

SWoPP2005 HPC-3 (科学技術計算)

並列分枝限定法における 耐故障アルゴリズムの評価

川上 健太*, 合田憲人†

* 東京工業大学大学院総合理工学研究科物理情報システム創造専攻

† 東京工業大学大学院総合理工学研究科物理情報システム専攻

1. 研究背景

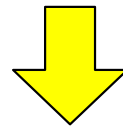
- **グリッド計算環境の特性**
 - 多数の計算資源が混在しているため、故障に脆弱
- **既存の並列アプリケーション**
 - 故障が発生すると全体の処理が停止するものがある
- **耐故障機構の必要性**
 - 耐故障性をアプリケーションで実現する事が必要

1000台のマシンで計算した時に、
全てのマシンが正常とは限らない!!

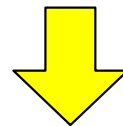


2. 研究目的

耐故障性を実現すると実行時間が増大してしまう



- ・各マシンにおける**実行時間の解析**
 - ・**実行時間を削減**させるための新たな機構
- 】が必要

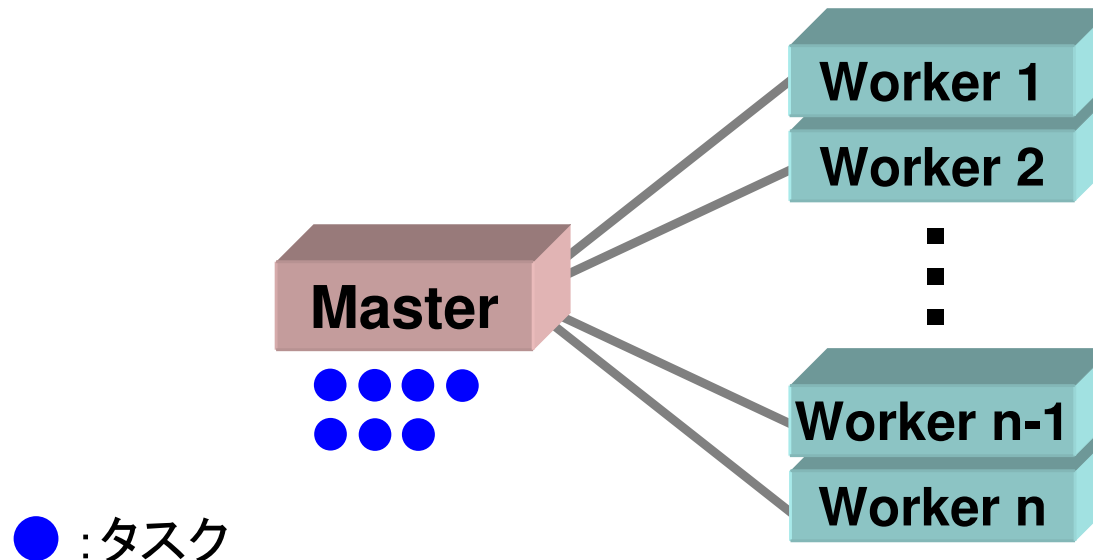


【研究目的】

- ・**実行時間を考慮した耐故障アルゴリズムの実装**
- ・各アルゴリズムの**理論的な解析**

3. 対象とする計算モデル

- アプリケーション: 並列分枝限定法
- 計算モデル: マスタ・ワーカ型モデル
 - マスタ: タスクプールを管理し, ワーカにタスクを投げる(1台のみ)
 - ワーカ: マスタから与えられたタスクを計算する(残り全て)



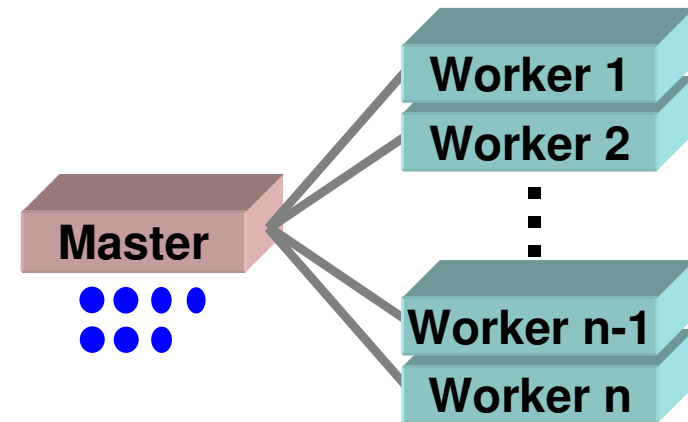
4. 関連研究

- タスクに優先度を付けて、優先度の高い処理を多重化させることで障害に備える。

耐故障/耐高負荷を考慮した並列分枝限定法と基本性能の評価 (SAC SIS 2004, P85-92)
久保田 和人, 仲瀬 明彦

| | | | | | |
|-------|-----|-----|-----|-----|----|
| タスク番号 | 8 | 9 | 5 | 6 | 7 |
| 優先度 | 300 | 250 | 200 | 100 | 80 |
| 多重度 | 3 | 2 | 1 | 1 | 1 |

優先度はタスク毎に決定

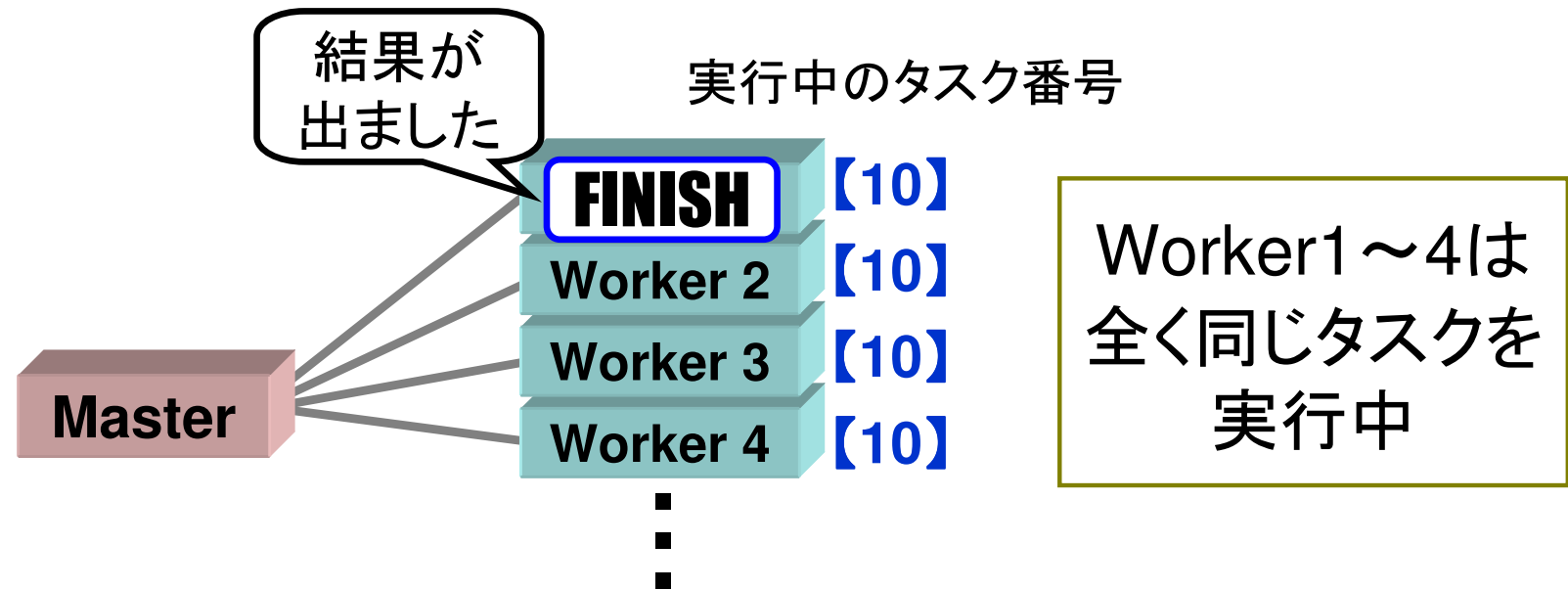


優先度が最高のタスクは3つのワーカで実行される。

【問題点】 多重度を大きくした場合に冗長な処理が増加し、実行時間が増大してしまう。

5. 提案手法 ～タスクの多重化～ (1)

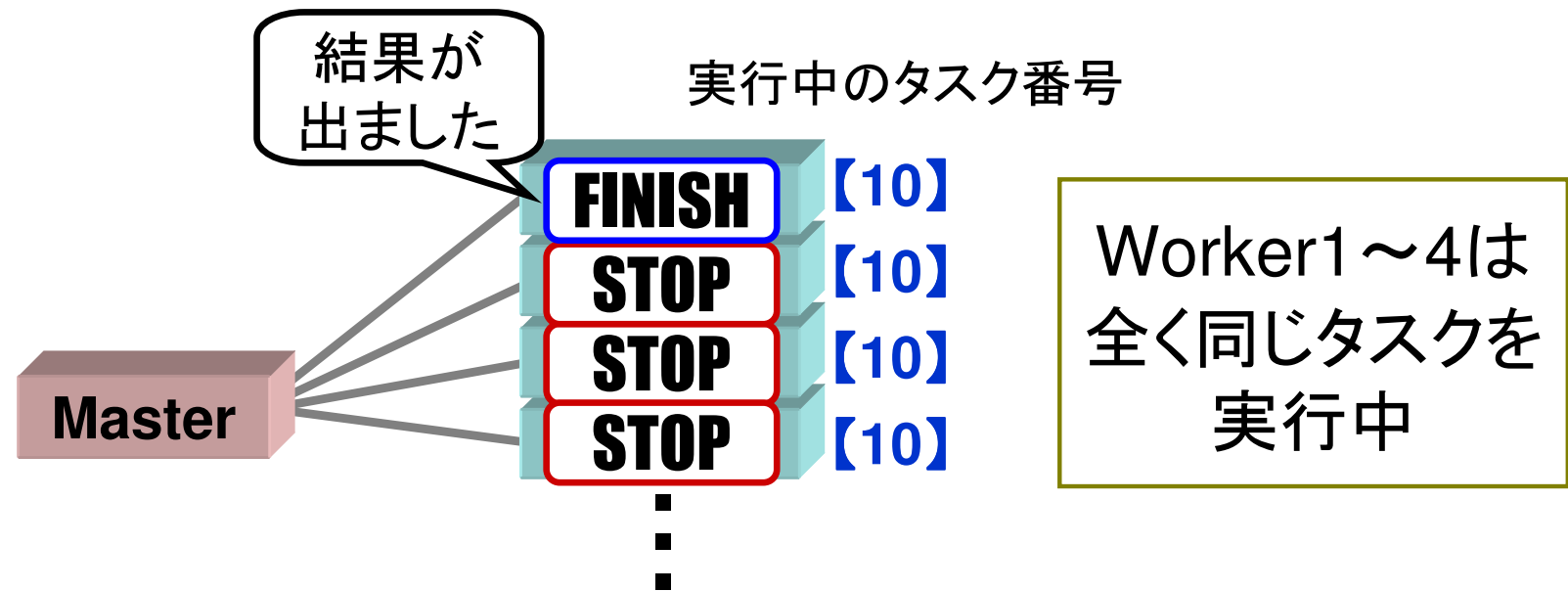
- タスクの多重化による実行時間のオーバヘッドを削減するために、**ワーカの不要な処理の中断機構**を実装した。



この時点で、同じタスクを実行している他のワーカの処理は不要となる。

5. 提案手法 ～タスクの多重化～ (1)

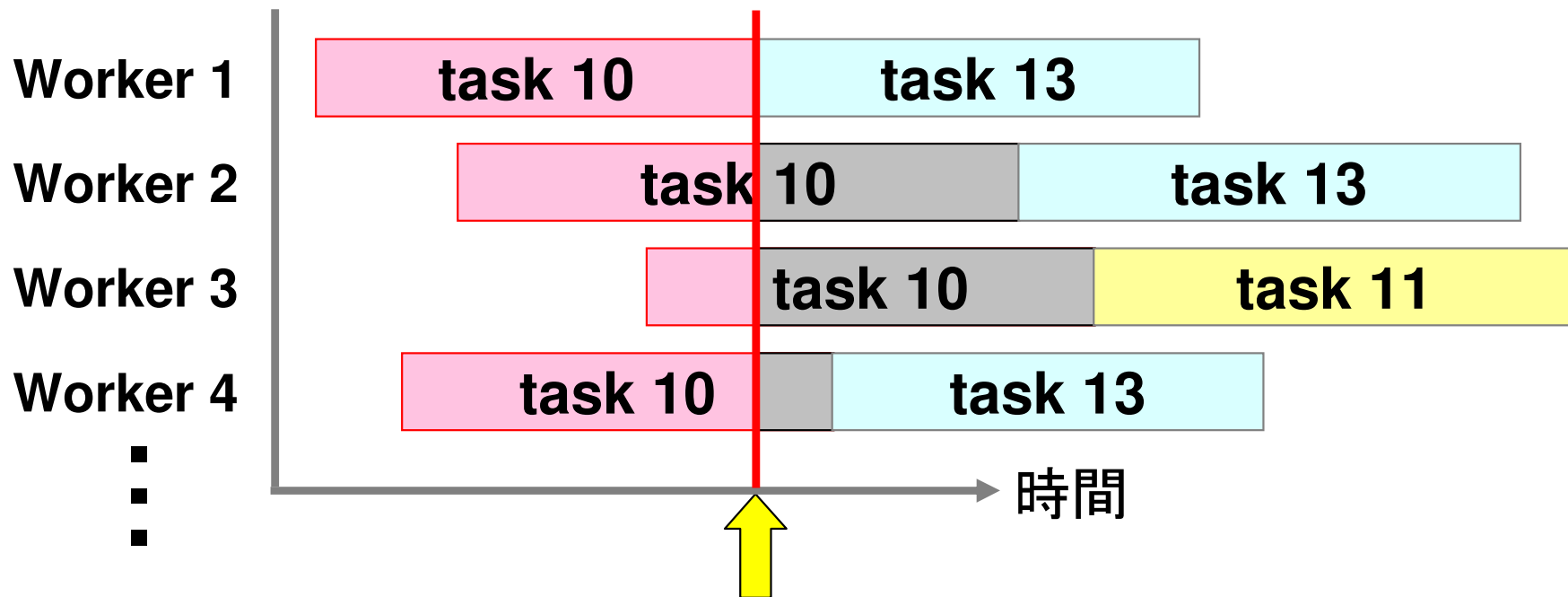
- タスクの多重化による実行時間のオーバーヘッドを削減するために、**ワーカの不要な処理の中断機構**を実装した。



この操作によって、実行時間のオーバーヘッドが削減される。

5. 提案手法 ～タスクの多重化～ (2)

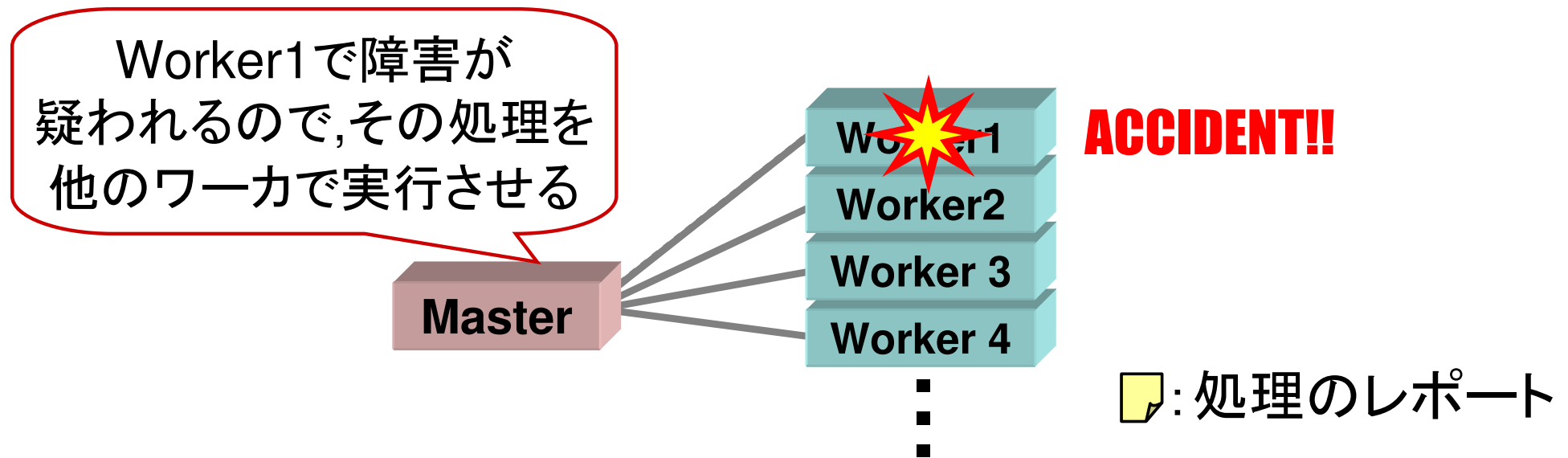
- 不要な処理の中断は、以下の様に実行時間を削減する。



これらの処理をキャンセルすることによって
実行時間を削減する

5. 提案手法 ～ワーカーの監視～ (1)

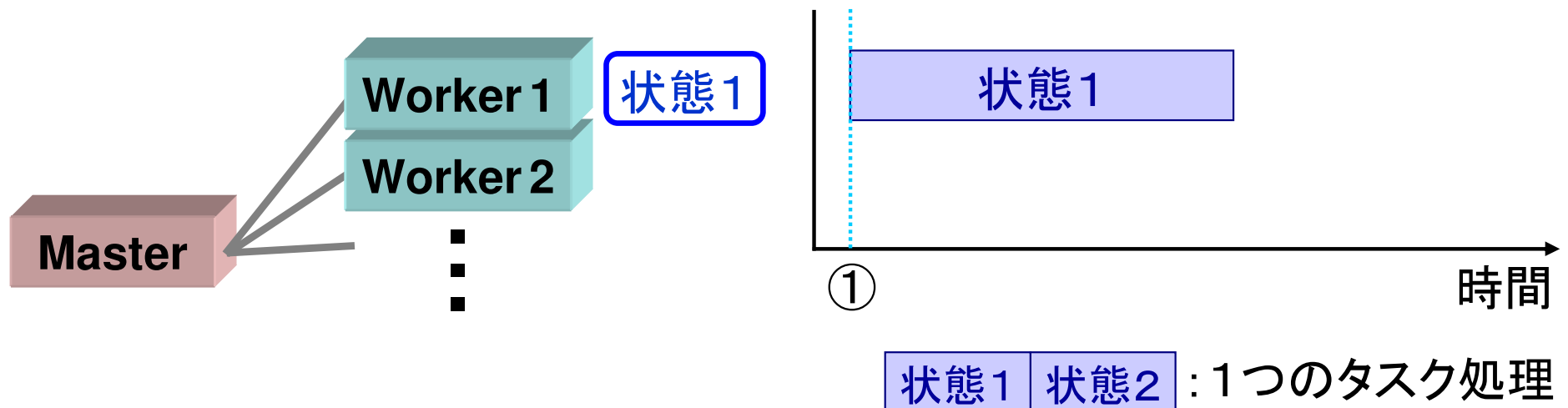
- ワーカーはマスタに処理状況を定期的に報告し、ワーカーの計算状況が全く変化しない場合、障害が発生していると推測する。



処理状況の報告が全く来ないワーカーは故障が発生したものと見なす。

5. 提案手法 ～ワーカーの監視～ (2)

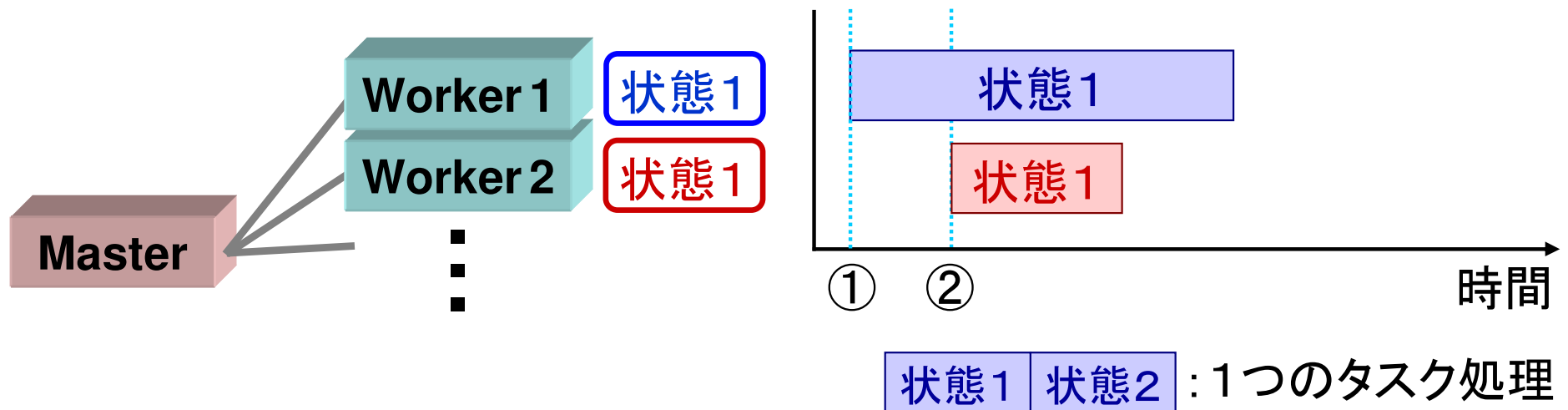
- マスタは以下のような場合にワーカーに障害が発生したと判断する.



① Worker1があるタスクの前半の処理を開始する。
これを【状態1】と呼ぶ。

5. 提案手法 ～ワーカーの監視～ (2)

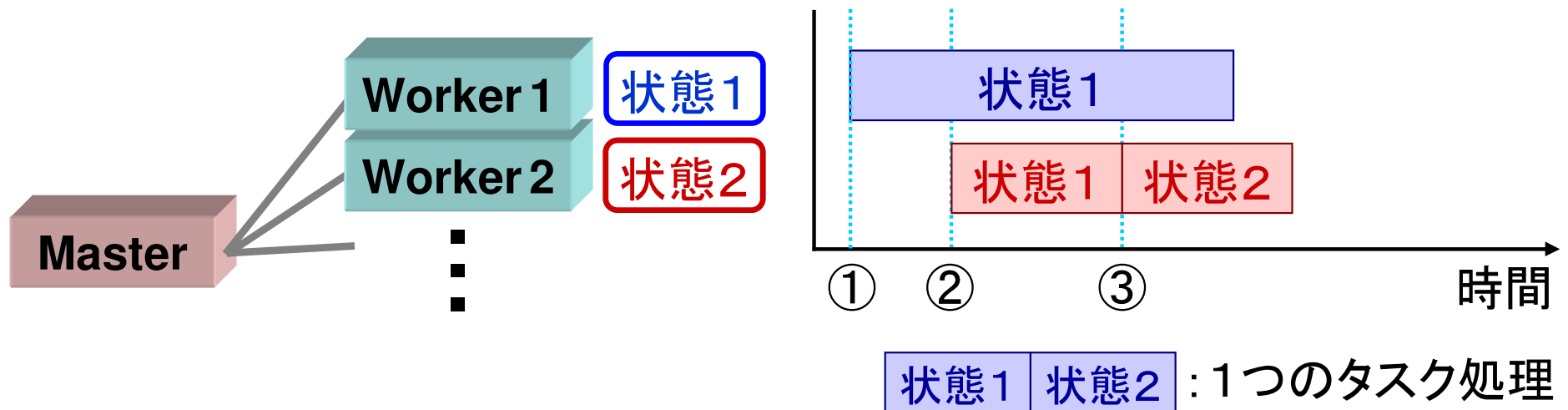
- マスタは以下のような場合にワーカーに障害が発生したと判断する.



② Worker2が別のタスクの前半の処理を開始する
【状態1】

5. 提案手法 ～ワーカーの監視～ (2)

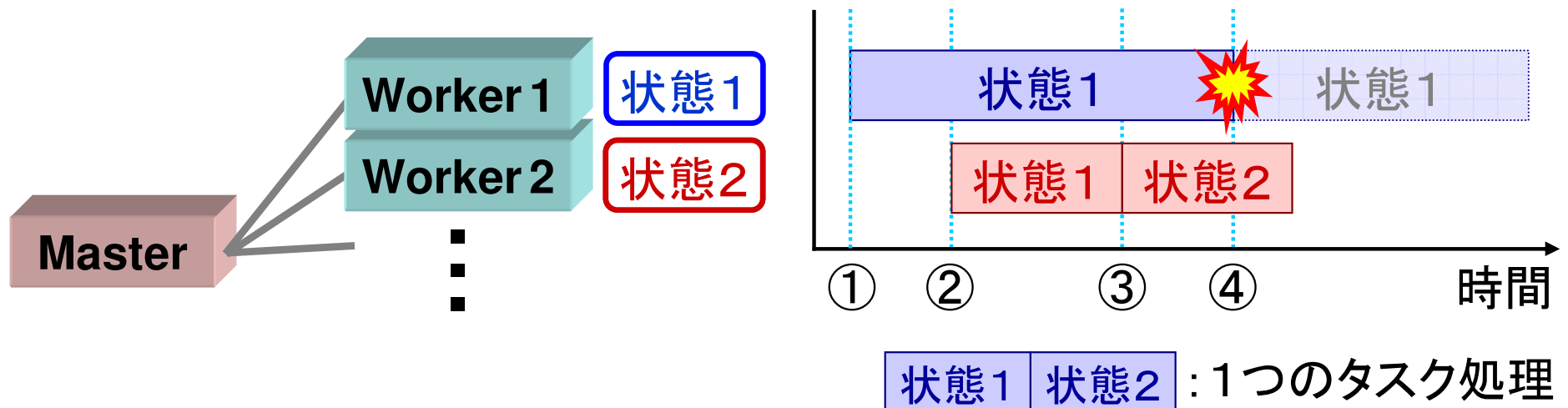
- マスタは以下のような場合にワーカーに障害が発生したと判断する.



③ Worker2がタスクの後半の処理を開始する
【状態2】

5. 提案手法 ～ワーカーの監視～ (2)

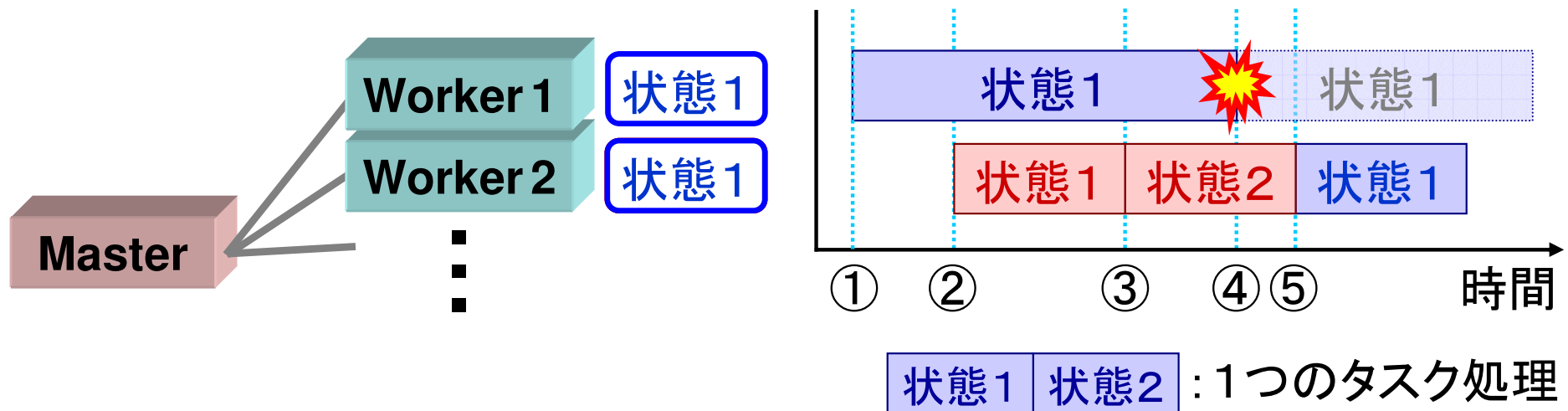
- マスタは以下のような場合にワーカーに障害が発生したと判断する.



④ Worker 1に何らかの障害が発生する.
これ以降Worker 1の状態は変化しない

5. 提案手法 ～ワーカーの監視～ (2)

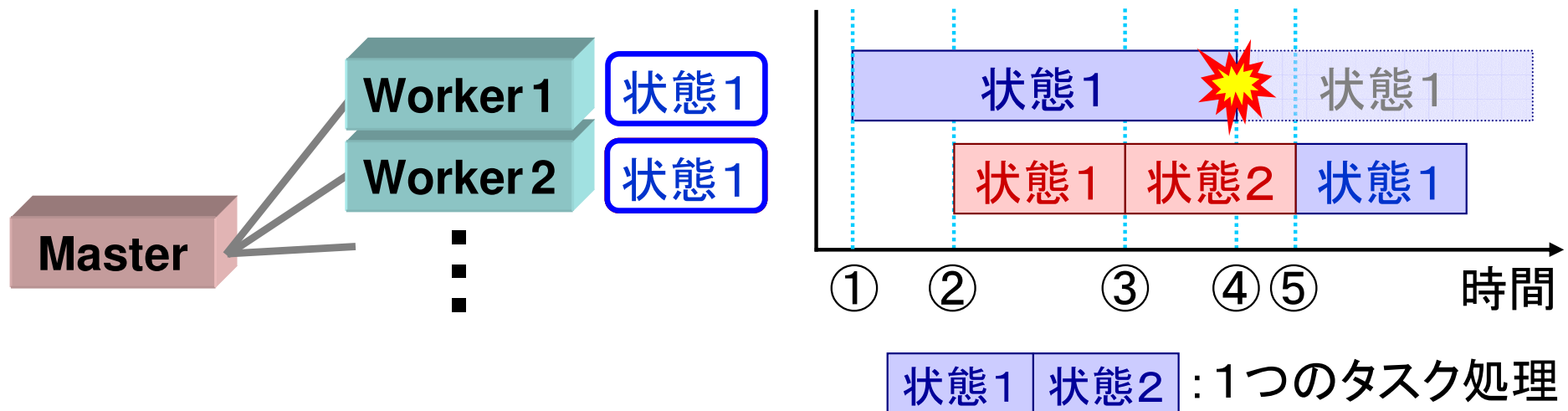
- マスタは以下のような場合にワーカーに障害が発生したと判断する.



⑤ Worker1の状態が変化しないので, Masterは Worker1で障害が発生したと判断し, そのタスクを Worker2に実行させる.

5. 提案手法 ～ワーカーの監視～ (2)

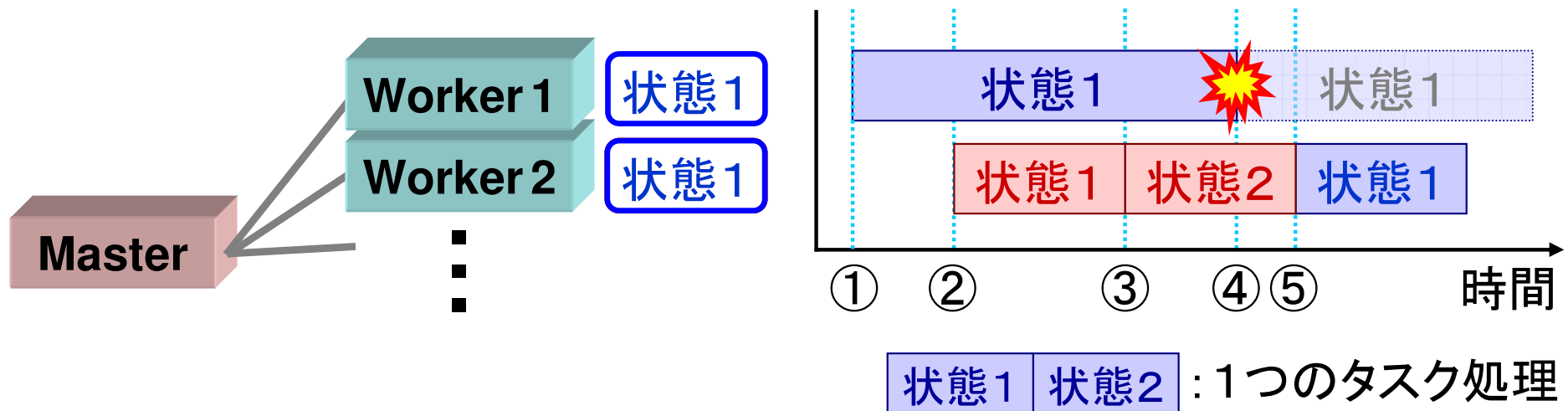
- マスタは以下の様な場合にワーカーに障害が発生したと判断する.



あるWorkerが他のWorkerの2倍以上処理に時間が掛かっている場合は, 障害が発生したと判断する.

5. 提案手法 ～ワーカーの監視～ (2)

- マスタは以下のような場合にワーカーに障害が発生したと判断する.



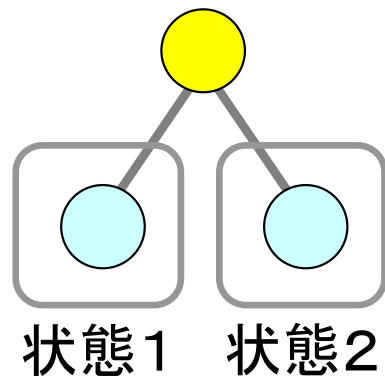
この手法は障害が発生した場合だけでなく、極端に処理が遅いワーカーが存在する場合にも効果を発揮する。
⇒耐故障性だけでなく、耐高負荷性も実現

5. 提案手法 ～ワーカの監視～ (3)

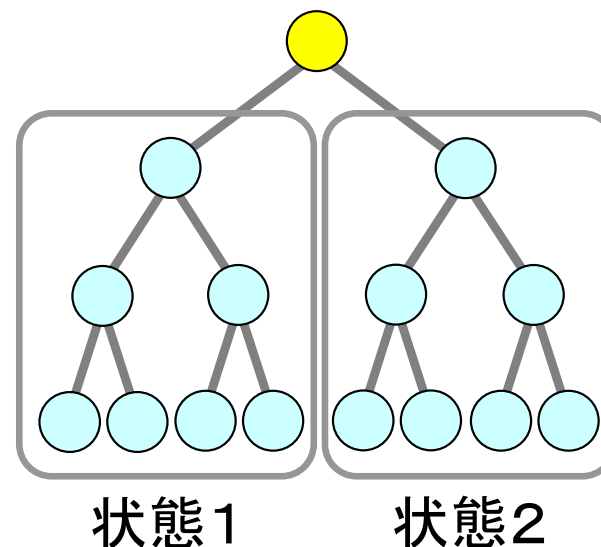
【他のワーカの2倍とした理由】

- ワーカが生成する子問題の数は最低で2であるため、丁度半分の処理が終了した時点で、ワーカの報告をすることとした。
- これによって、任意の並列分枝限定法アプリケーションで、この手法が適用可能となる。

【分枝の深さ: 1】



【分枝の深さ: 3】



● : タスク
○ : 子問題

6. アルゴリズムの評価

- 実行時間を評価するためにPC クラスタを用いて実験を行った。
- 実験構成

【使用したアプリケーション】

- BMI固有値問題アプリケーション

【各計算ノードの仕様】

- ノードA (1 nodes):

Intel Xeon 2.4GHz × 2CPU, Memory 512MB

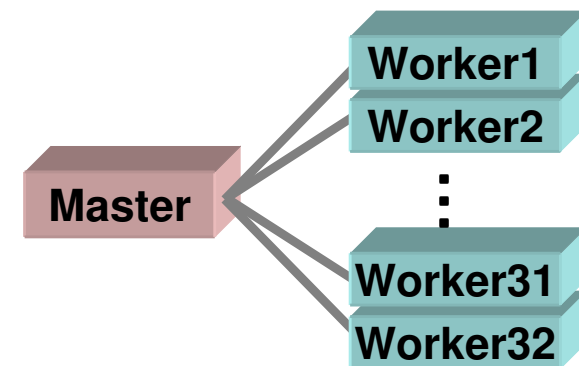
- ノードB (16 nodes):

Pentium III 1.4GHz × 2CPU, Memory 512MB

【各計算ノードの台数】

- マスタ: 1台, ワーカー: 32台

※Ninfを用いて実装



6. 実験の概要

【実験1】 基本性能の評価（ワーカ数:1～32）

ワーカの台数を変化させて実行時間を比較する.

【実験2】 不要な処理の中断機構の評価（ワーカ数:1～32）

不要な処理の中断機構による実行時間の変化を評価する.

【実験3】 耐高負荷性の評価（ワーカ数:32）

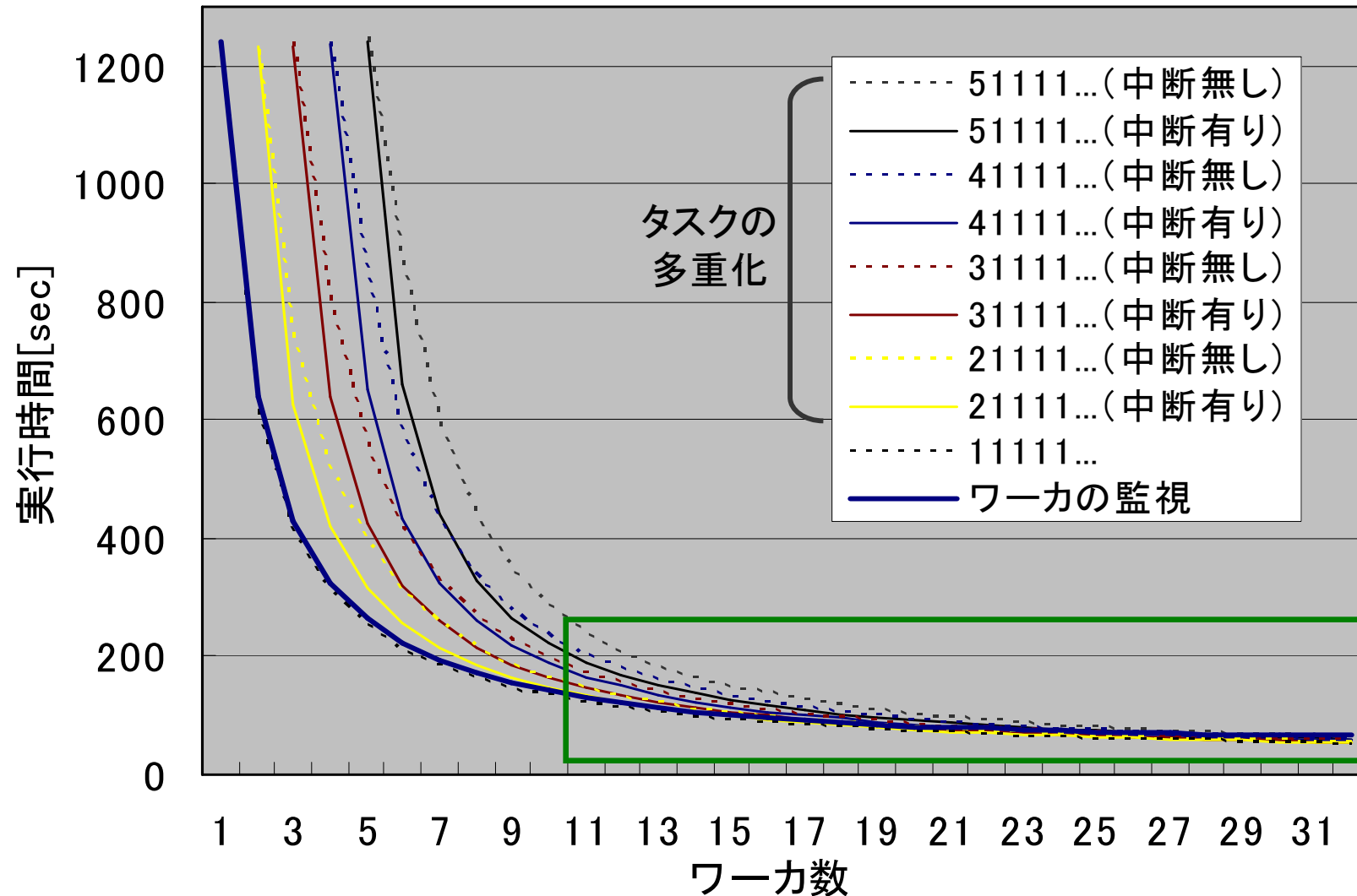
高負荷なワーカが存在する場合の実行時間を計測する.

【実験4】 耐故障性の評価（ワーカ数:32）

ワーカに障害が発生した場合の実行時間の変化を計測する.

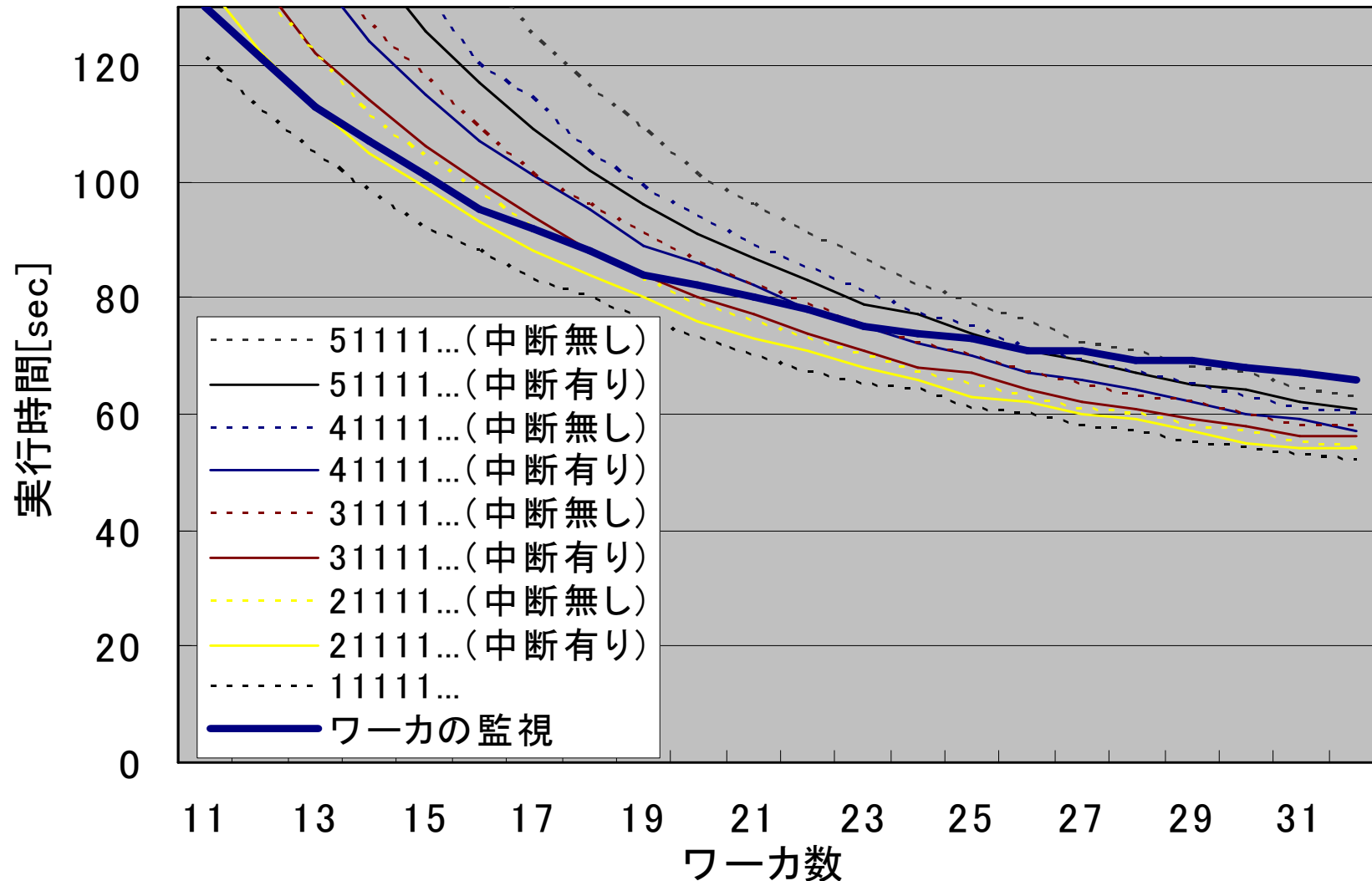
(※実験1～3は障害無し)

実験1. 耐故障アルゴリズムの実行時間への影響



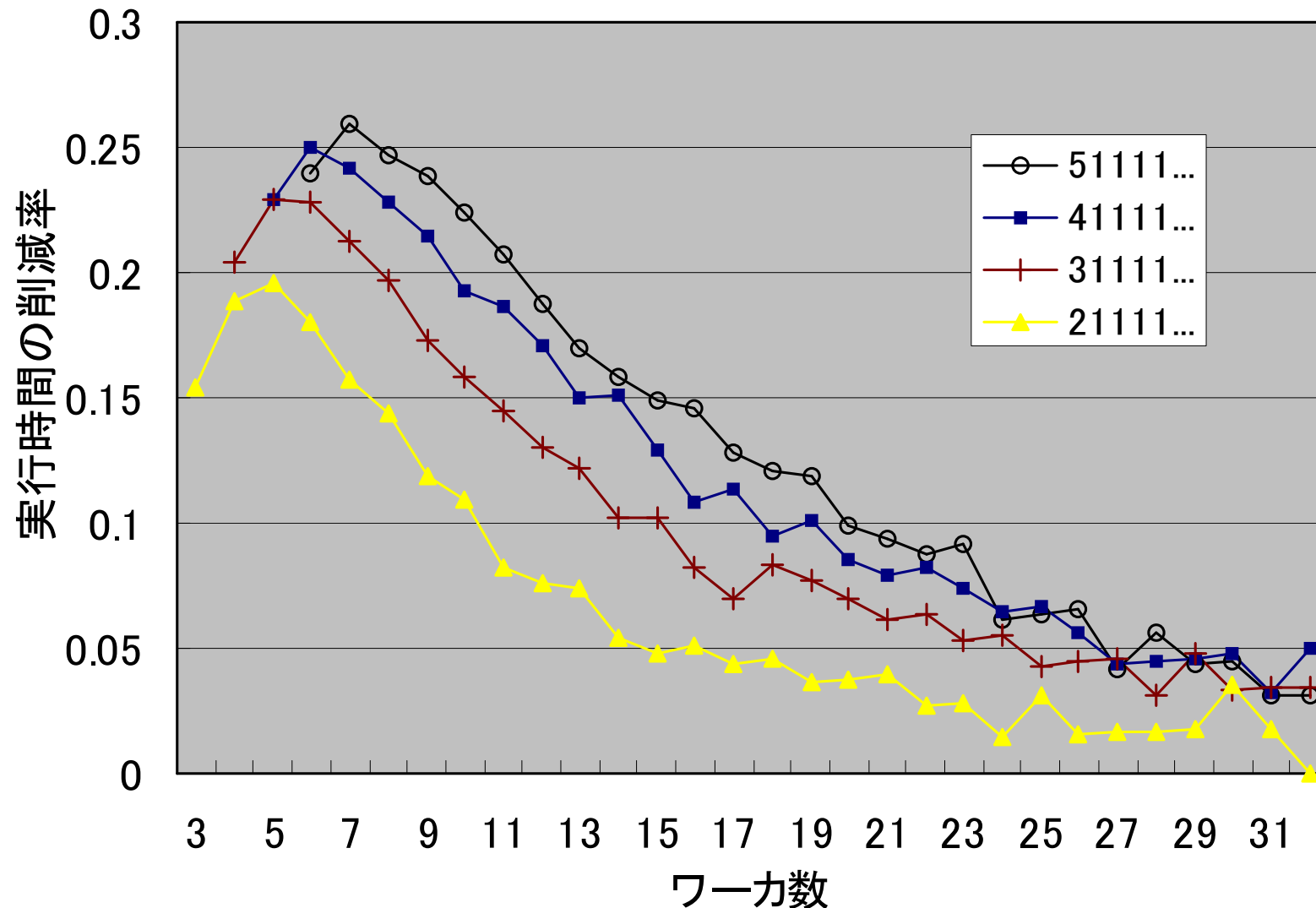
多重度リストの値が増加すると、実行時間も増大することが確認された。

実験1. 耐故障アルゴリズムの実行時間への影響



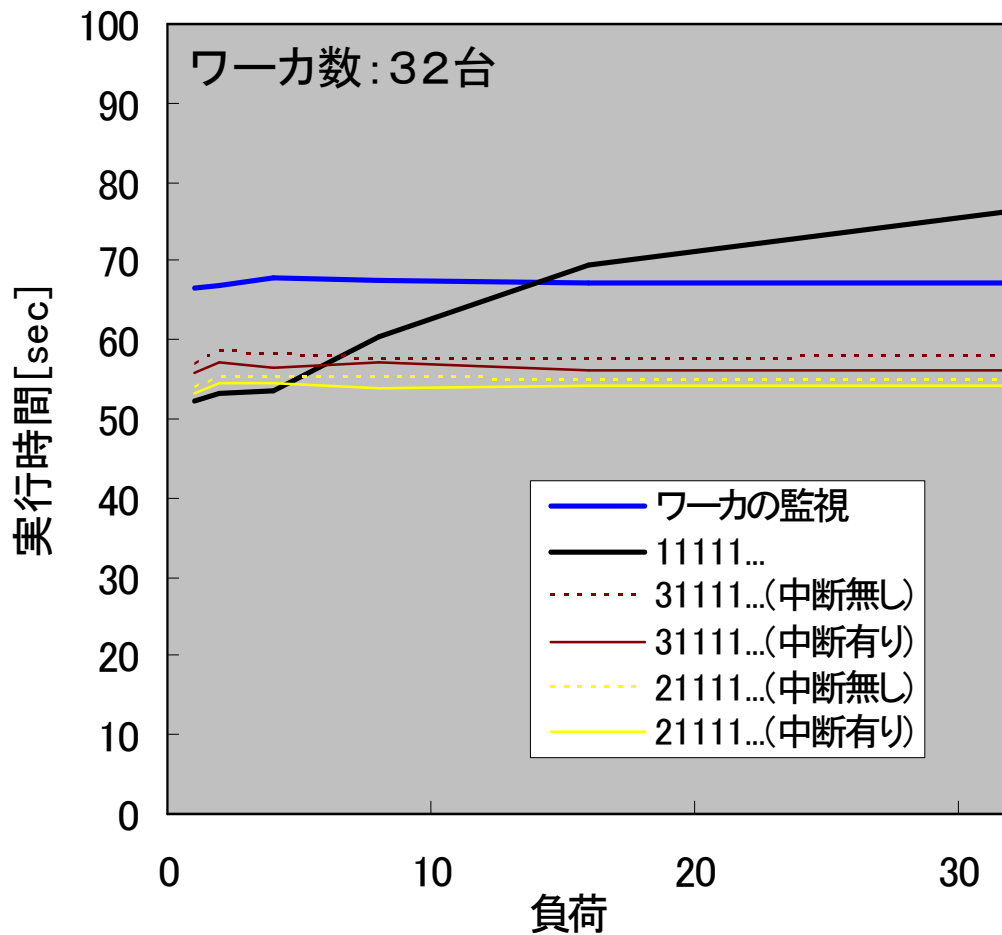
ワーカの監視による手法は、ワーカ数が増加するにつれて実行時間のオーバヘッドが増加した。
(マスタ・ワーカ間の通信が増加するため)

実験2. 多重度リストの値と中断機構の効果

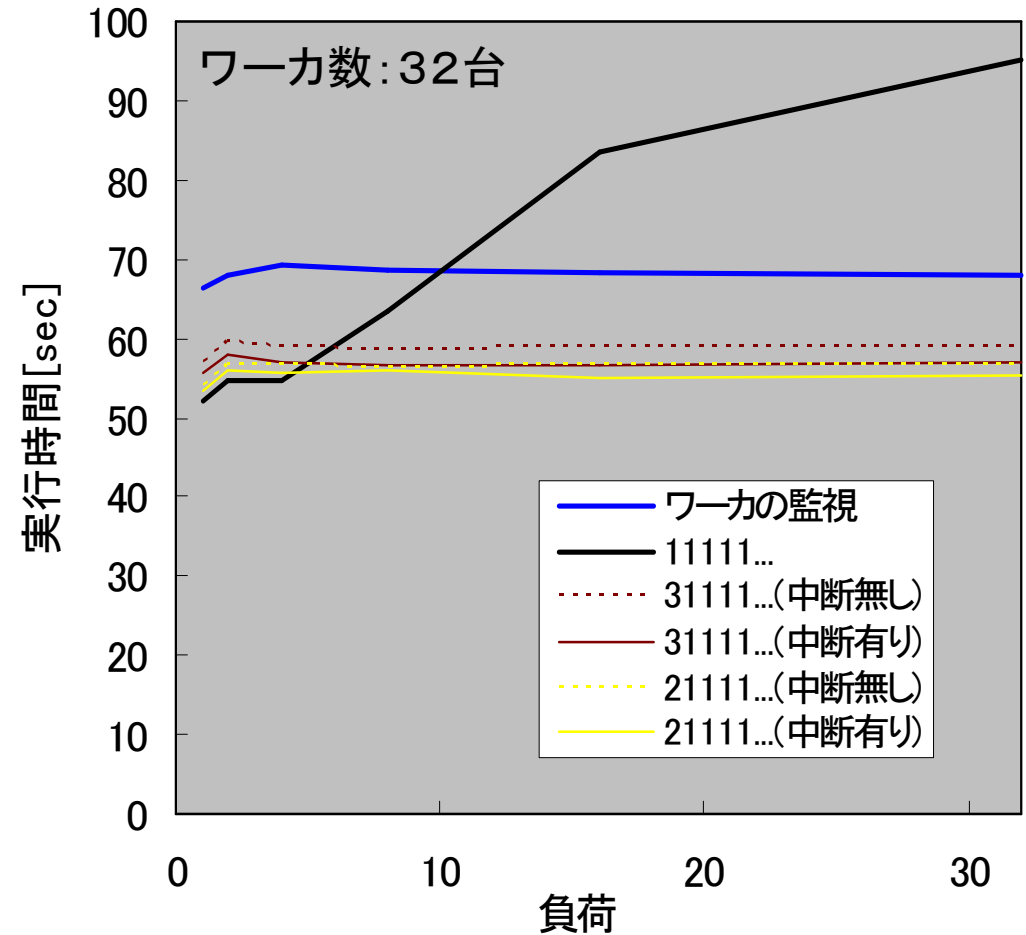


多重度リストの値が, ワーカの総数に対して大きい場合に, 削減率も大きくなった.

実験3. 高負荷なワーカによる実行時間への影響



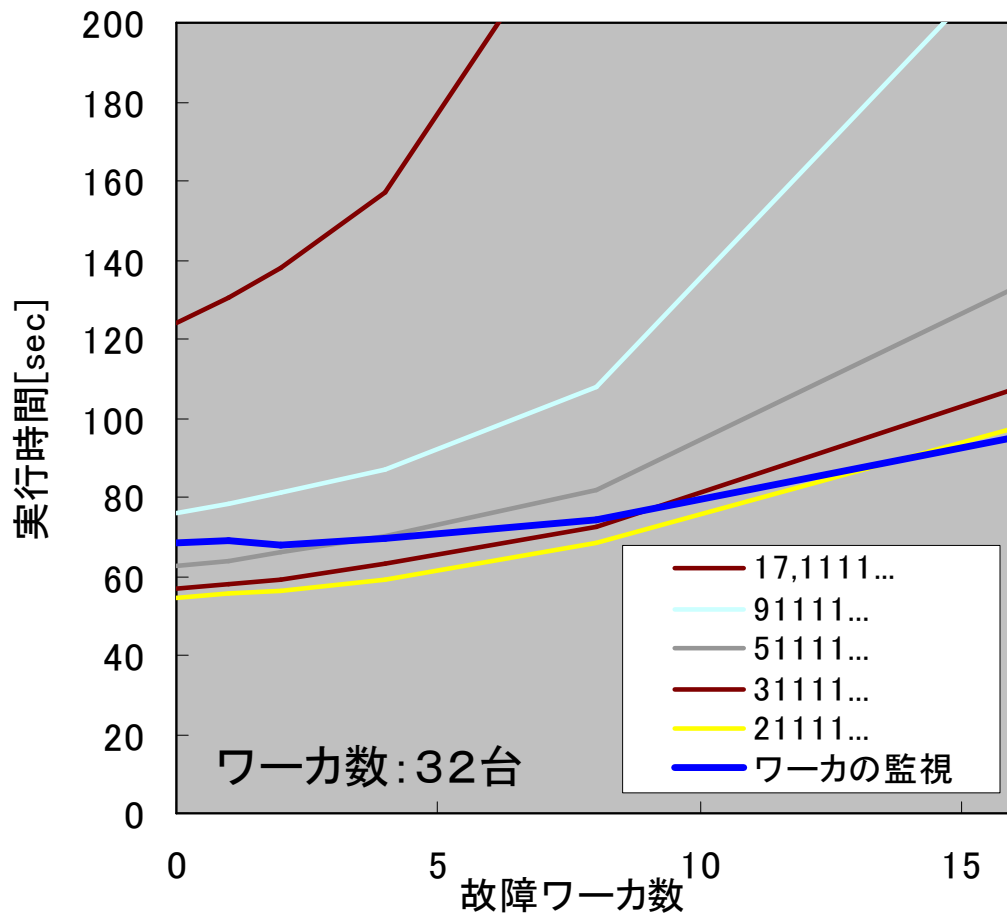
【高負荷ワーカ: 1台】



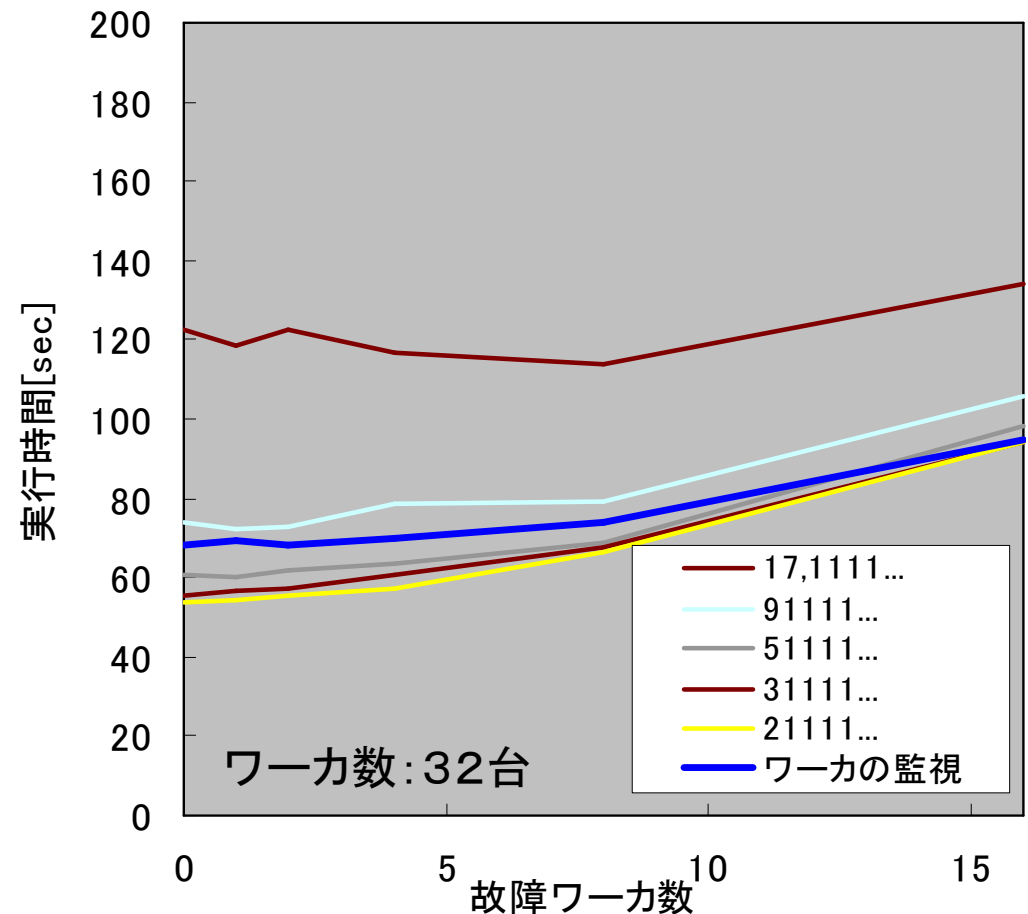
【高負荷ワーカ: 2台】

両手法とも、高い耐高負荷性を実現することが確認された

実験4. ワーカの障害による実行時間の変化



【不要な処理の中断: 無し】



【不要な処理の中断: 有り】

タスクの多重化手法に関しては、不要な処理の中断機構によって実行時間が最大で9割削減された。

7. 実験のまとめ

【ワーカの監視による手法】

- 障害が発生した場合でも、新たなオーバヘッドが生じる事無く、残存ワーカで処理が続行される事が確認された。
- この手法によって高負荷なワーカや障害が発生したワーカが存在しても、安定して処理が実行されることが確認された。

【タスクの多重化による手法】

- 実際に障害が発生した場合、不要な処理の中断によって実行時間が大きく削減された。
- 特に、多重度リストの値が大きい場合、実行時間は最大で9割削減された。

8. 2つの手法の比較

【タスクの多重化手法の利点】

- 実装によるオーバヘッドが殆ど無い

【タスクの多重化手法の欠点】

- ワーカ数が少ない場合冗長処理によるオーバヘッドが大きい
- ワーカの信頼性に応じて多重度リストを設定する必要がある

《ワーカの監視手法の利点》

- 多重度リストの値を設定する必要が無いため、様々な環境に対して容易に適用可能である

《ワーカの監視手法の欠点》

- ワーカ数が大きい場合、マスタ・ワーカ間の通信によるオーバヘッドが大きい

8. 2つの手法の比較

【タスクの多重化手法の利点】

- 実装によるオーバヘッドが殆ど無い

【タスクの多重化手法の欠点】

- ワーカ数が多い場合、ワーカー処理に使用するメモリ量が大きい

ワーカ数が十分に大きい場合 ⇒ タスクの多重化手法が有効

ワーカ数が小さい場合 ⇒ ワーカの監視手法が有効

- 多重度リストの値を設定する必要が無いため、様々な環境に対して容易に適用可能である

《ワーカの監視手法の欠点》

- ワーカ数が大きい場合、マスタ・ワーカ間の通信によるオーバヘッドが大きい

9. まとめと今後の展望

【まとめ】

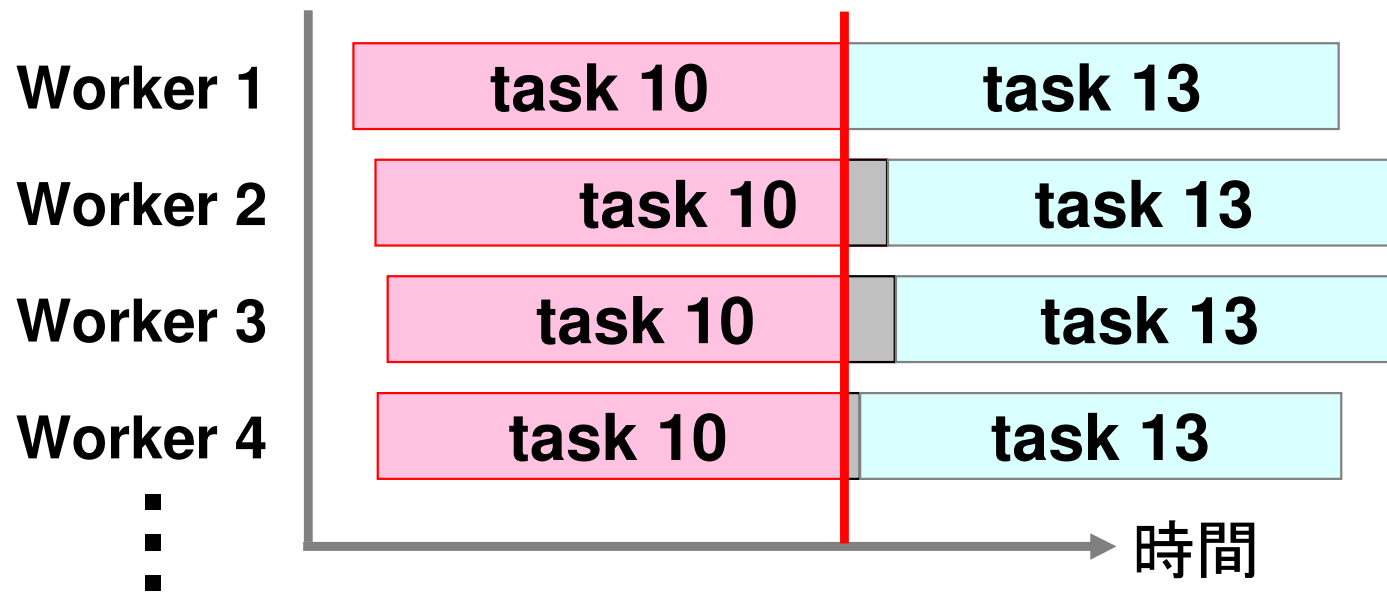
- 耐故障性を実現するための2つの手法を考案し、それらの性能を評価した。

【今後の展望】

- 2つの手法について、故障のシナリオを変化させて比較する。
- タスクの多重化による手法に関しては、多重度を処理の状況に応じて変化させ、実行時間の変化を調べる。
- 両者のアルゴリズムの理論的な解析を行う。
(実行時間と故障確率の関係, 信頼性と多重度の関係 etc)

【補足1】 提案手法 ～タスクの多重化～

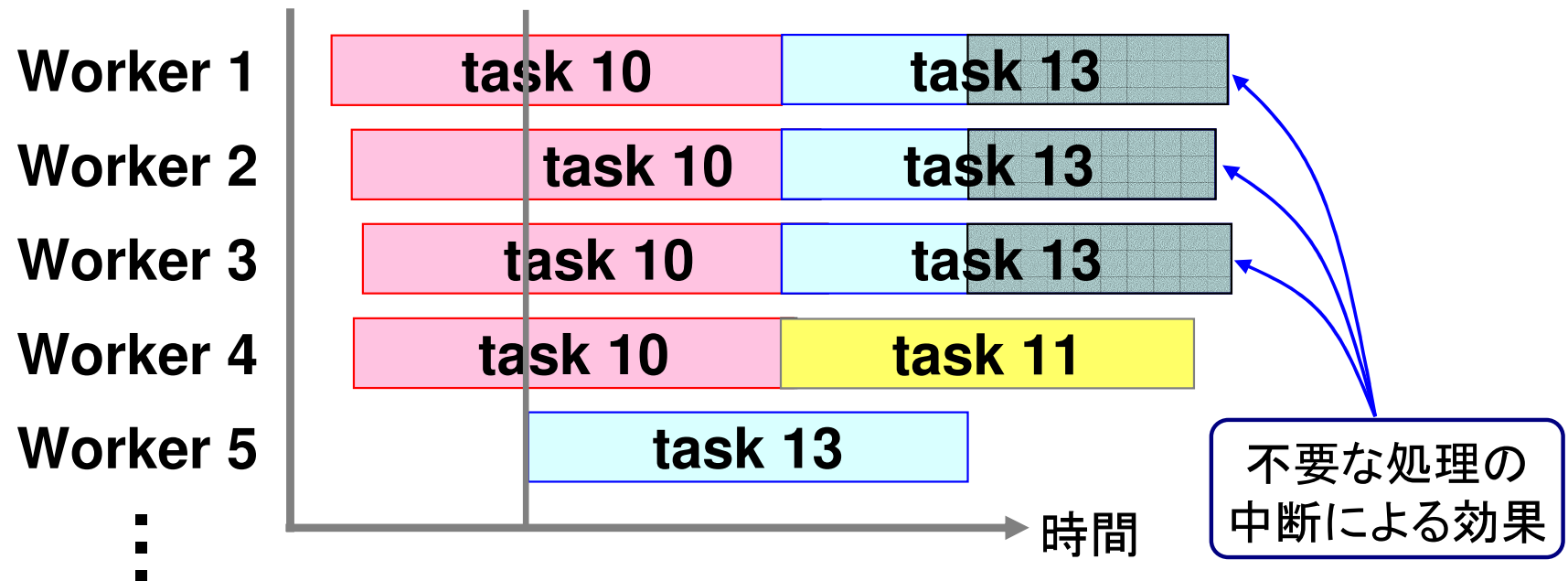
- 複数のワーカが同時に同じタスクの処理を開始すると、不要な処理の中断効果は低下する.



多重度が4である場合、ワーカ1～4が常に同じタスクを実行することによって中断効果が減少する恐れがある。

【補足1】 提案手法 ～タスクの多重化～

- しかし、実際は同一のワーカが常に同じタスクを処理する状況にはならない。



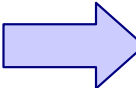
| | | | | | |
|-------|----|----|----|----|----|
| タスク番号 | 10 | 13 | 11 | 14 | 12 |
| 優先度 | 98 | 85 | 77 | 60 | 51 |
| 多重度 | 4 | 1 | 1 | 1 | 1 |

【補足2】 提案手法 ～タスクの多重化～

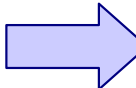
【タスクの多重化による手法(中断無し)】

- 多重度リストを「51111...」とし、8台のワーカで実行した場合、4台のワーカで障害が発生すると、以下のようなになる。

多重度リストの先頭の値が5のため、全てのワーカが同じ問題を処理する事になる
⇒ 逐次版とほぼ同じ時間が掛かってしまう。



| | | | | | | | | |
|-------|---|---|---|---|---|---|---|---|
| ワーカ番号 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| タスク番号 | 5 | 5 | 5 | 5 | 2 | 2 | 2 | 1 |



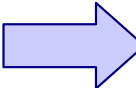
| | | | | | | | | |
|-------|----|----|----|----|---|---|---|---|
| ワーカ番号 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| タスク番号 | 98 | 98 | 98 | 98 | 2 | 2 | 2 | 1 |

【補足2】 提案手法 ～タスクの多重化～

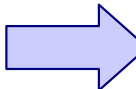
【タスクの多重化による手法(中断有り)】

- 多重度リストを「51111...」とし、8台のワーカで実行した場合、4台のワーカで障害が発生すると、以下のようなになる。

障害が発生したワーカに関して、その処理をキャンセルし新たにタスクを投げる。



| | | | | | | | | |
|-------|---|---|---|---|---|---|---|---|
| ワーカ番号 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| タスク番号 | — | — | 5 | 3 | — | — | — | 1 |



| | | | | | | | | |
|-------|---|---|---|---|---|---|----|---|
| ワーカ番号 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| タスク番号 | 5 | 5 | 5 | 3 | 5 | 5 | 10 | 1 |