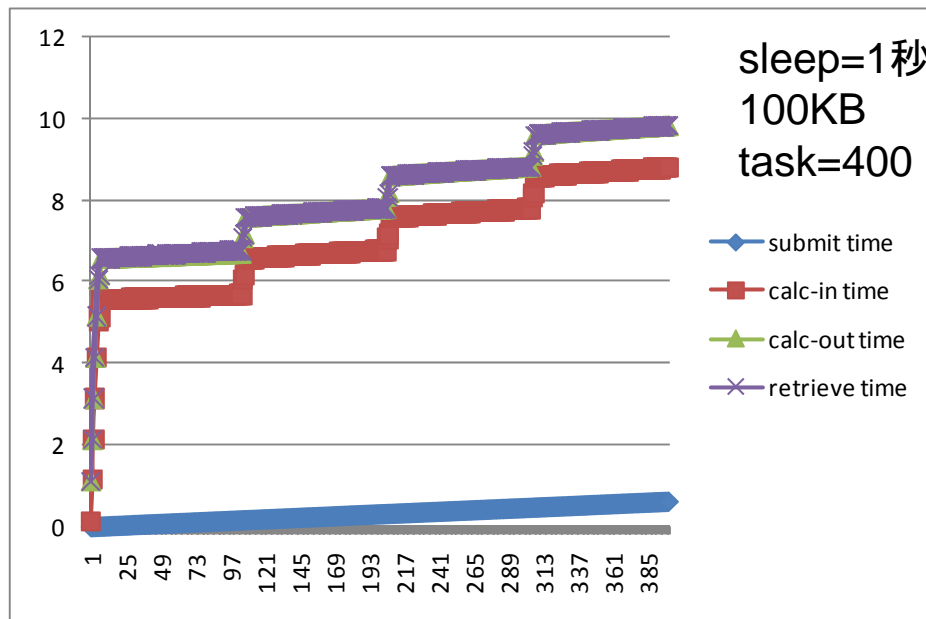
### ○固定のパラメータ

ノード数: 100

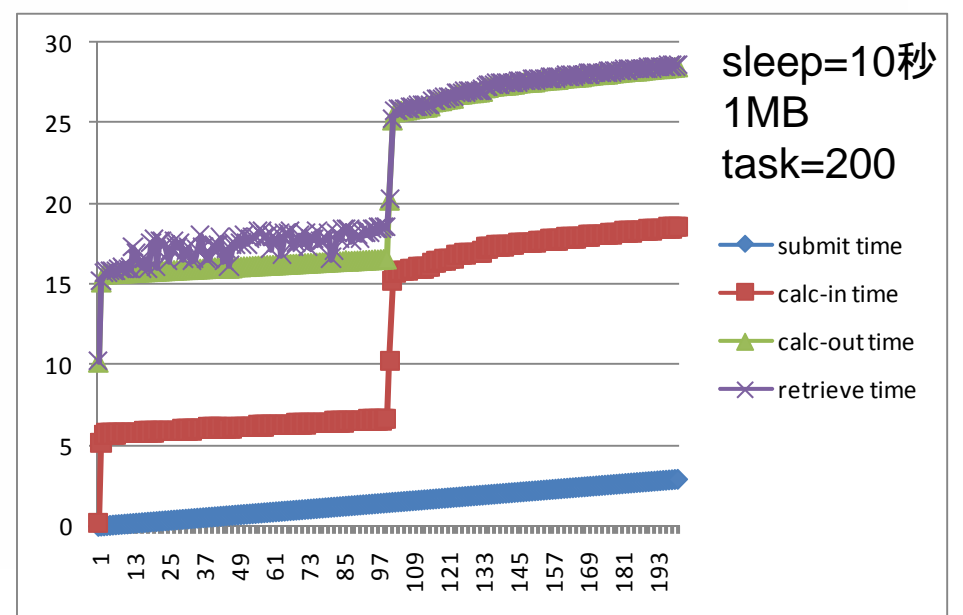
タスク数: = ノード数 (または2倍の200)

共通データサイズ: 0KB

# シナリオ2 結果例(Symphony)



- ノード(コア)数 = 100
- 処理時間にデータ転送時間が隠ぺいされれば影響少ない
- 実行時間  
≒ 立ち上げ時間 + 処理時間 + 転送時間
- $\text{コア数} \times \text{データサイズ} / \text{転送速度} < \text{Sleep}$  なら影響が少ない ようである
- グラフがギザギザの部分: Symphony のスケジューラのフロー制御によりタスク受信タイミングが前後しているため発生



# 3) 最初のタスクの立ち上げコスト

---

## ○観点

- ・各計算ノードの最初の1つ目のタスクが処理を開始するまでの時間(スケジューラ起動コスト)を見積もりたい
- ・理想的には100 までは定数時間 100以上はリニアに増加
- ・しかし、最初のタスク立ち上げのコストがあるので (Symphonyの場合)、1000は100の場合の10倍より小さいのではないか
- ・立ち上げのコストが大きく、かつ1回のタスク数が少ない場合  
コア数を制限した方が効率が良いのでは

## ○可変のパラメータ

タスク数: 1, 10, 100, 200, 300, 500, 1000

## ○固定のパラメータ

ノード数: 100

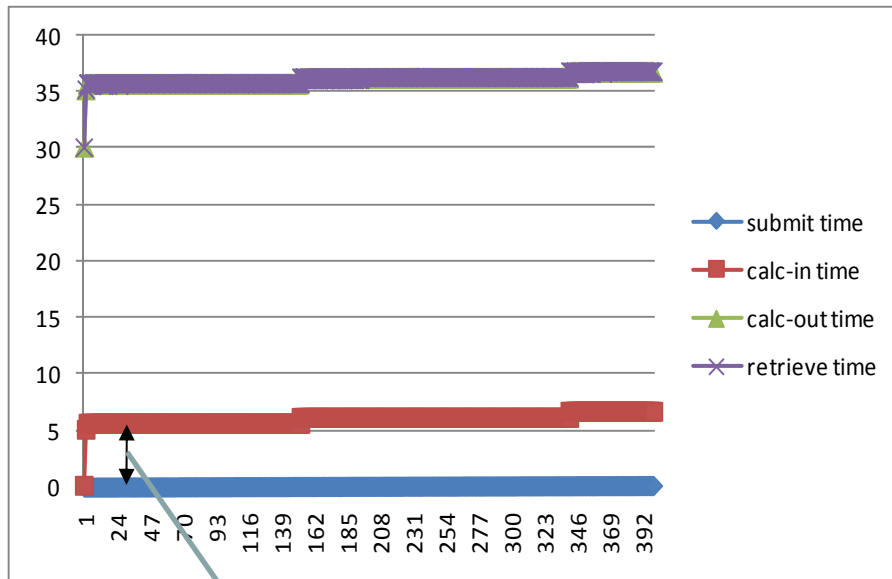
タスク時間: 30秒均一

入出力データサイズ: 0KB

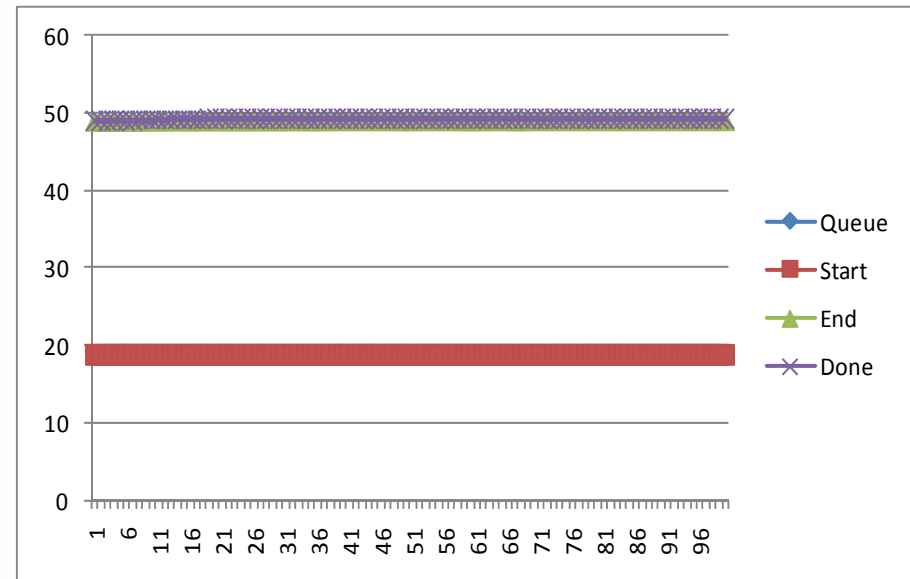
共通データサイズ: 0KB

# シナリオ3 結果例

- ノード(コア)数 = 400, タスク数=1200, sleep=30秒, Symphony (左)
- ノード(コア)数 = 100, タスク数=300, sleep=30秒, Ninf-G + SGE (右)
- 共通データサイズ、タスクデータサイズ= 0KB
- 立ち上げ時間 Symphony:5秒、Ninf-G + SGE:20秒
- Symphony は、後半やや増えているが、それほど大きくない



立ち上がり時間



## 4) タスク毎データによる遅延時間の測定

---

### ○観点

- ・クライアントでのサブミット開始からすべての結果取得までの時間
- ・転送するデータ(行き)の量でどう変化するか

### ○可変のパラメータ

入力データサイズ: 0KB, 10KB, 100KB, 1MB(1,024KB), 10MB, 100MB, 1GB(1,048,576KB)

ノード数: 100, 200, 400 (固定)

タスク数: =ノード数 × 1, × 2, × 3 (× 3あたりは時間がかかるので、優先度低)

### ○固定のパラメータ

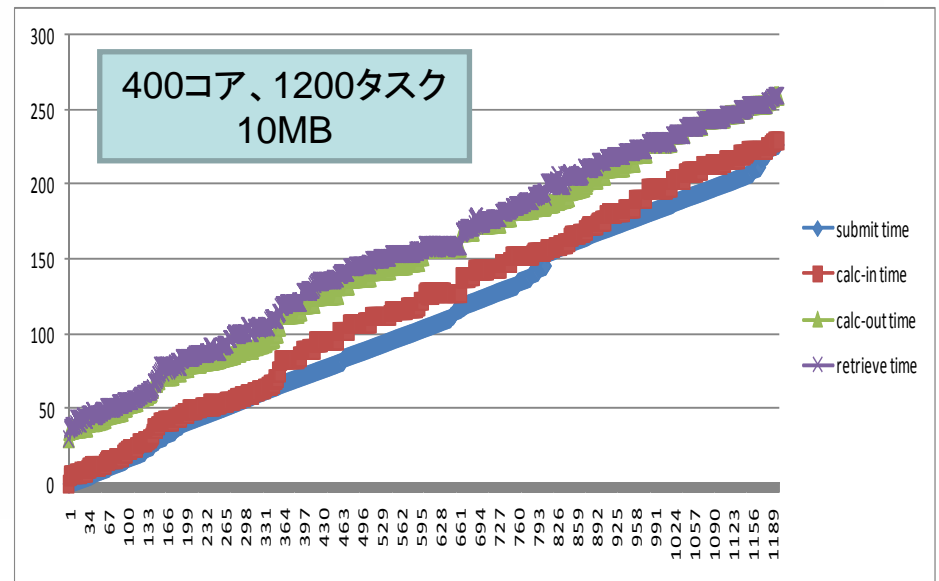
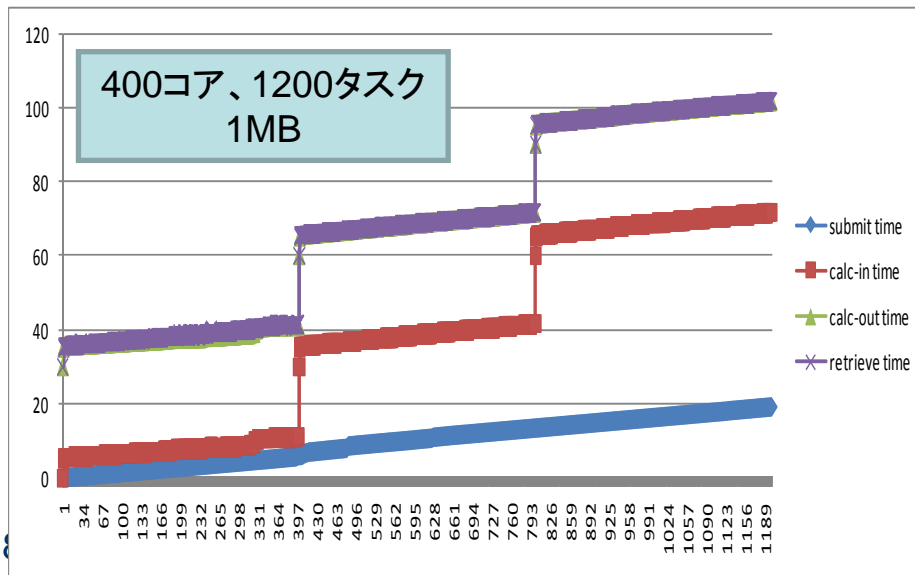
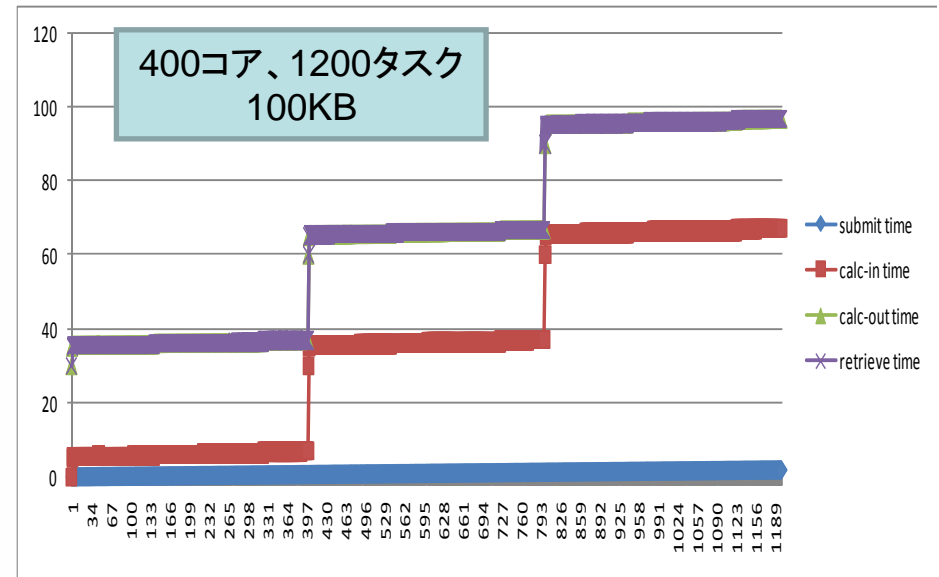
タスク時間: 30秒均一

共通データサイズ: 0KB

出力データサイズ: 0KB

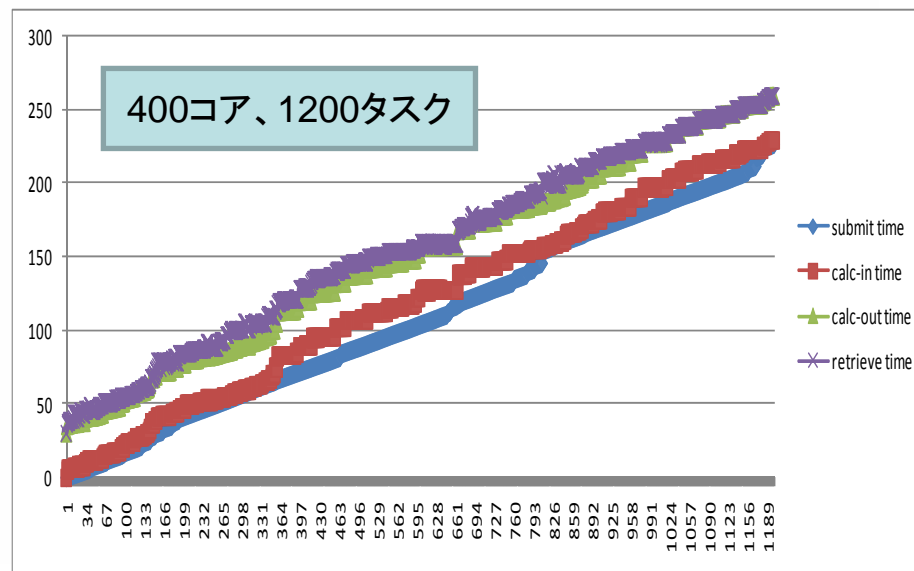
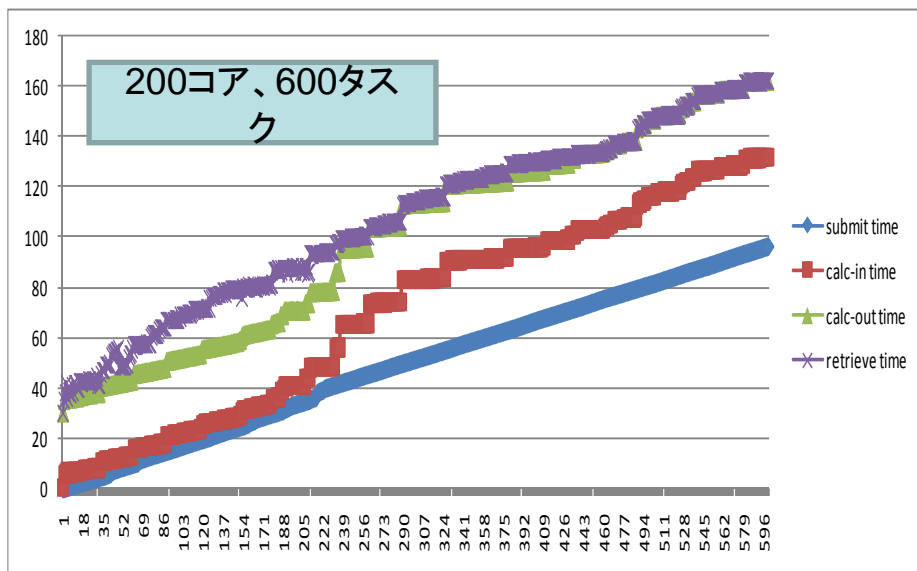
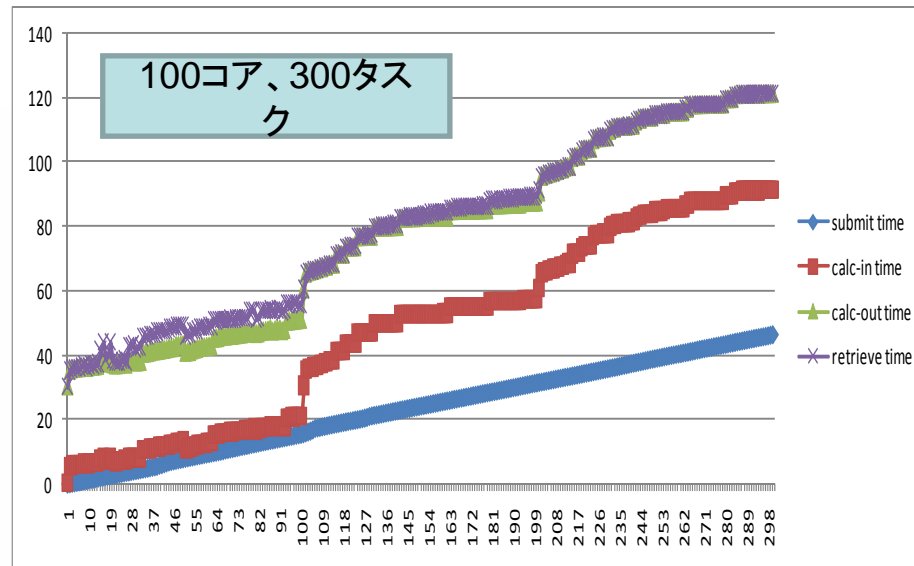
# シナリオ4 結果例 (Symphony)

- 転送するデータ(行き)の量で結果取得までの時間がどう変化するか
- 100-400コア、タスク数=コア数×3、Sleep30秒、入力データサイズ変更
- 1MBまでは影響少ない
- 10MBからは大きい
- コア数×データサイズ／転送速度 < Sleepなら影響が少ない
- 400コア×1MB/1Gbps=3.2
- 400コア×10MB/1Gbps=32



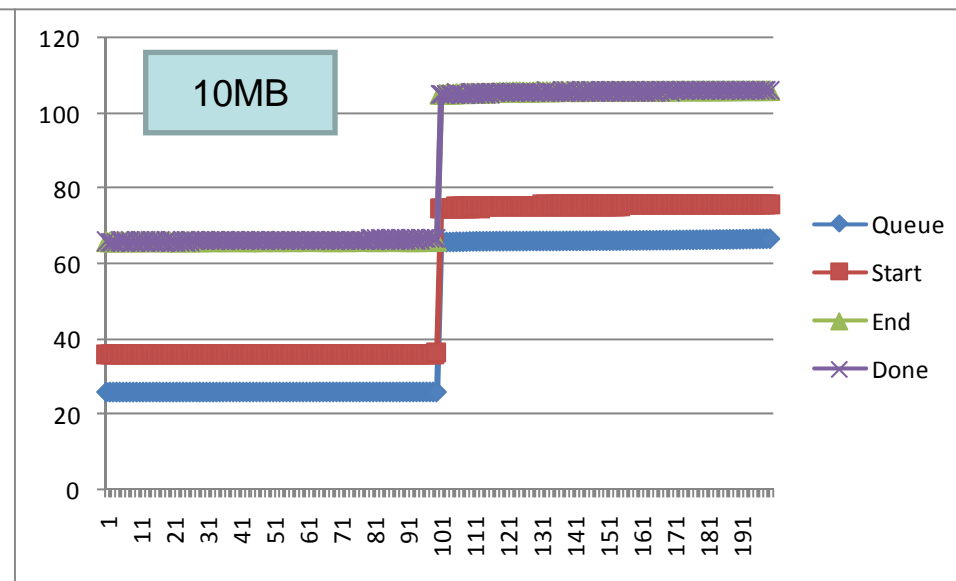
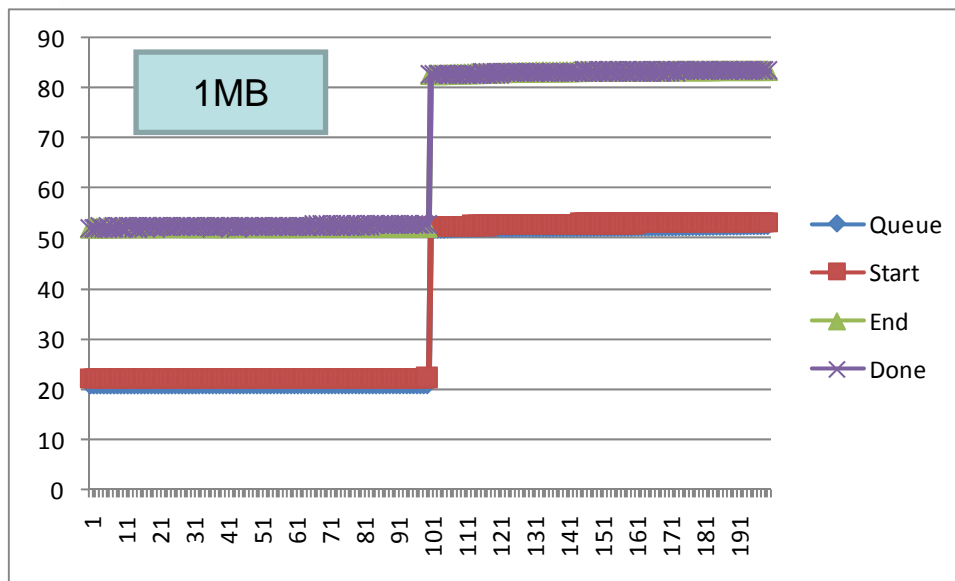
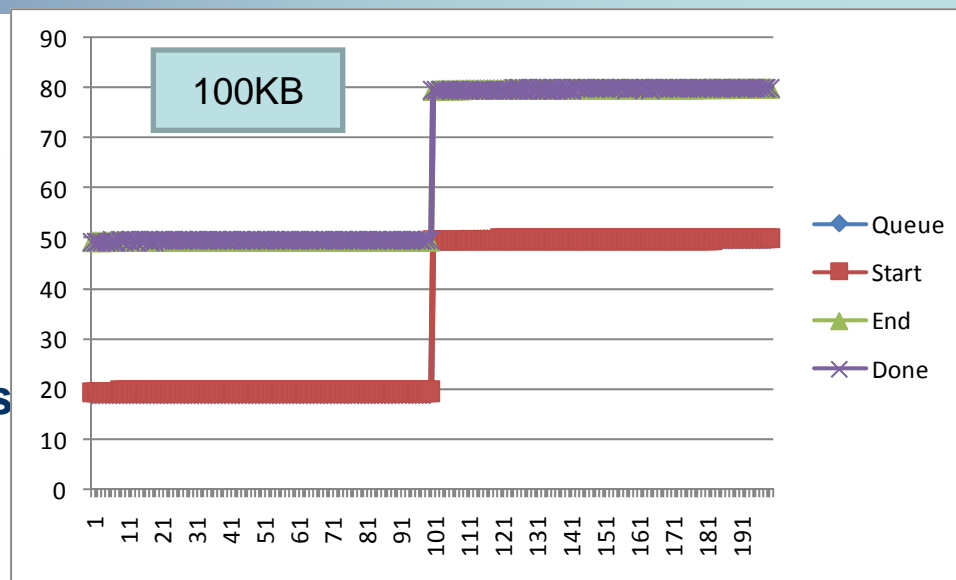
# シナリオ4 結果例 (Symphony)

- Sleep30秒、入力データ10MBの場合
- 10MB入力データで影響が出始める
- コア数×データサイズ／転送速度 < Sleepなら影響が少ない
- $100\text{コア} \times 10\text{MB} / 1\text{Gbps} = 8$
- $200\text{コア} \times 10\text{MB} / 1\text{Gbps} = 16$
- $400\text{コア} \times 10\text{MB} / 1\text{Gbps} = 32$
- 転送速度
- $10\text{MB} \times 300\text{タスク} / 46.2\text{秒} = 519\text{Mbps}$



# シナリオ4 結果例(Ninf-G)

- ノード数100 **タスク数200**
- sleep 30秒、入力データサイズ変更
- データ転送がジョブの実行に擾乱を与えないのできれいに終了する
- データ転送  
10MB × 100タスク / 9.7秒 = 825Mbps



## 5) タスク毎戻りデータによる遅延時間の測定

### ○観点

- ・クライアントでのサブミット開始からすべての結果取得までの時間
- ・転送するデータ(戻り)の量でどう変化するか

### ○可変のパラメータ

出力データサイズ: 0KB, 10KB, 100KB, 1MB(1,024KB), 10MB, 100MB, 1GB(1,048,576KB)

ノード数: 100, 200, 400 (固定)

タスク数: =ノード数 × 1, × 2, × 3 (× 3あたりは時間がかかるので、優先度低)

### ○固定のパラメータ

タスク時間: 30秒均一

共通データサイズ: 0KB

入力データサイズ: 0KB

シナリオ4と同様  
なので、割愛

## 6) 総データ量をタスクで分割

### ○観点

- ・リスク分析の場合など、タスクの粒度の最適値が、データの転送部分で見えてくるか？
- ・タスクの粒度が変わっても1タスクが読み込むデータサイズは同一とする

### ○可変のパラメータ

総タスク実行時間: 100s, 10s

(タスク数: 実行時間) = (1, 100s), (10, 10s), (25, 4s), (50, 2s), (100, 1s), (200, 0.5s),  
(400, 0.25s), (1000, 0.1s)

(タスク数: 実行時間) = (1, 10s), (10, 1s), (25, 0.4s), (50, 0.2s), (100, 0.1s), (200,  
0.05s), (400, 0.025s)

入力データサイズ: 0KB, 10KB, 100KB, 1MB(1,024KB)

### ○固定のパラメータ

ノード数: 100 (固定)

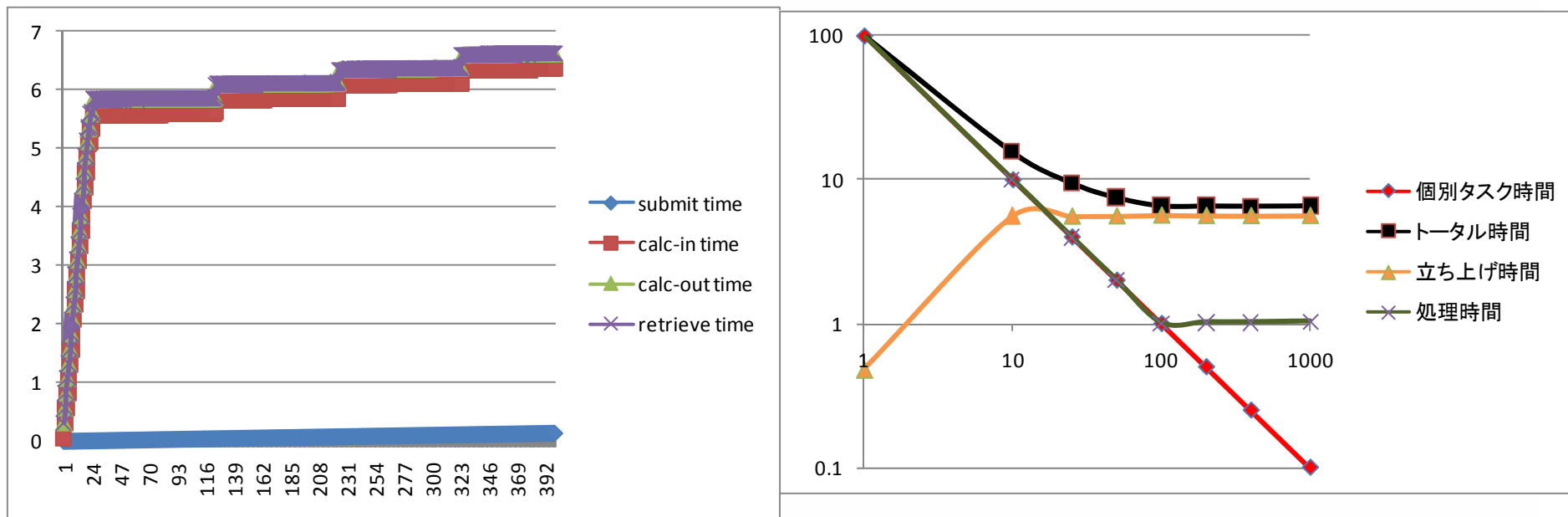
共通データサイズ: 0KB

出力データサイズ: 0KB

# シナリオ6 結果例 (Symphony)

- ノード(コア)数 = 100, トータル処理時間 = 100秒、データサイズ = 0KB
- 赤: 個別タスクの理想処理時間、黒: 実際の実行時間
- 立ち上げ時間(橙)のため、ある以上の分割ではトータルは短くならない
- 最初から全部立ち上がっている場合は、緑のようになる  
ノード(コア)数以上のタスクは処理が順次処理が行われるため、トータルの処理時間は短くならない

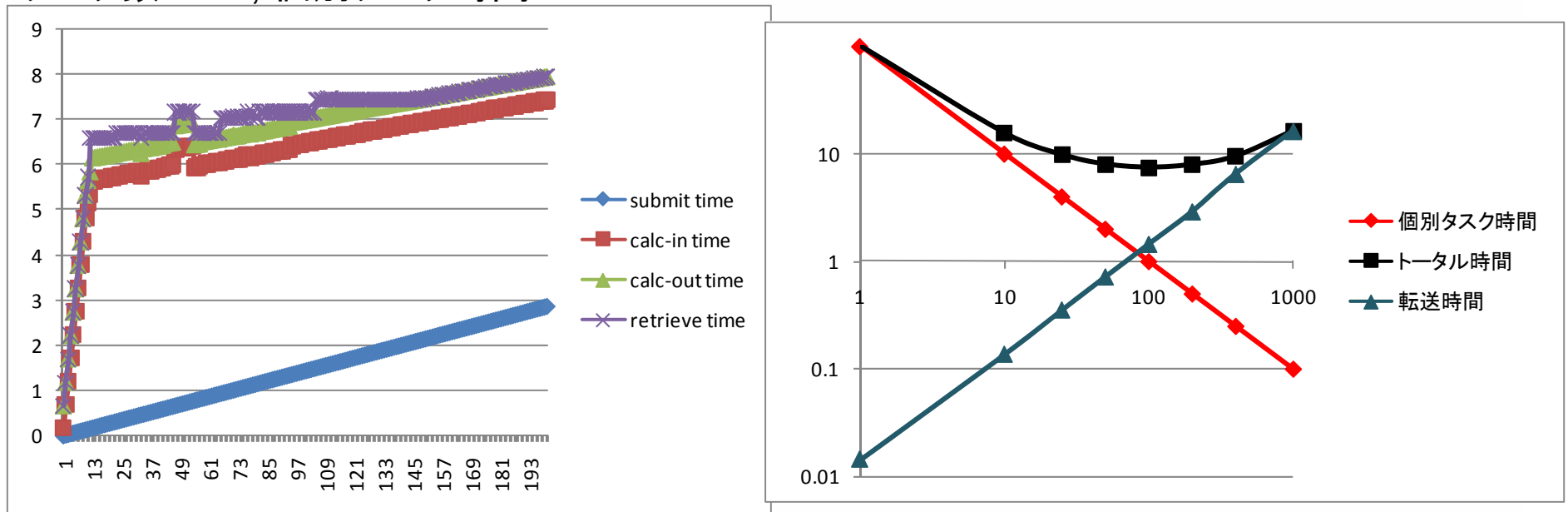
タスク数=400, 個別タスク時間=0.25



# シナリオ6 結果例 (Symphony)

- ノード(コア)数=100, トータル処理時間=100秒、データサイズ=1MB
- 青: 個別タスクの理想処理時間、赤: 実際の実行時間
- 個別タスクの処理時間短縮に対して、タスクが増えるためデータ転送時間が増加。クロスする100タスク付近まではトータル時間の短縮が見られるが、それ以上のタスク数の場合は処理時間が長くなる
- どこでクロスするかは、データサイズとトータル処理時間に依存

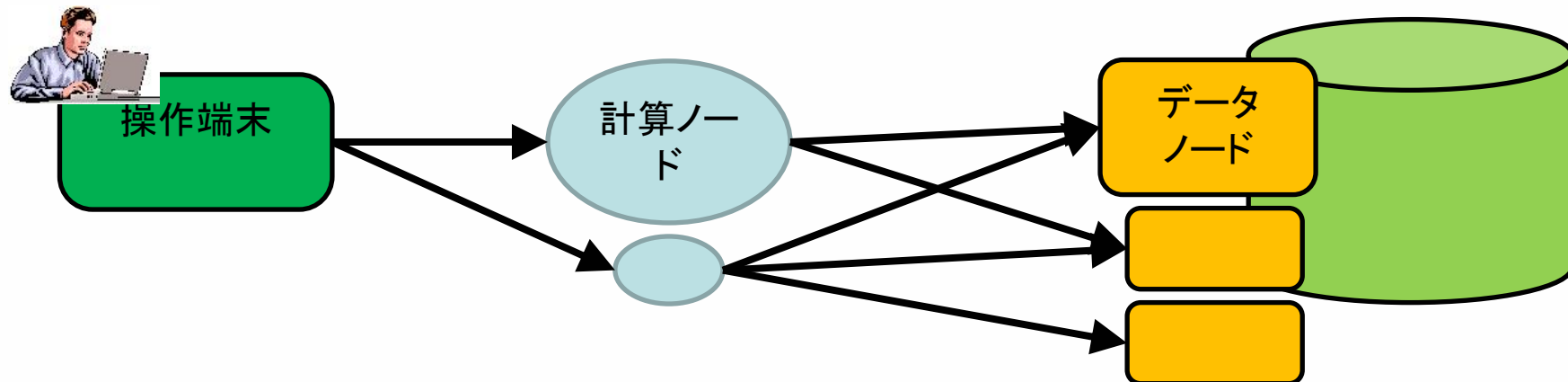
タスク数=200, 個別タスク時間=0.5



# データグリッド概要

- データグリッド部分

- 計算ノードとデータノードをそれぞれ異なるサーバで複数使用
- データノードに予め測定用データを分散配置
- 計算ノード上でデータアクセスのタスクを起動、タスクの実行時間を計測



ベンチマーク評価として、可変なパラメータ

ノード数: タスク処理に使用するノード(コア)数

データノード数: データを分散配置するノード数

タスク数: 生成するタスクの数

データサイズ: 読み/書き/更新するデータの規模

タスク処理時間: ダミーのCPU負荷時間、CPU負荷を与えないSleep時間

# データグリッドベンチマーク詳細

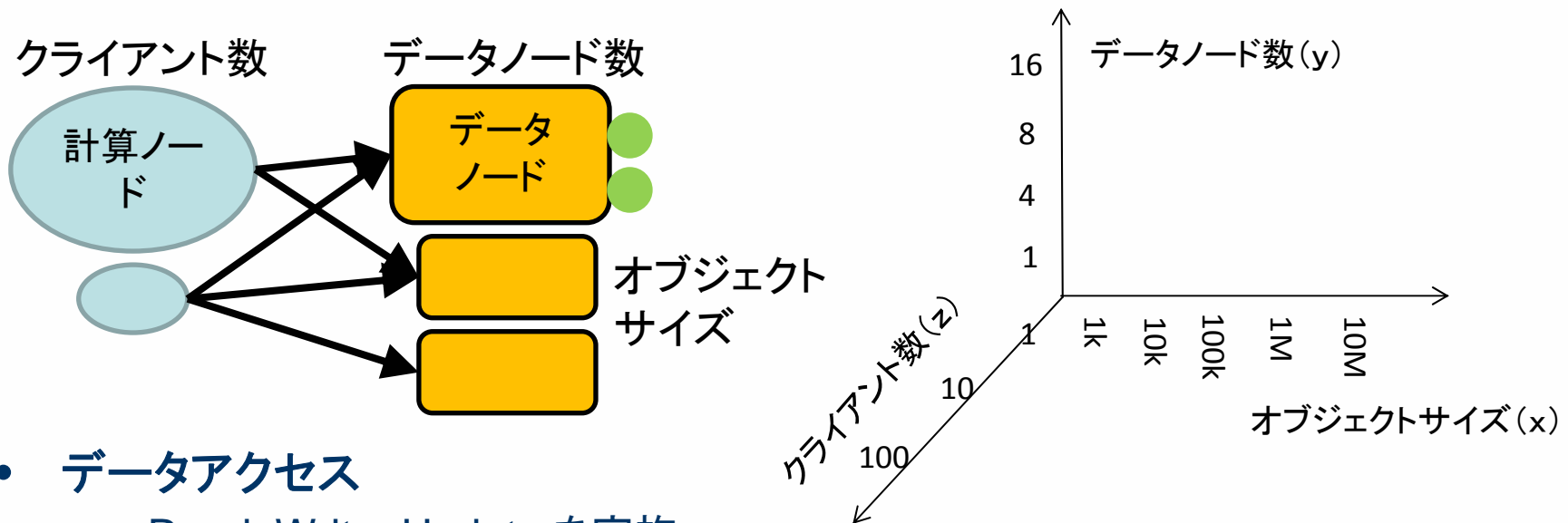
- **NNodes** : クライアントノード数
  - タスクを処理するノード数
- **MNodes** : データサーバ数
  - データを分散保持するサーバ数
- **NTasks** : タスク数
  - アプリケーションが発生させるタスクの数
- **NRepeats** : 繰り返し回数
  - タスク内にあるDoループの繰り返す回数
- **ReadDataSize** : object数
  - 一つのタスクがDo ループの中で読み込むデータオブジェクトの数
- **UpdateDataSize** : object数
  - 一つのタスクがDo ループの中で更新するデータオブジェクトの数
- **WriteDataSize** : object数
  - 一つのタスクがDo ループの中で書き込むデータオブジェクトの数
- **NObjects** : オブジェクト数
  - タスクが読み込みおよび更新するデータのオブジェクト数
- **ObjectSize** : オブジェクトサイズ
  - key をのぞいたダミーデータのサイズ (KB)
- **SharedDataSize** : 共有データサイズ
  - 全てのタスクが読み込むデータのオブジェクト数
- **ProcessTime** : ダミー処理時間
  - タスク内でCPUに負荷を与えるための処理時間で、乱数発生(java 関数名)を呼び出す回数で指定
- **SleepTime** : Sleep時間
  - タスク内でCPU負荷を与えずに時間だけを経過させる秒数

# タスクプログラムのコア部分

- タスクのIDが  $it$  ( $0 \sim N\text{Tasks}-1$ ) とします
- **Do  $is=0, \text{SharedDataSize}-1$** 
  - Read 共通データ  $\text{key}=is + N\text{Objects}$
- **End**
- **Do  $ip=0, N\text{Repeats}-1$**
- **Do  $ir=0, \text{ReadDataSize}-1$** 
  - Read 個別データ  $\text{key} = \text{MOD}((it * N\text{Repeats} + ip) * (\text{ReadDataSize} + \text{UpdateDataSize}) + ir, N\text{Objects})$
- **End**
- **Do  $ipt=0, \text{ProcessTime}-1$** 
  - Java 乱数生成の呼び出し
- **End**
- **Sleep (SleepTime)**
- **Do  $iu=0, \text{UpdateDataSize}-1$** 
  - Update 個別データ  $\text{key} = \text{MOD}((it * N\text{Repeats} + ip) * (\text{ReadDataSize} + \text{UpdateDataSize}) + \text{ReadDataSize} + iu, N\text{Objects})$
- **End**
- **Do  $iw=0, \text{WriteDataSize}-1$** 
  - Insert 個別データ  $\text{key} = N\text{Objects} + \text{SharedDataSize} + (it * N\text{Repeats} + ip) * \text{WriteDataSize} + iw$
- **End**
- **End**

# データグリッド事前テスト評価シナリオ

- テスト案(各可変軸で変えてみる) **今回は予備実験として実施**
  1. データノード8固定、クライアント100固定、**オブジェクトサイズ**(1KB, 100KB, 1M, 10M), R/U/W=100
  2. データノード8固定、**クライアント**(1, 10, 100)、1オブジェクト=1KB, R/U/W=1000
  3. **データノード**(1, 4, 8)、クライアント100固定、1オブジェクト=1KB, R/U/W=1000

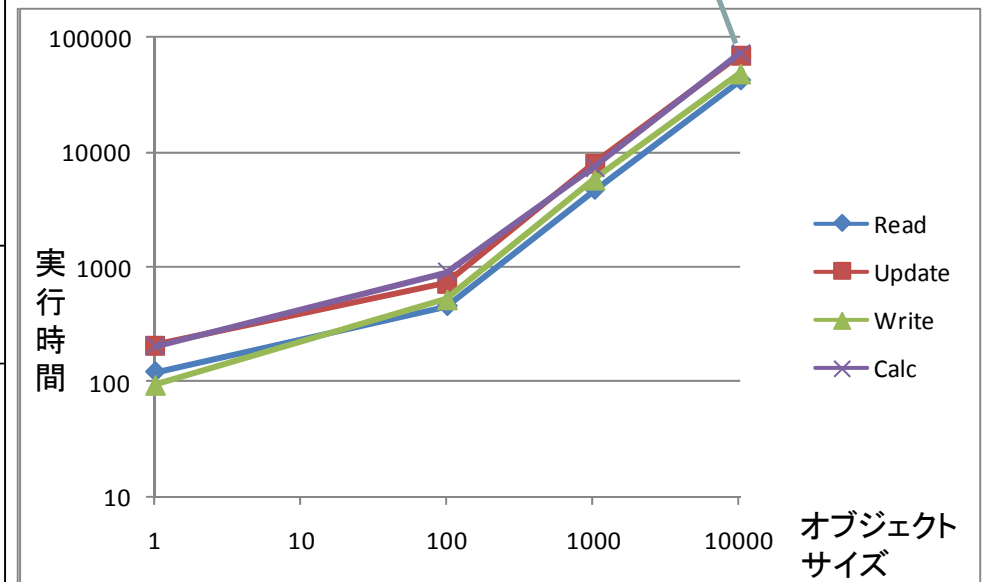
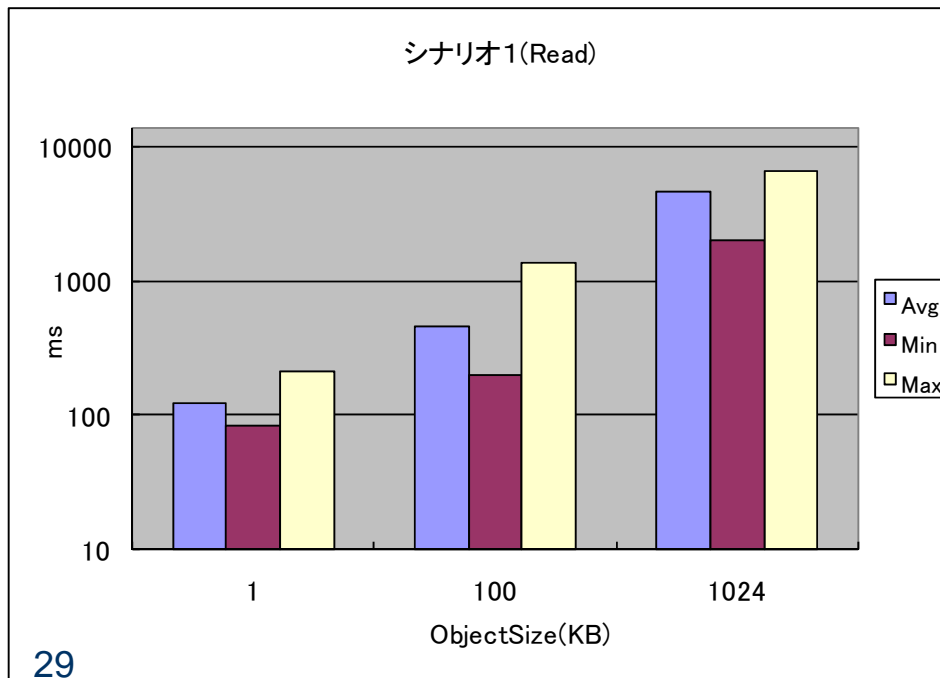


- **データアクセス**
  - Read, Write, Update を実施  
Update は、Read と Write を行うため処理時間がかかる
- **複数回実施**
  - 起動直後の試行も含むため、mix/maxの差が大きい

# 評価シナリオ1 結果例(Coherence)

- データノード8固定、クライアント10固定、オブジェクトサイズ(1KB, 100KB, 1M, 10M), R/U/W=100
- オブジェクトサイズの大きい領域ではデータ転送とアクセス時間がほとんどを占める
- オブジェクトサイズの小さい領域では、データへのアクセスにかかる処理時間に近づく(Update 1回で2ミリ秒程度)
- 例えばUpdateの場合 以下のように見ることができそう  
実行時間 =  $7.2\text{秒}/1\text{MB} \times \text{データサイズ} + 0.2\text{秒}$

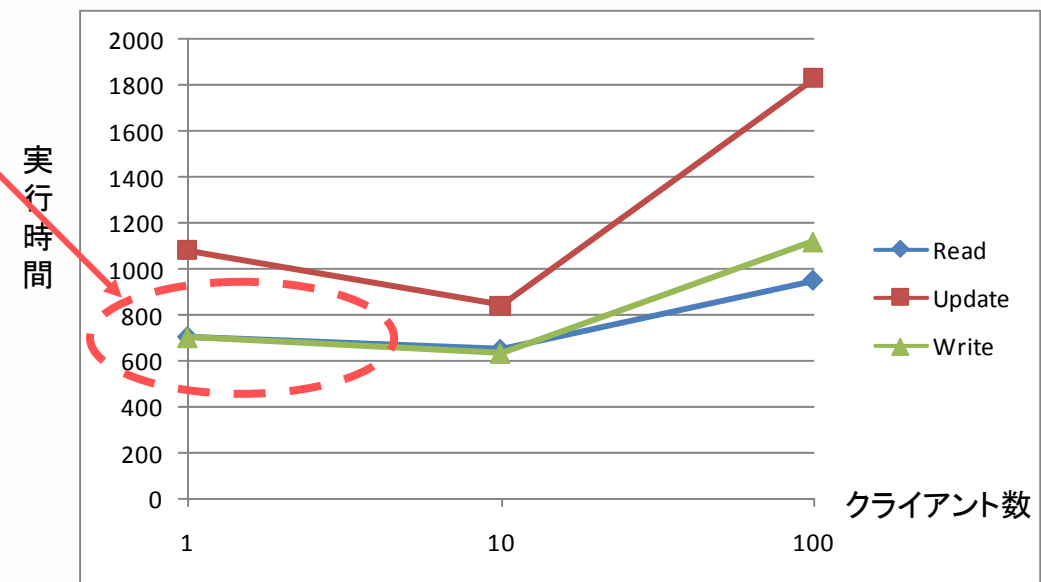
トータル  
10GB



## 評価シナリオ2 結果例 (Coherence)

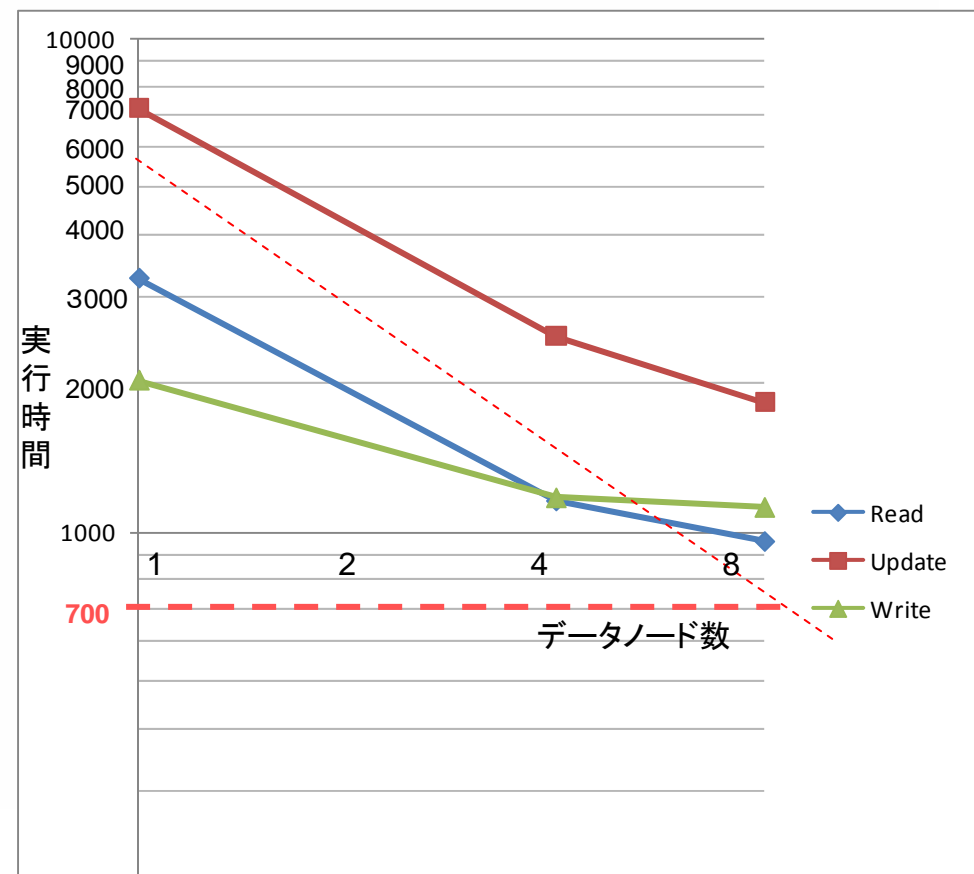
- データノード8固定、クライアント(1, 10, 100)、1オブジェクト=1KB,R/U/W=1000
- クライアント数10程度はアクセス量として負荷が多くないもよう
- クライアント数が1と10の場合では、統計的な誤差の範囲内の違いと見られる
- クライアント数100程度となると、アクセスに競合が生じ、実行時間が長くなる

一番負荷の少ない状態で、  
実行時間は約700ミリ秒



# 評価シナリオ3 結果例 (Coherence)

- データノード(1, 4, 8)、クライアント100固定、1オブジェクト=1KB, R/U/W=1000
- データノードが1つの場合、複数のクライアントからのアクセスが集中・競合し、処理時間が長い。
- データノード数を増やしていくと、一つのデータノードへの負荷が小さくなり、処理時間が短縮される。
- これは、スループットの測定ではないので、台数を増やして性能がスケールすることを見るものではない。
- データノード数を増やしていくと、負荷が分散した結果、無負荷の状態でアクセスする際の性能(前の結果から約700ミリ秒)に近づいて行く



# まとめ:ベンチマーク結果概要

## コンピューティンググリッド

- スケジューラがデータ配布の要となる場合、データ転送はネットワーク速度とデータ量に依存
- データ転送が律速になる条件は  
ノード(コア)数 × データサイズ / 転送速度 < タスク処理時間
- ノード(コア)のスケジューラ立ち上げ時間はソリューション / パラメータ設定依存 (5秒から20秒)

## データグリッド

- 今回の測定は予備実験として実施。スループットの測定は行っていない。
- 複数データノードに分散することで、1筐体の物理メモリ以上のデータを保持可能
  - 今回の測定では最大で10GBのデータをデータグリッドに格納
- データアクセスの時間はかなりばらつく (最小と最大で10倍近く開く場合もある)
  - 起動直後の試行も含んでいることの影響が考えられる
  - 次回の本計測で計測方式を改善する予定
- データノード数を増やしていくと、アクセス実行時間の短縮は見られるが、負荷が分散した結果、無負荷の状態アクセスする際の性能に近づいて行く
  - スループットの評価は次回の本計測で実施予定
- タスク数が100程度まで増えても、データノードを分散すれば、大きなボトルネックにならずにアクセス可能