

動的なノード群構成機構を備えた 階層型グリッド環境: Jojo2

青木仁志 (東工大)

中田秀基 (産総研、東工大)

田中康司 (早大)

松岡聡 (東工大、国情研)

背景



- 組み合わせ最適化問題

- 多次元パラメータ関数の最適値を求める
 - 分枝限定法、遺伝的アルゴリズム、etc
- 実用上の問題では膨大な計算量が必要
- 大規模細粒度アプリケーションも存在
 - 実行時間が 1 秒未満の小さなタスクなど

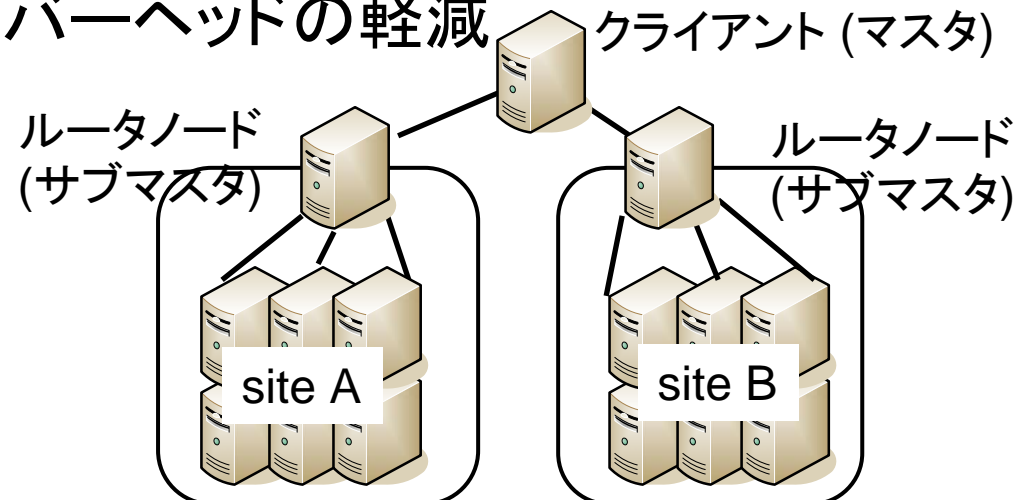
→ グリッド上でのマスタ・ワーカ方式による解法

- 単純なマスタ・ワーカ方式による問題点

- マスタとワーカ間の通信のオーバーヘッド
- ワーカ数に対するマスタのスケーラビリティ

階層的マスタ・ワーカ方式

- マスタ・ワーカ間にサブマスタを導入
- マスタ・ワーカ方式におけるマスタの機能を、マスタと複数のサブマスタに分担
 - 単一マスタへの負荷集中の問題を解決
- サブマスタとワーカを単一のクラスタ内に配置
 - サブマスタとワーカ間の通信をクラスタ内に局所化することでオーバーヘッドの軽減

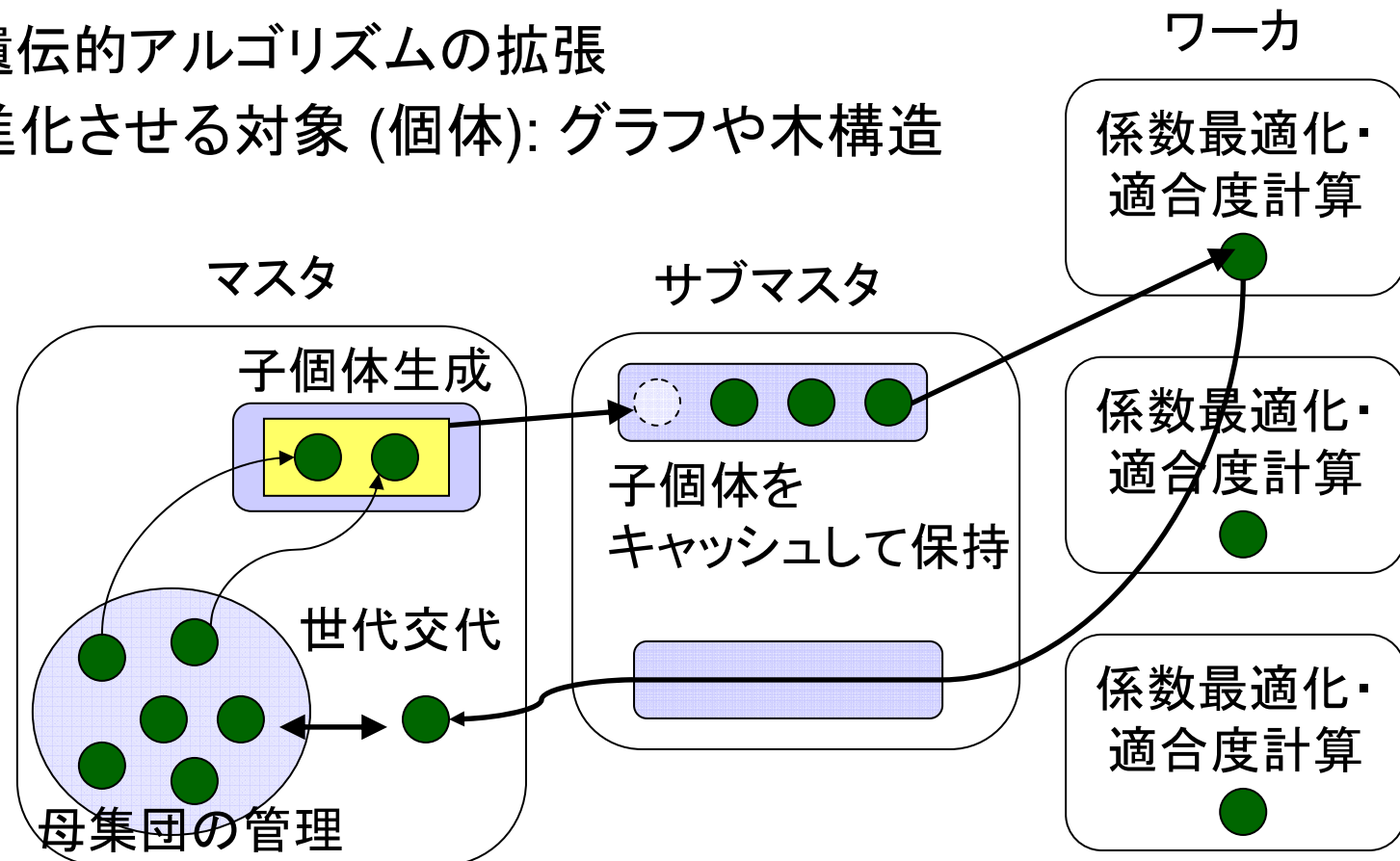


階層的マスタ・ワーカ方式による アプリケーション実装の例

- 遺伝的プログラミングによる遺伝子ネットワーク推定

- 遺伝的プログラミング

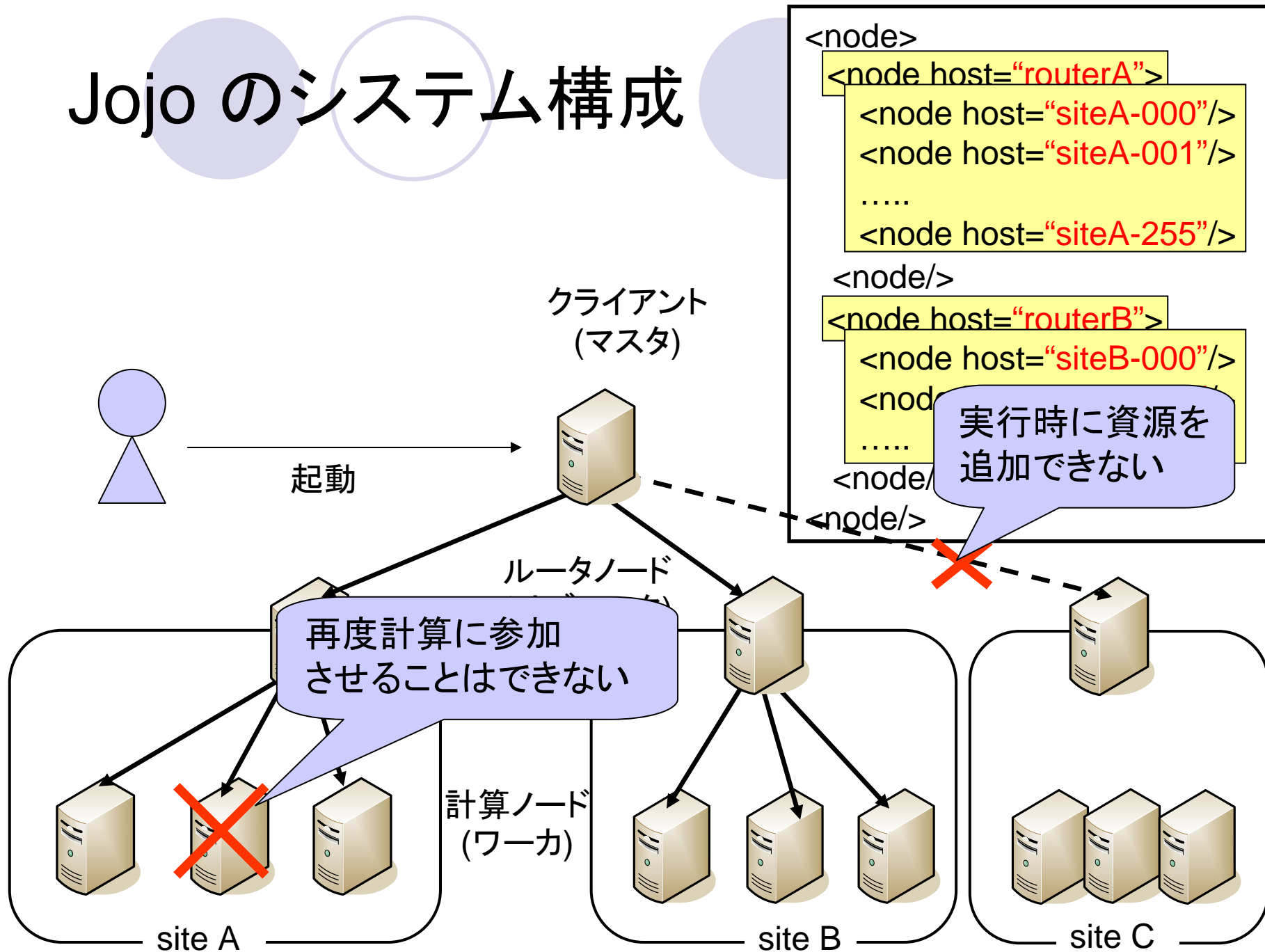
- 遺伝的アルゴリズムの拡張
- 進化させる対象 (個体): グラフや木構造



関連研究 – Jojo [Nakadaら '04]

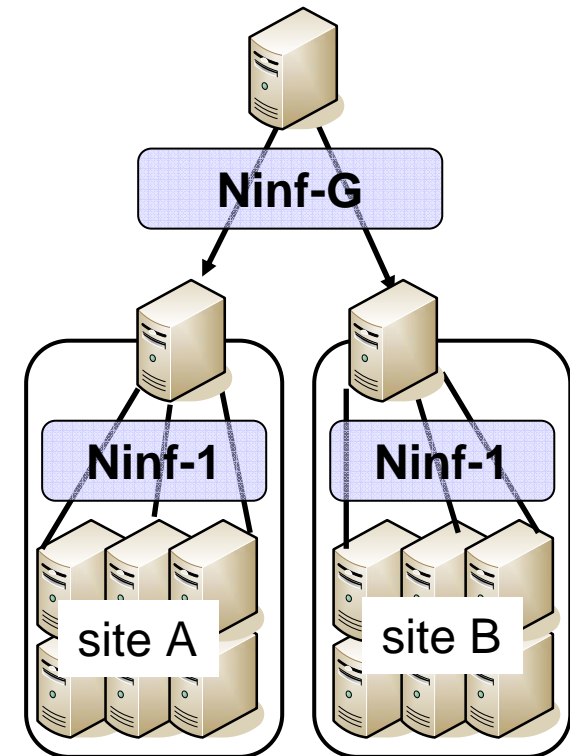
- 階層構造を持つ環境に適した分散実行環境
 - 柔軟な多階層実行機構
 - GSI [Fosterら '98] や ssh を用いたセキュアな起動と通信
 - 直感的で並列実行に適したメッセージパッシング API
 - プログラムコードの自動アップロード
- 問題点
 - 煩雑で静的なコンフィグレーション
 - ノード数固定を前提としたプログラミングモデル
 - 耐故障性の欠如

Jojo のシステム構成



関連研究 [合田ら'04][小野ら'05]

- Ninf-G + Ninf-1 による階層的
マスタ・ワーカ方式のプログラミング
 - クラスタ外のネットワークでは Ninf-G
 - 強力なユーザ認証機構、通信路の暗号化
 - クラスタ内では Ninf-1
 - セキュリティ機能を提供しない代わりに高速な通信が可能
- 問題点
 - 動的なノードの追加・削除が不十分
 - 複数のプログラミングツールを組み合わせなくてはならないので、ユーザの負担大





問題点

- 既存の階層的マスタ・ワーカ方式のプログラミング環境の問題点

- 煩雑で静的なコンフィグレーション

- 数千台規模ではメンテナンスコスト大


- 動的なノードの増減への対処が不十分

- 必要に応じた実行中のノードの追加、削除が不可欠

⇒ 動的なノード群構成が不可欠

- 事前設定不要な動的なノードの発見

- 動的なノードの参加、脱退



目的と成果

- 目的

- 動的なノード群構成を備えた階層型グリッドプログラミング環境の提供

- 成果

- グリッドの階層構造に適合した動的なノード群構成の提案
- 本プログラミング環境を現実的なアプリケーションに適用し予備的な性能評価を行った

提案

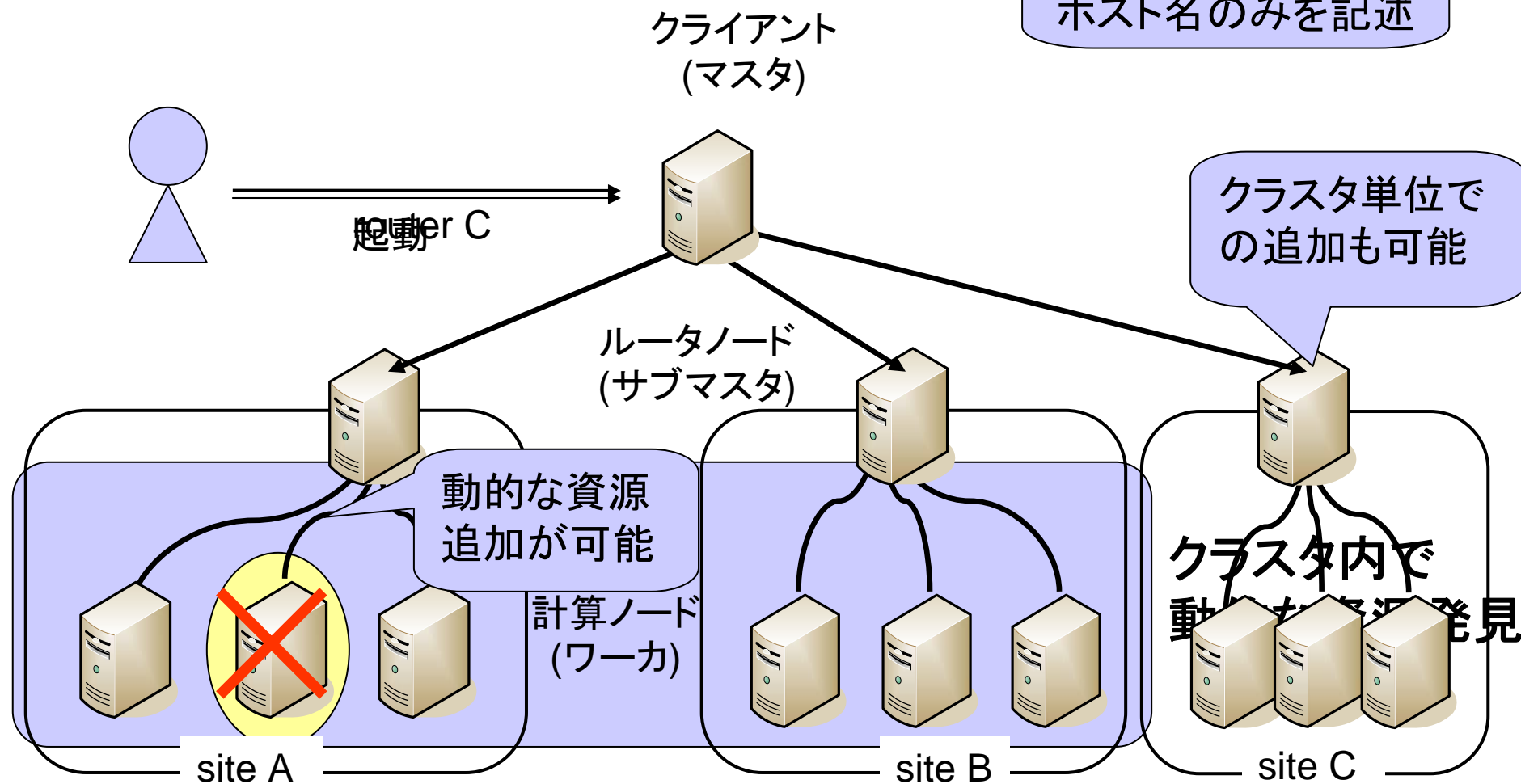


- Jojo をベースとした動的構成機構を備えたプログラミング環境 Jojo2
 - グリッド環境に適合した階層構造
 - クラスタ外のネットワークでは静的なコンフィグレーションを用いたサブマスタの起動
 - クラスタ内では動的なノード群構成
 - 動的なノード増減に対応したクラスフレームワーク

Jojo2 のシステム構成

```
<node>  
  <node host="routerA"/>  
  <node host="routerB"/>  
</node>
```

サブマスタの
ホスト名のみを記述



システムの設計



- 耐故障性

- Heartbeat & Timeout による故障検知

- 不均質性

- Pure Java による実装

- ポータビリティの確保

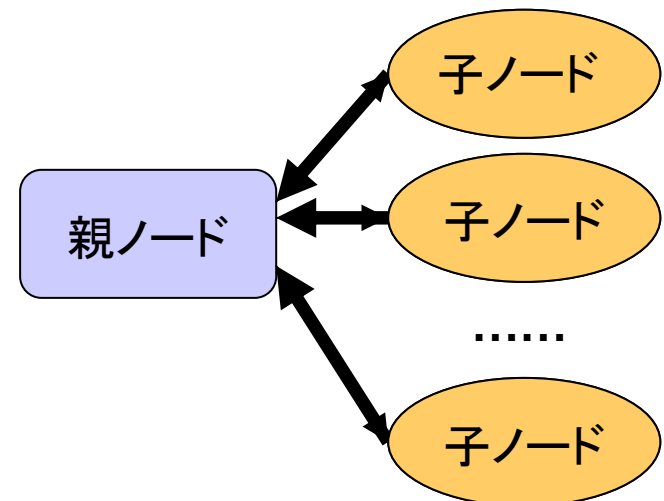
- ユーザプログラムの自動アップロード機構

- 広域での分散実行を前提としているので、ユーザが全ノードにアップロードするのは負担大

- プログラムのバージョンの違いなどのトラブルを未然に防ぐ

クラスフレームワークの設計

- メッセージパッシング機構の提供
 - × 識別子を用いた子ノードへのメッセージ配送
 - 子ノード群は動的に変化
 - 参加・脱退したノードの識別子はどうするか？
- Pull 型のプログラミングモデルを採用
 - 子ノードが親ノードにジョブ等を要求
 - 下位レベルとの通信はブロードキャストのみをサポート
 - 特定子ノードとの通信を制限
 - ただし子ノードからのリクエストに対する返信はユニキャスト

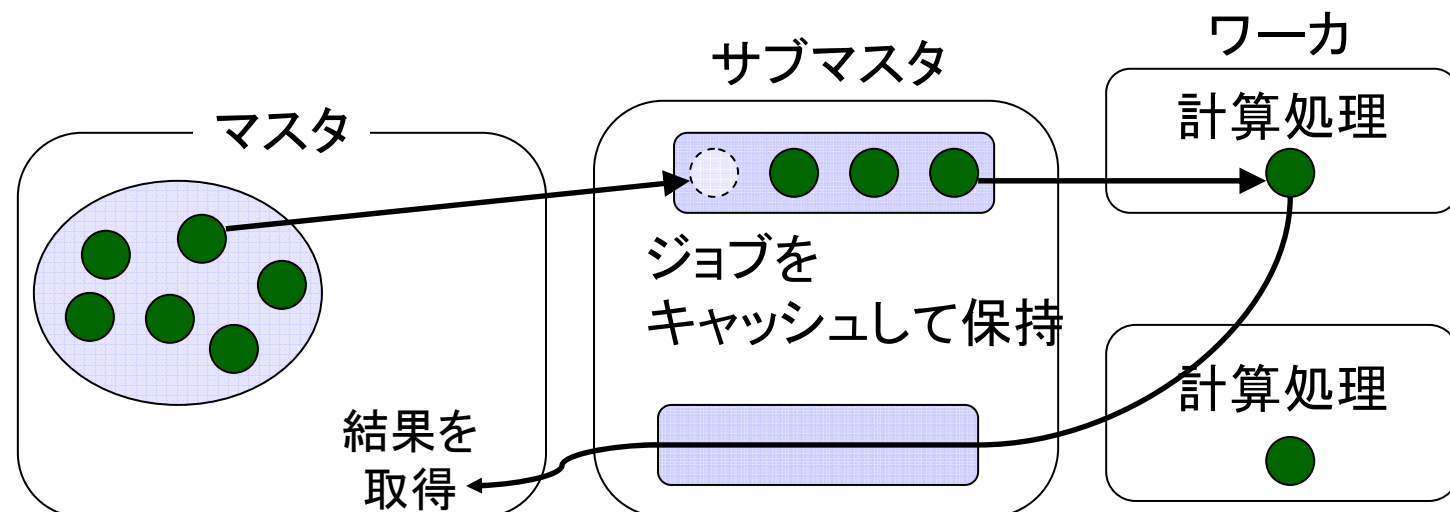


クラスフレームワークの設計

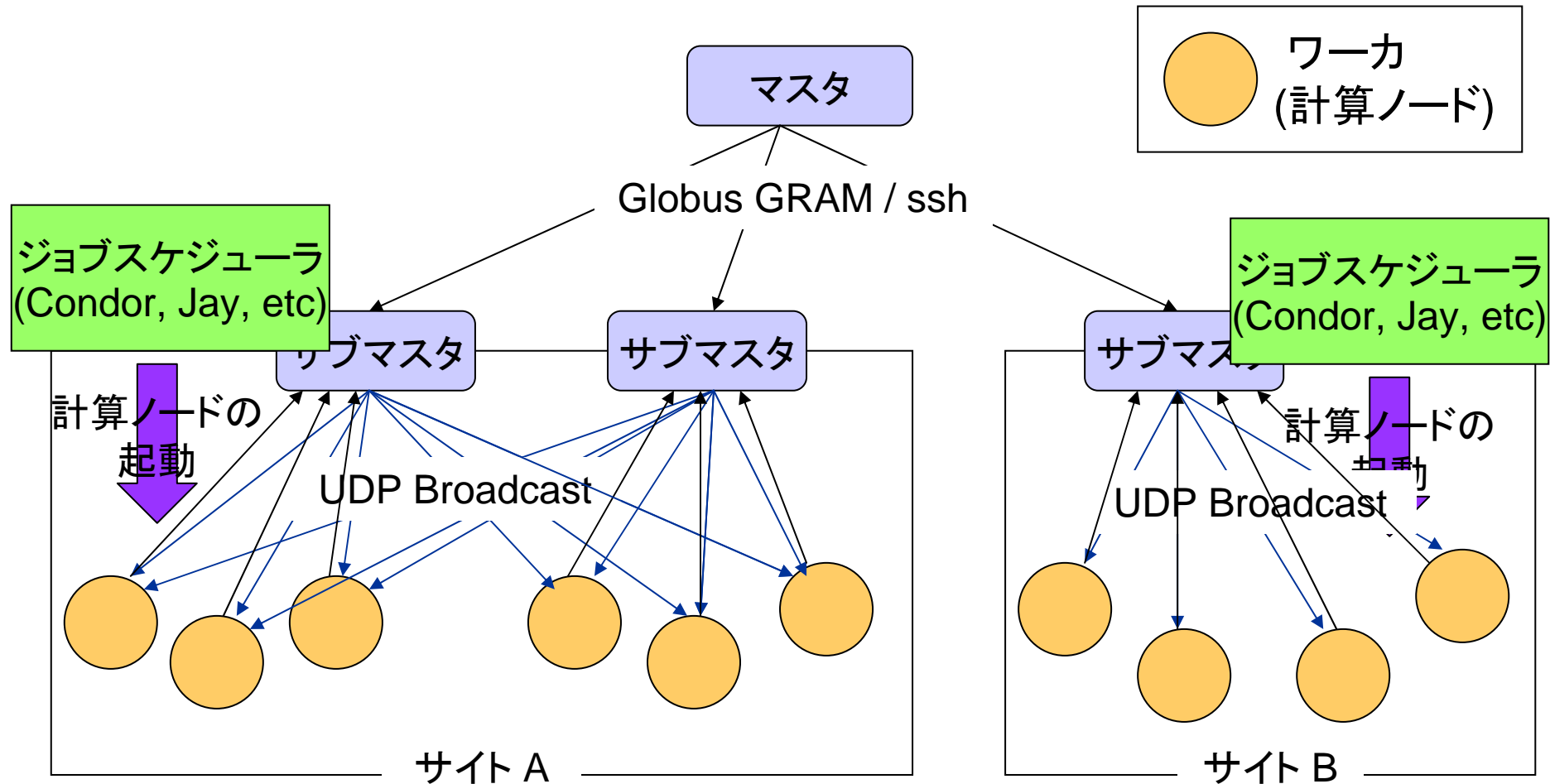
- ノードの参加、脱退を検出する機構が必要
 - 故障ノードに割り当てたジョブの再配布
 - 子ノード数に応じた処理の変更など
- ノードの参加、脱退をハンドルするメソッドを提供
 - ノードが参加、脱退したときに呼び出される
 - ユーザがノードの参加、脱退に対する処理を記述
 - アプリケーションによって必要な処理が異なるため

クラスフレームワークの設計

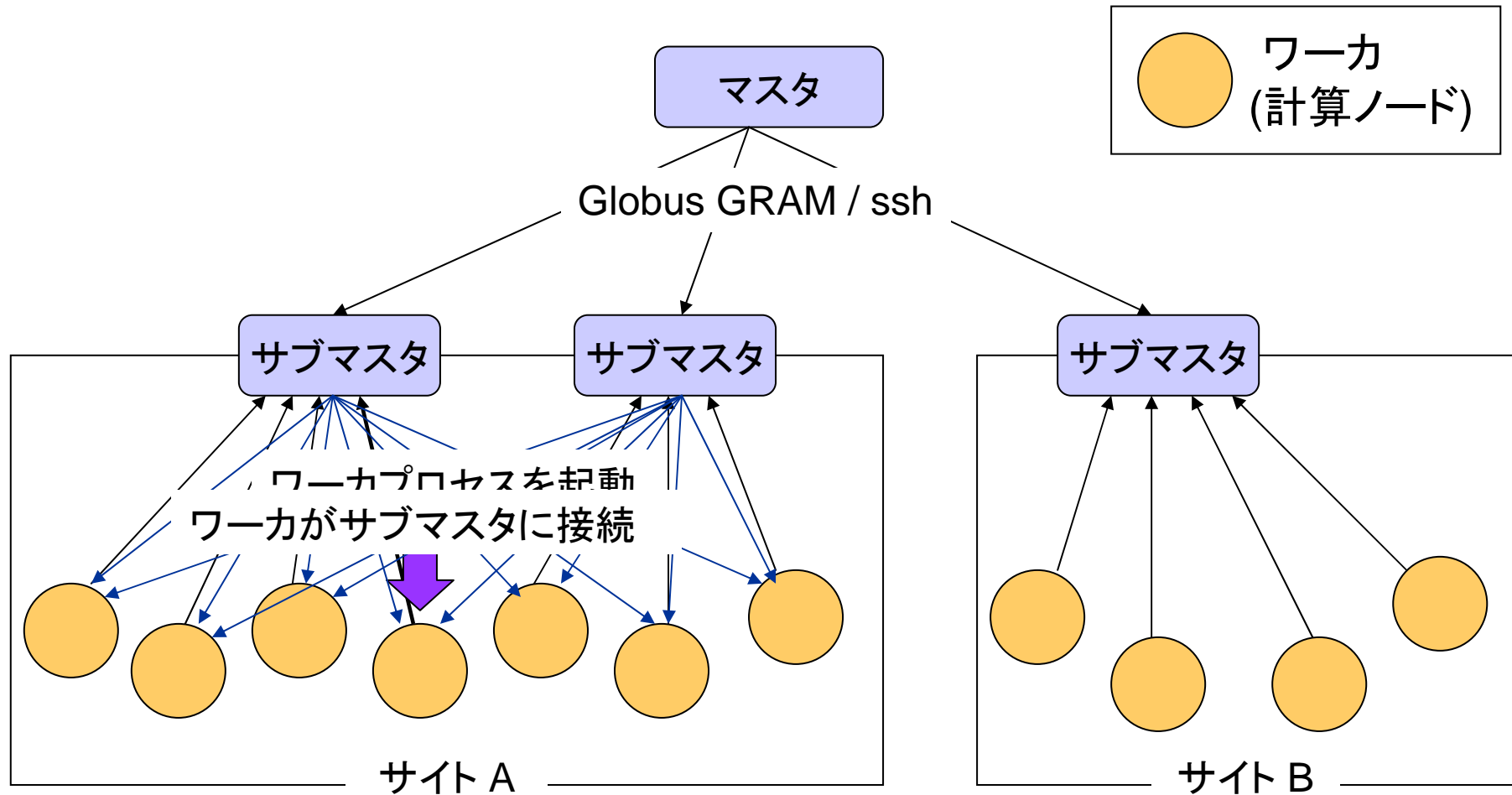
- ユーザがノードの参加・脱退のハンドラを記述するのは負担が大きい場合がある
- 特定のケースで利用可能な上位ライブラリを提供
 - ユーザはマスタとワーカの計算処理のみを記述



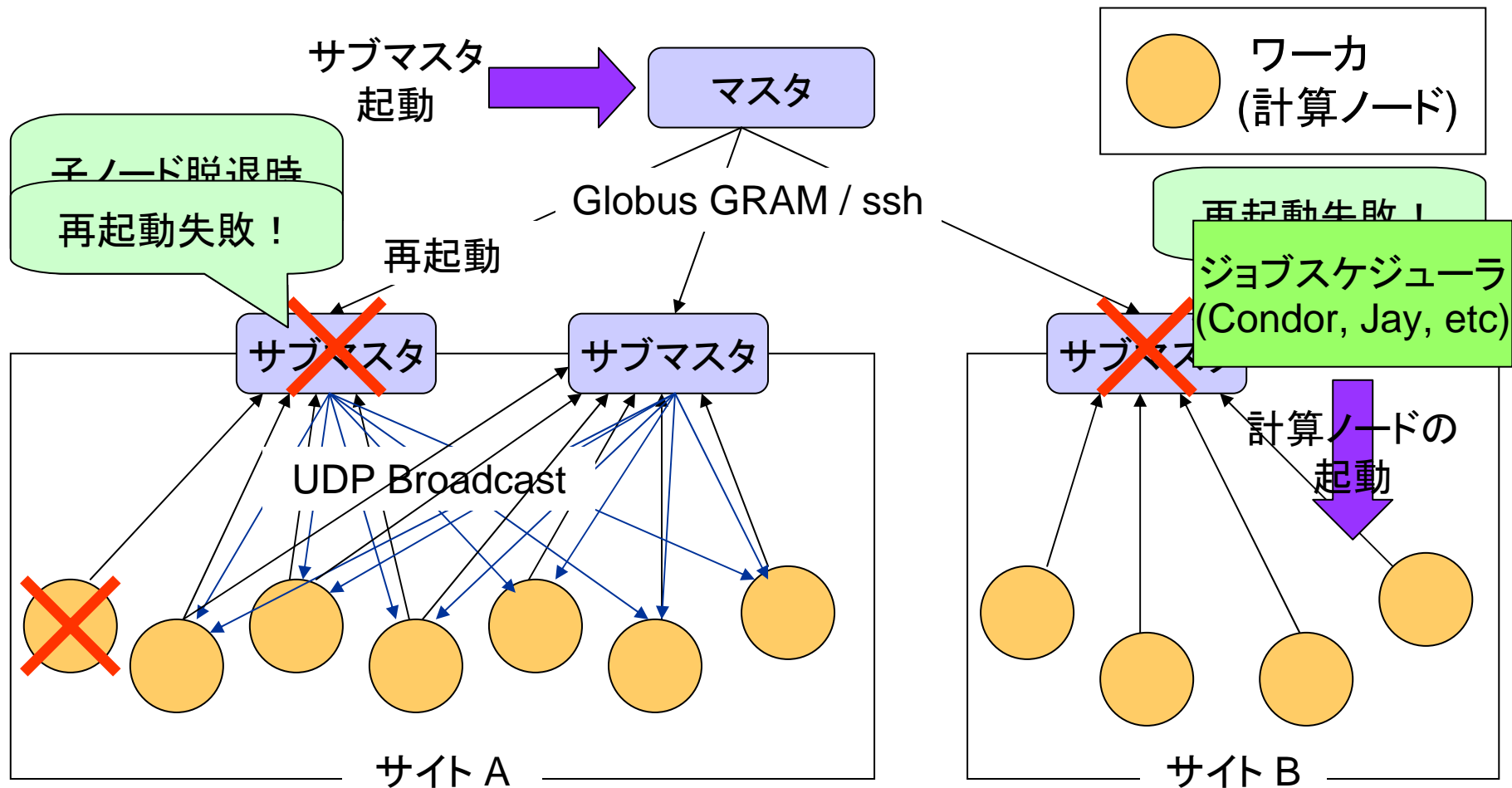
実装 - ノードの動的構成



実装 - ノードの動的な追加



実装 - 耐故障性



実装 – クラスフレームワーク

- ユーザは Jojo2 で提供される抽象クラス Code を継承して、プログラムを実装
 - 典型的には各レイヤごとに実装
 - マスタ、サブマスタ、ワーカ
 - ParentNode, DescendantNodes などのサポートクラスを用いてプログラミングを行う

クラスフレームワーク: Code

```
public abstract class Code {
    ParentNode parent; /* 親ノード */
    DescendantNodes descendants; /* 子ノード */

    /* 本体の処理 */
    public void start();

    /* 送信されてきたオブジェクトの処理 */
    public void handleReceiveParent(Message msg);
    public Object handleReceiveDescendants(Message msg);

    /* ノードの参加・脱退に対する処理 */
    public void handleAddDescendant(int id);
    public void handleDeleteDescendant(int id);
}
```

予備的性能評価

- 評価項目

- スケーラビリティの測定
- 耐故障性の検証

- サンプルアプリケーション

- 遺伝的プログラミングによる遺伝子ネットワークの推定アプリケーション

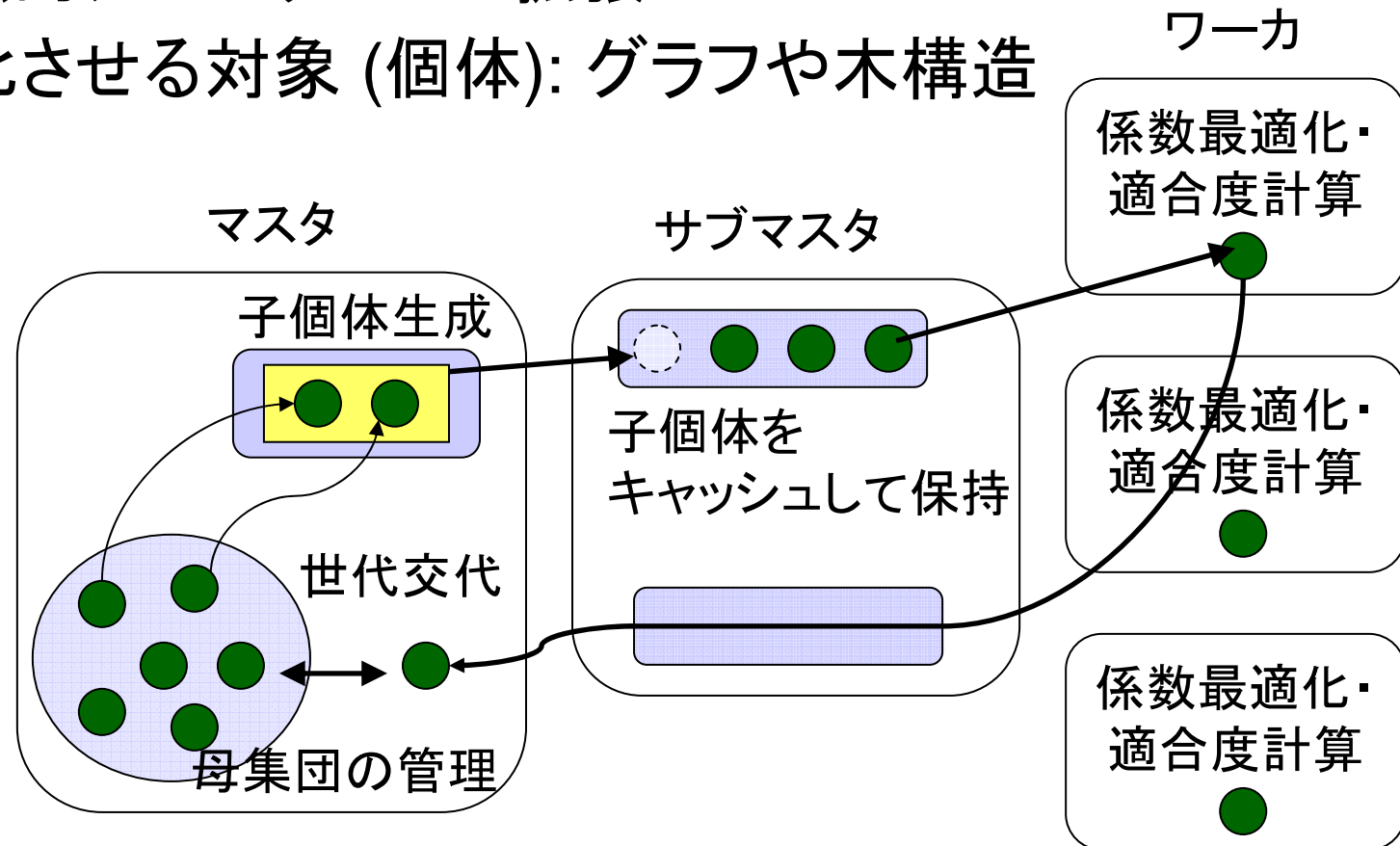
- 評価環境

- 松岡研究室 PrestoIII

CPU	Opteron 242
RAM	2 GB
Network	1000Base-T
OS	Linux 2.4.27
Java	JDK 1.5.0_06

遺伝的プログラミングによる 遺伝子ネットワーク推定アプリケーション

- 遺伝的プログラミング
 - 遺伝的アルゴリズムの拡張
 - 進化させる対象 (個体): グラフや木構造

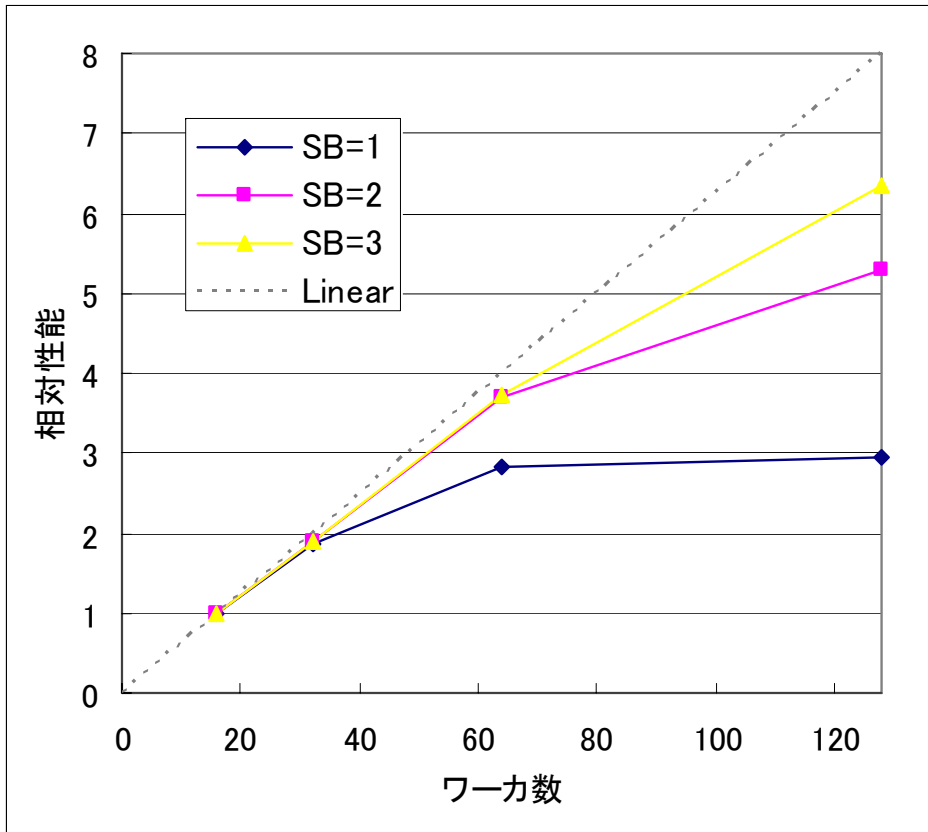


スケーラビリティの測定

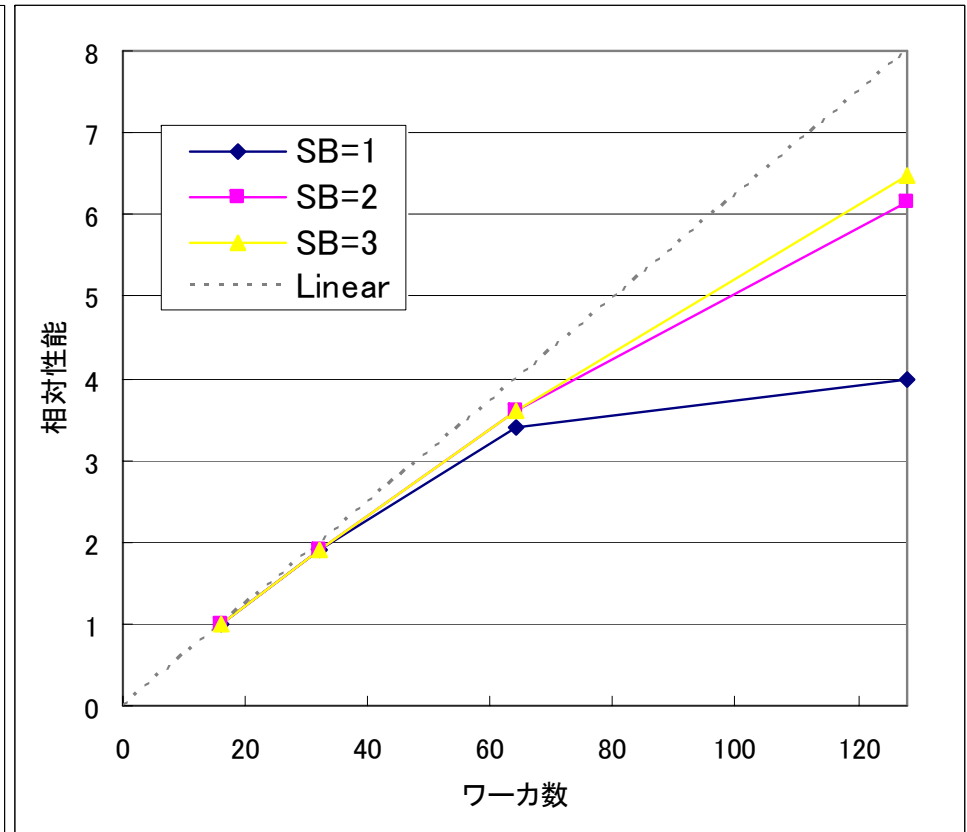
- サブマスタ数 (1~3)、ワーカ数 (8~128) を変えて実行時間を比較
- ルンゲクッタの刻み幅 (RK)
 - ワーカ上で行われる係数最適化、適合度計算に関わるパラメータ
 - ワーカの処理時間に影響

RK	処理時間 [ms]
1/4	143
1/8	228
1/16	490

スケーラビリティの測定結果

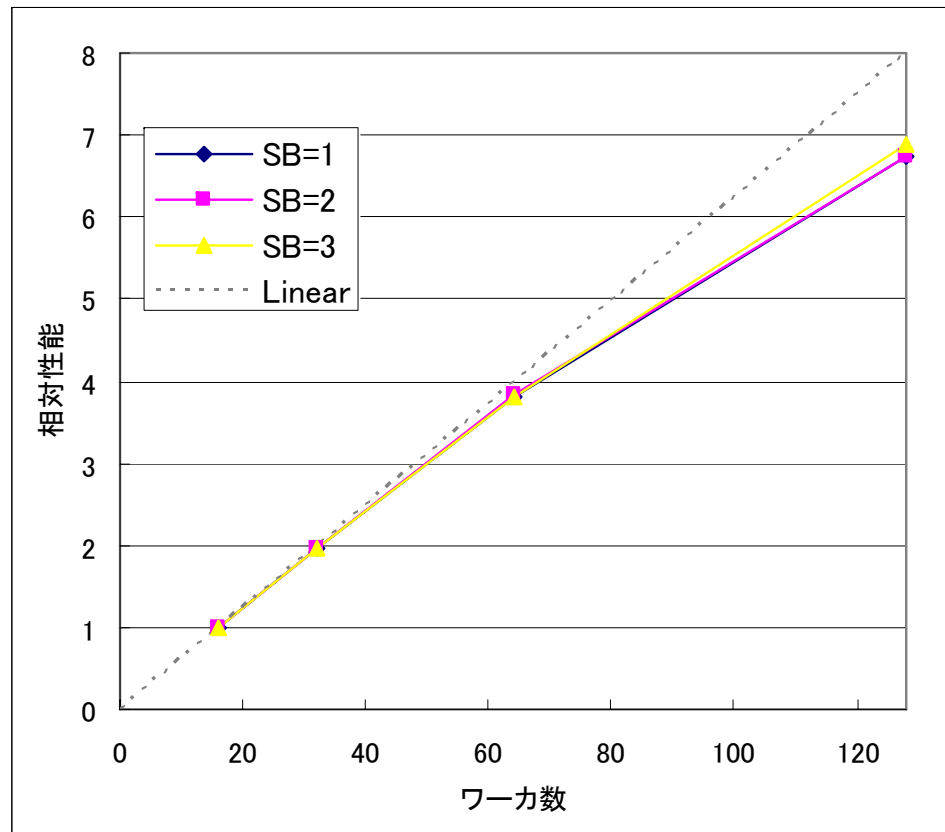


RK=1/4
処理時間: 143 [ms]



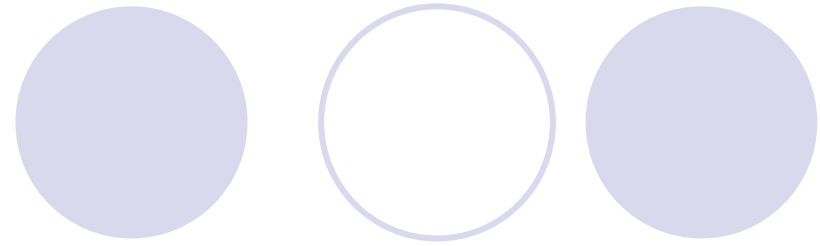
RK=1/8
処理時間: 228 [ms]

スケーラビリティの測定結果



RK=1/16
処理時間: 490 [ms]

耐故障性の検証



- 3 パターンの外乱を与えたときの実行時間を比較

- 起動時：サブマスタ数 1、ワーカ数 64

(a) 正常実行時

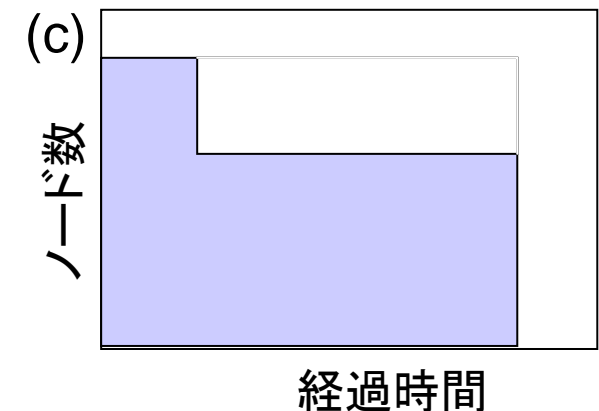
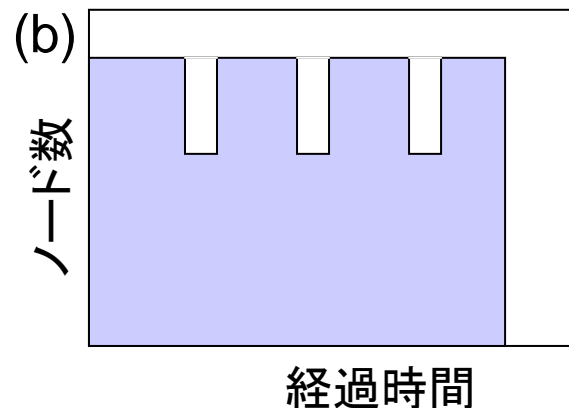
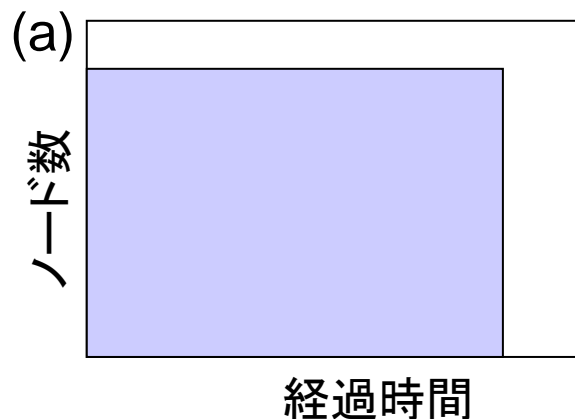
(b) 故障ノード発生時 (ノード追加あり)

- 3 分毎に 16 ノードのワーカプロセスを再起動 (計 3 回)

- プロセスを停止後、10 秒後にプロセスを起動

(c) 故障ノード発生時 (ノード追加なし)

- 起動後から 3 分後に 16 ノードのワーカプロセスを停止



耐故障性の検証結果

	(a) 正常実行時	(b) 故障ノード発生時 (ノード追加あり)	(c) 故障ノード発生時 (ノード追加なし)
実行時間 [sec]	682	709	839

- (b) でワーカプロセスを再起動してからジョブが割り当てられるまでの時間: 22.7 秒

考察

- (b) で故障によるオーバヘッドがないと仮定した場合の実行時間を求める

- (b) で停止したプロセスが計算に参加していない時間:
 $(10 + 22.7) \times 3 = 98.1$ 秒

停止してから再起動するまでの時間

ジョブが割り当てられるまでの時間

98.1 秒 + 684 秒 \doteq 683 秒

(b) の実行時間

(a) の実行時間

追加したノードに正常にジョブが割り当てられていることが確認された！

経過時間



まとめ

- 階層的なグリッド環境に適した実行環境
Jojo2 の設計・実装について述べた
 - グリッドの階層構造に適合した動的なノード群構成
 - 動的構成に適したクラスフレームワークの提供
- 予備的な性能評価と耐故障性の検証を行い
その結果を示した

今後の課題

- WAN 間の通信を伴う、より現実的な環境での評価
- 様々なアプリケーションへの適用
- Jojo2 を用いた並列化組み合わせ最適化システム jPoP の再設計・実装
 - jPoP [中川ら'04]
 - 最適化問題解法のためのテンプレート群
 - 問題領域に依存なデータ構造や操作のみを定義するだけで組み合わせ最適化アプリケーションを実装可能
 - 動的なノードの増減への対応