



# Grid 計算環境における デッドラインスケジューリングの性能

---

竹房 あつ子

日本学術振興会特別研究員 / 東京工業大学

松岡 聡

東京工業大学 / JST

<http://ninf.is.titech.ac.jp/bricks/>



# Grid計算環境

---

- ✎ HPCアプリケーションのための計算プラットフォーム
- ✎ 効率よいアプリケーション配置
  - ? スケジューリング
    - ✎ 単一アプリケーションの実行時間を最短にすることが目的
    - e.g., AppLeS, APST, AMWAT, MW, performance surface, stochastic scheduling, etc.



# NES: Network-enabled Server

---

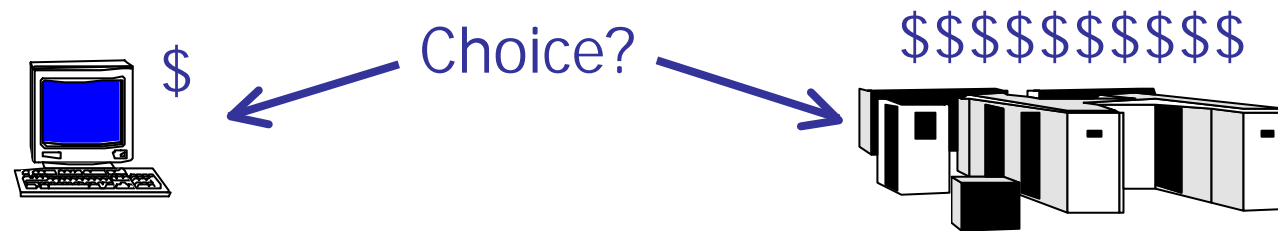
- ✍ Grid上で計算資源とともに情報資源を提供するシステム
  - ✍ e.g. Ninf, NetSolve, Nimrod
- ✍ クライアントサーバアーキテクチャ
- ✍ RPCベースのプログラミングモデル
- ✍ EP型科学・工学アプリケーションとの親和性高い
  - ✍ 分子生物学, 遺伝子情報, オペレーションリサーチ

複数クライアント・複数サーバでのスケジューリング？

# NESスケジューリング

## ✎ 経済モデル - Grid通貨による課金システム

(E.g. [Zhao and Karamcheti '00], [Plank '00],  
[Buyya '00])



? 実環境での経済モデルは確立していない

## ✎ Nimrod [abramson '00] システムによる

### デッドラインスケジューリングの評価

ユーザがジョブのデッドラインを指定し,そのデッドライン  
までにジョブを終了させる



# 我々のアプローチ

---

- ✎ Grid上の複数クライアントがデッドラインを要求するジョブを投入
- ✎ **簡単な経済モデル**下でのデッドラインスケジューリング手法
  - ✎ デッドラインスケジューリングの失敗率の低下
  - ✎ 低資源コスト
- ✎ **Bricks**システムでのシミュレーションによる評価
  - Gridスケジューリングのための性能評価システム



# 発表内容

---

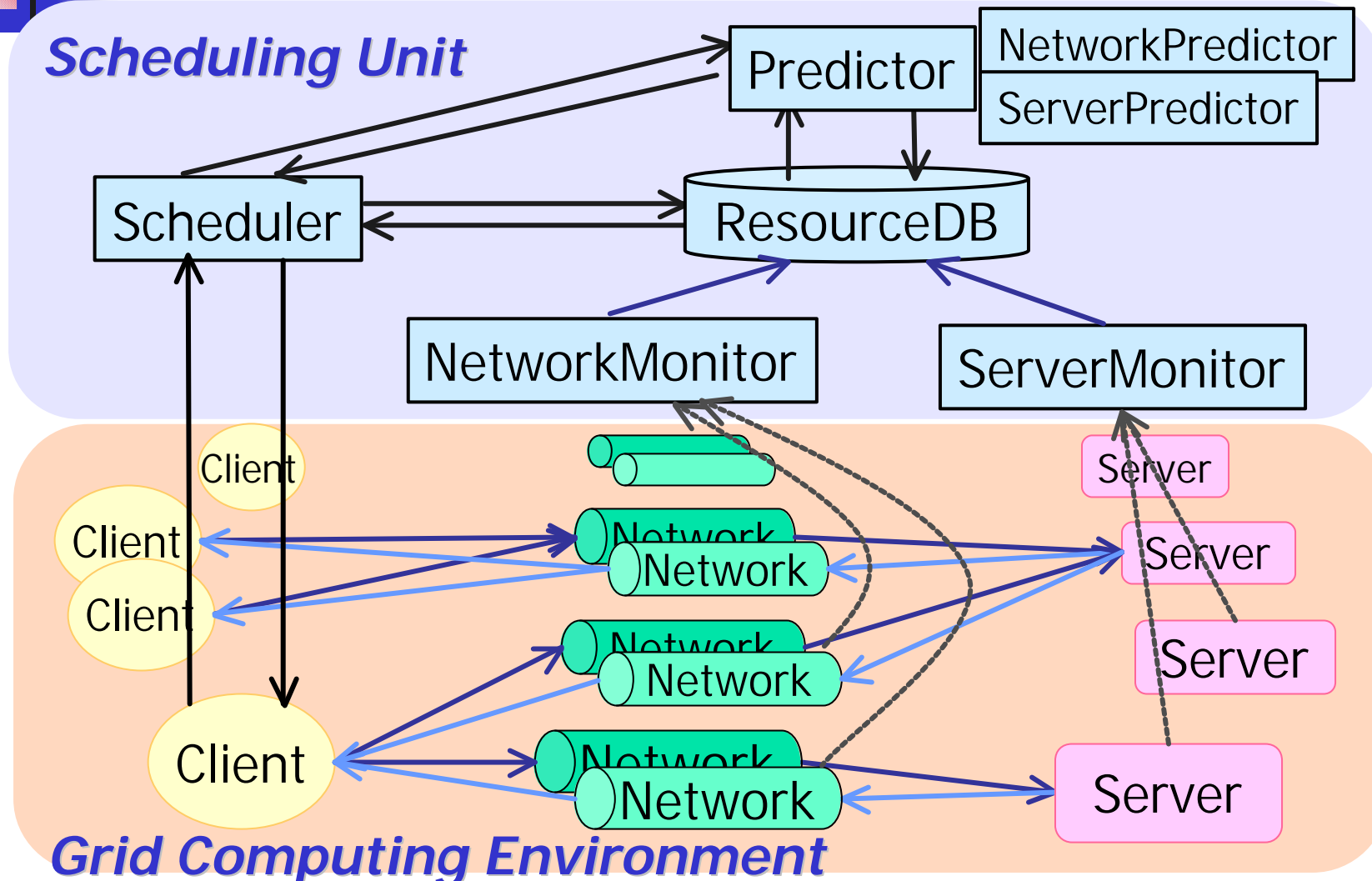
- ✎ Bricksの概要とその拡張
  - ✎ スケーラブルで現実的なシミュレーション
- ✎ デッドラインスケジューリングアルゴリズム
  - ✎ Load Correctionメカニズム
  - ✎ Fallbackメカニズム
- ✎ Bricksでの複数クライアント・サーバ環境での評価
  - ✎ 資源負荷 , 資源コスト , 資源予測精度に対する conservativeness , アルゴリズムの有効性



# Bricks :Gridスケジューリング評価システム

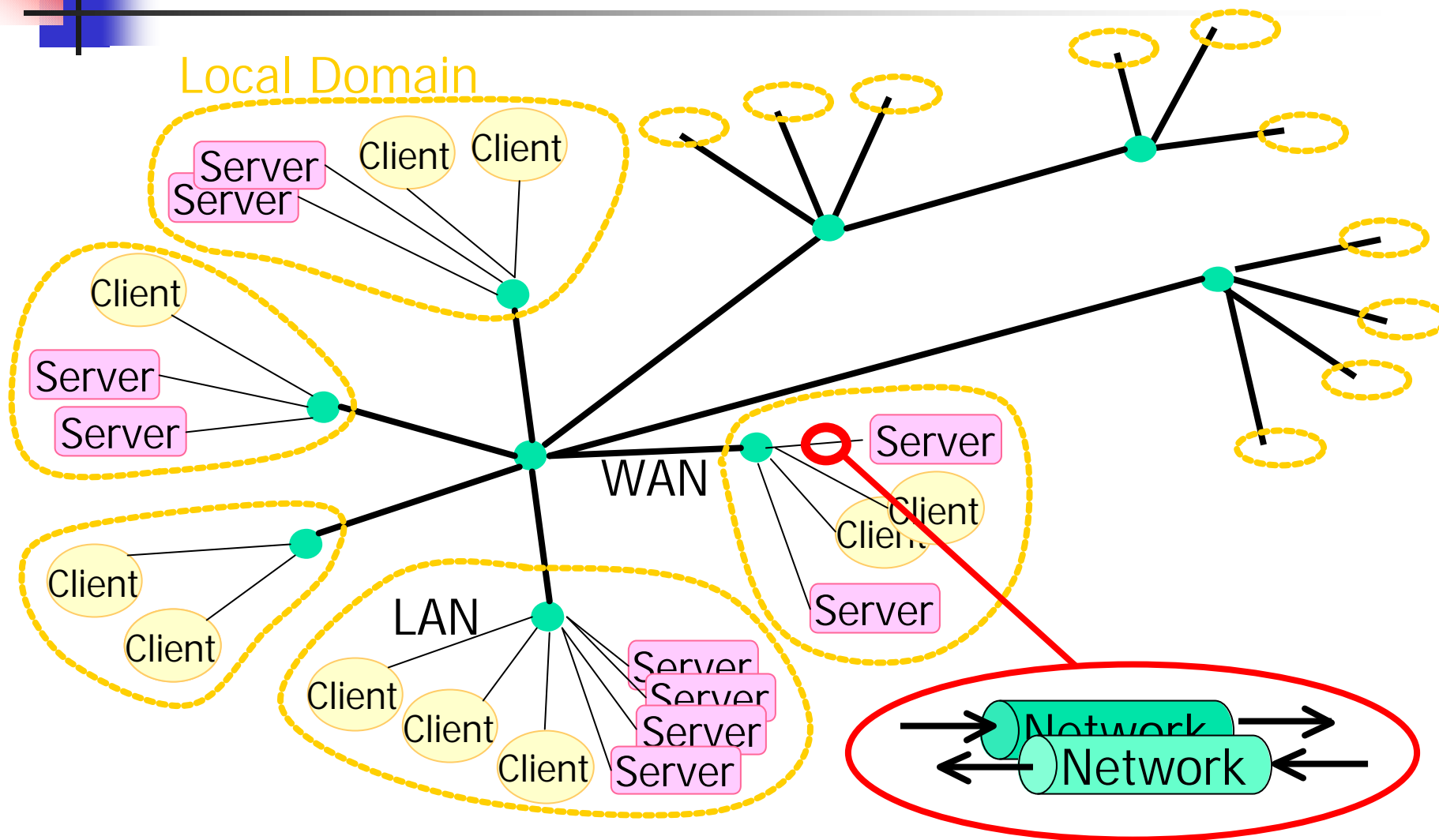
- ✎ 評価用シミュレーションフレームワーク
  - ✎ スケジューリングアルゴリズム
  - ✎ スケジューリングフレームワークコンポーネント (e.g. 予測モジュール)
- ✎ 評価のための様々な機能を提供
  - ✎ 再現性があり,制御が容易な評価環境
  - ✎ 柔軟なシミュレーション環境設定 (e.g. Gridトポロジ, 資源モデル, クライアントモデル)
  - ✎ 既存Gridコンポーネントのための評価環境 (e.g., NWS forecaster [HPDC99])

# Bricksシステムアーキテクチャ

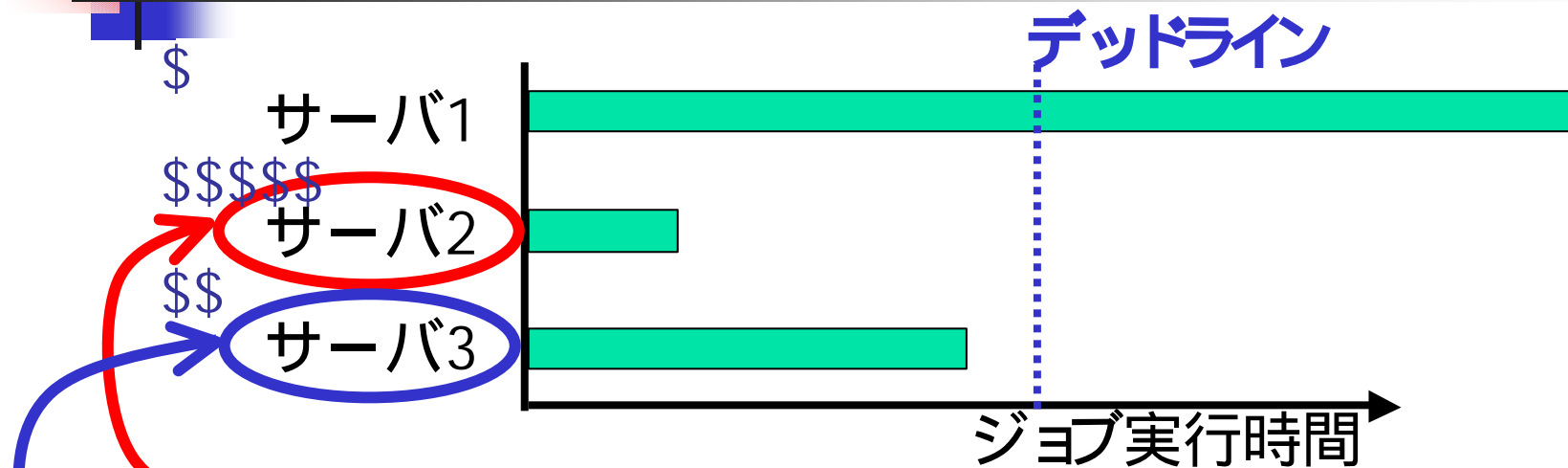




# 木構造ネットワークモデル



# デッドラインスケジューリング



既存NESスケジューラ：

ジョブの実行時間の最短化を目指す? Greedy

デッドラインスケジューリング：

各ジョブのデッドラインまでにジョブを終了させる

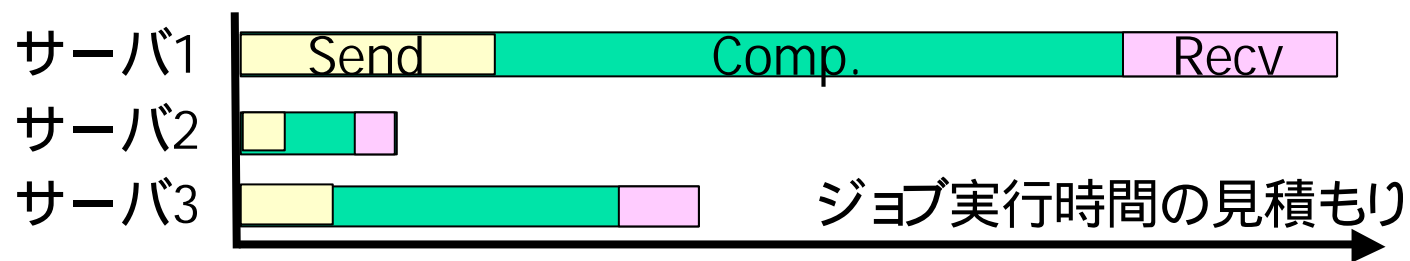
# デッドラインスケジューリング アルゴリズム

- 1 各サーバ $S_i$ でのジョブ処理時間 $T_{si}$ の見積もり:

$$T_{si} = W_{send}/P_{send} + W_{recv}/P_{recv} + W_s/P_{serv} \quad (0 \leq i < n)$$

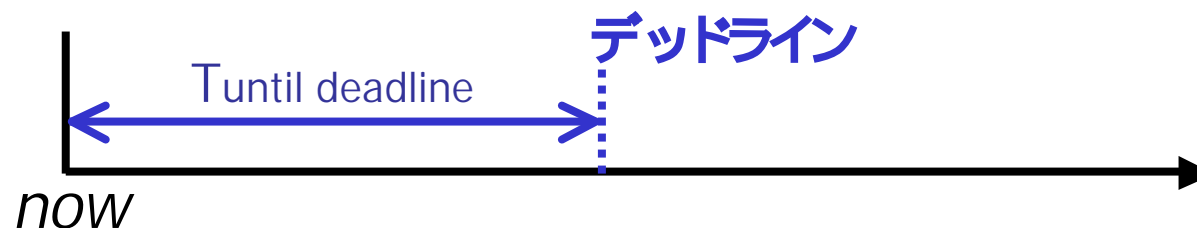
$W_{send}, W_{recv}, W_s$ : send/recv データ量, 論理演算数

$P_{send}, P_{recv}, P_{serv}$ : send/recvスループット, サーバ性能の予測値



- 2  $T_{\text{until deadline}}$ の算出:

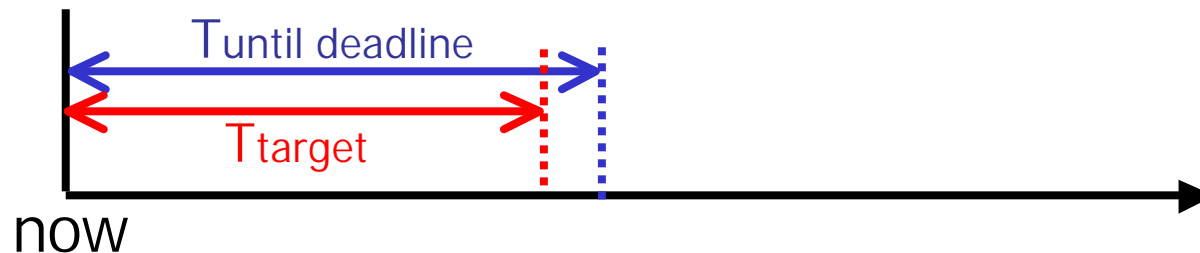
$$T_{\text{until deadline}} = T_{\text{deadline}} - \text{now}$$



# デッドラインスケジューリング アルゴリズム (Cont.)

3 ジョブ処理時間の目標値  $T_{target}$  の算出 :

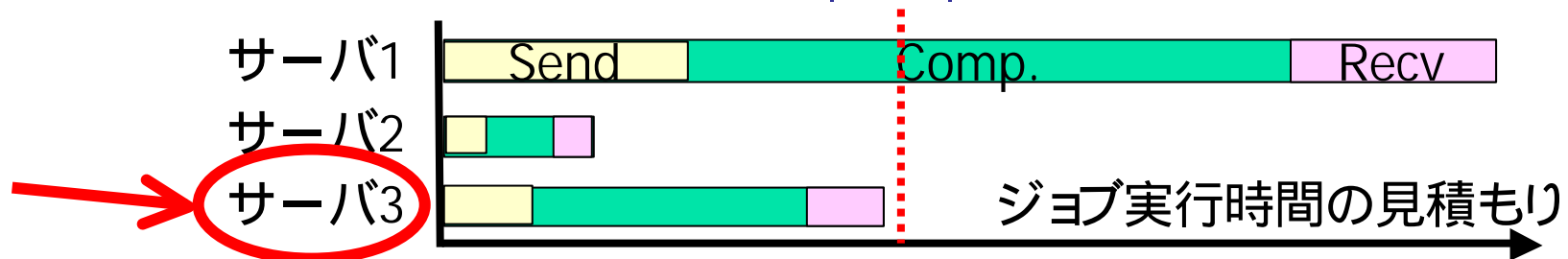
$$T_{target} = T_{until\ deadline} \times Opt \quad (0 < Opt \leq 1)$$



4 適切なサーバ  $S_i$  の選択

条件 :  $MinDiff = \min(Diff_{S_i})$  where  $Diff_{S_i} = T_{target} - T_{S_i} \geq 0$

Otherwise  $\min(|Diff|)$





# デッドラインスケジューリングの 失敗要因

---

- ✎ 予測精度の保証なし
- ✎ モニタリングシステムは急激な負荷の変化の察知困難
- ✎ FCFSの場合サーバへの到着順の変化
- ? 負荷予測値の補正 (Load Correction)
- ? サーバ自身にスケジューリングの機能を追加 (Fallback)

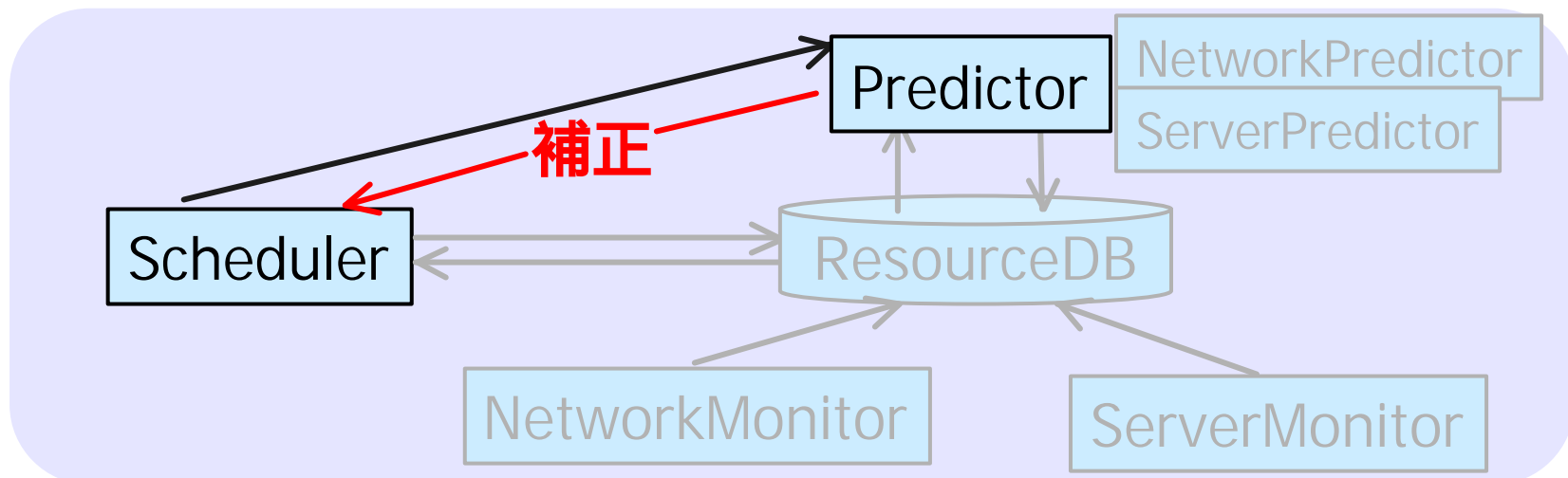
# Load Correctionメカニズム

- スケジューリング結果から資源予測値を補正
- 予測値  $Loads_i$  の補正 :

$$Loads_{i \text{ corrected}} = Loads_i + N_{jobs \ Si} \times pload$$

$N_{jobs \ Si}$ : サーバ  $S_i$  に割り当てられた 処理が終了していないジョブの総数

$Pload (= 1)$ : 補正の度合いを決定する任意の値



# Fallbackメカニズム

- サーバ自身にスケジューリングの機能を追加
- サーバで到着したジョブがデッドラインに間に合うか判断

Fallbackの条件：

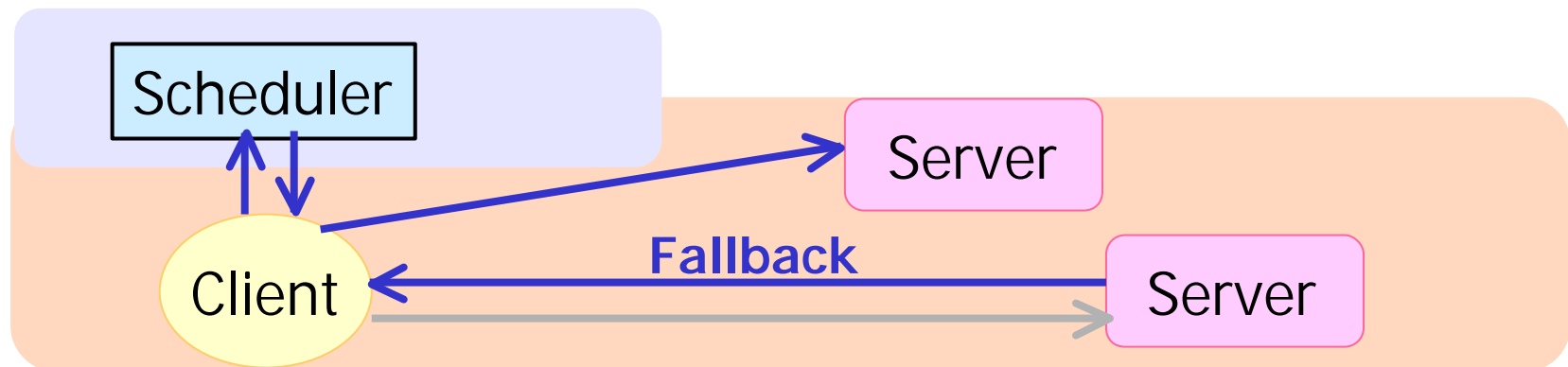
$T_{\text{until deadline}} < T_{\text{send}} + E_{\text{Texec}} + E_{\text{Trecv}} \ \&\&$

$N_{\text{max. fallbacks}} ? N_{\text{fallbacks}}$

$T_{\text{send}}$  : send通信時間

$E_{\text{Texec}}, E_{\text{Trecv}}$ : recv通信時間, 計算時間の見積もり値

$N_{\text{fallbacks}}, N_{\text{max. fallbacks}}$  : ジョブのfallback総数, 最大fallback数





# 性能評価

✎ 複数クライアント・サーバを想定したBricks上で評価

✎ 資源負荷, 資源コスト, 資源予測精度に対する conservativeness, アルゴリズムの有効性

✎ 性能指標

✎ 失敗率: デッドラインまでに処理が終了しなかったジョブの割合

✎ 資源コスト: 全てのジョブの平均資源コスト

コスト = サーバの性能

e.g. 2つのジョブに 100 [Mops/s] と 300 [Mops/s] のサーバ

? コスト = 200



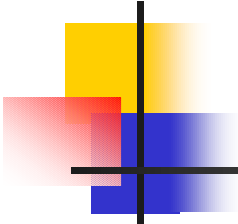


# 評価環境

---

## 📌 スケジューリングアルゴリズム

- 📌 Greedy :既存NESスケジューリング手法
- 📌 Deadline (Opt = 0.5, 0.6, 0.7, 0.8, 0.9)
- 📌 Load Correction (on/off)
- 📌 Fallback ( $N_{\max}$  fallbacks = 0/1/2/3/4/5)



# Bricksシミュレーション設定

## Grid Computing Environment (?75 ノード, 5 Grids)

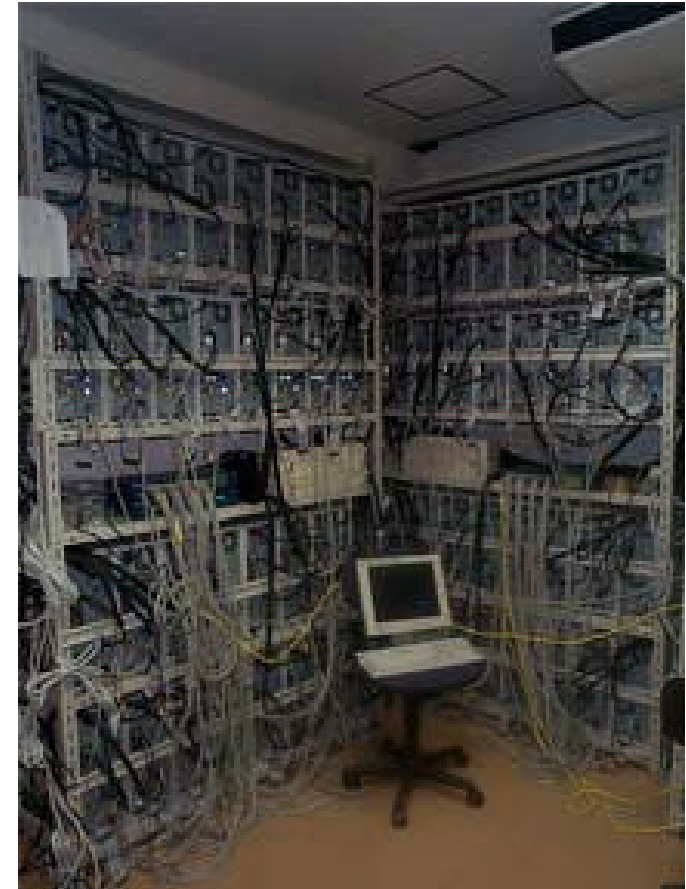
- ローカルドメイン数 :10
- ローカルドメインノード数 :5-10
- 平均 LAN バンド幅 :50-100[Mbits/s]
- 平均 WAN バンド幅 :500-1000[Mbits/s]
- 平均サーバ性能 :100-500[Mops/s]
- 平均サーバ負荷 :0.1

## クライアントジョブ

- send/recvデータ量 :100-5000[Mbits]
- 平均演算数 :1.5-1080[Gops]
- 平均ジョブ投入間隔 :  
60(負荷 :高), 90(負荷 :中), 120(負荷 :低) [sec]

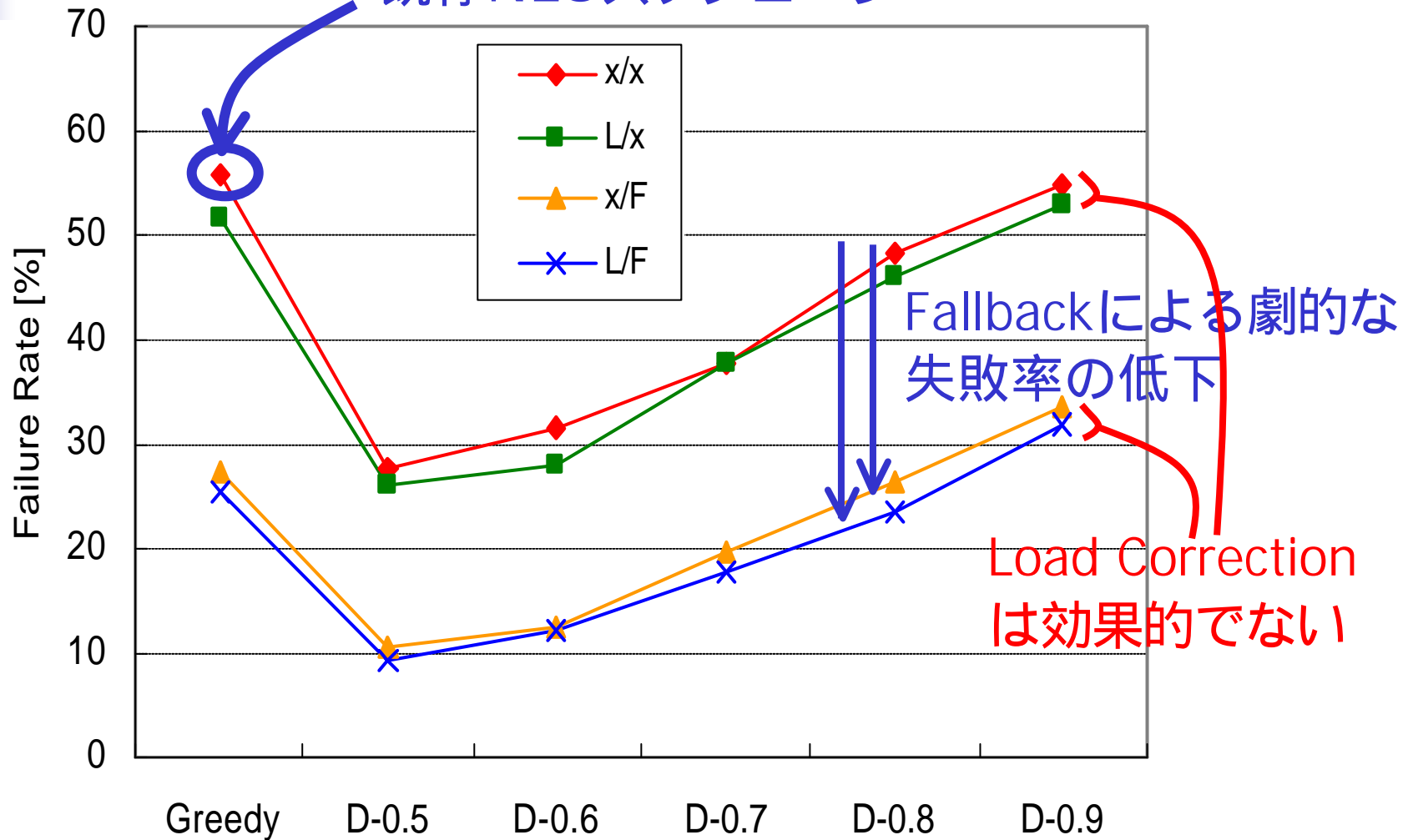
# シミュレーション実行環境

- ✎ Presto II クラスタ:
  - 東工大松岡研究室 66PE
    - ✎ Dual Pentium III 800MHz
    - ✎ メモリ: 640MB
    - ✎ ネットワーク: 100Base/TX
- ✎ BricksシミュレーションをAPSTを用いてEP実行
- ✎ 24 時間シミュレーション x 2,500  
(1 シミュレーション :30-60 [min],  
Sun JVM 1.3.0+HotSpot)



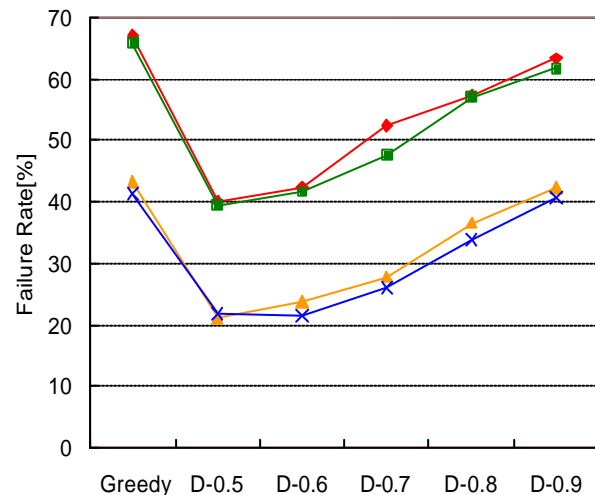
# 失敗率の比較 (負荷 : 中)

既存NESスケジューラ

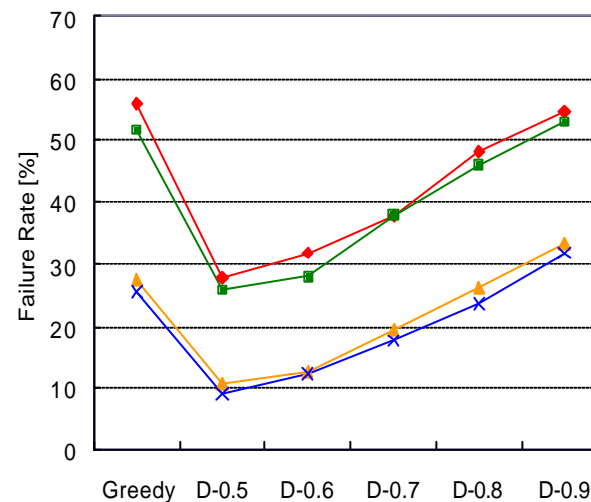


# 失敗率の比較(負荷 :高,中,低)

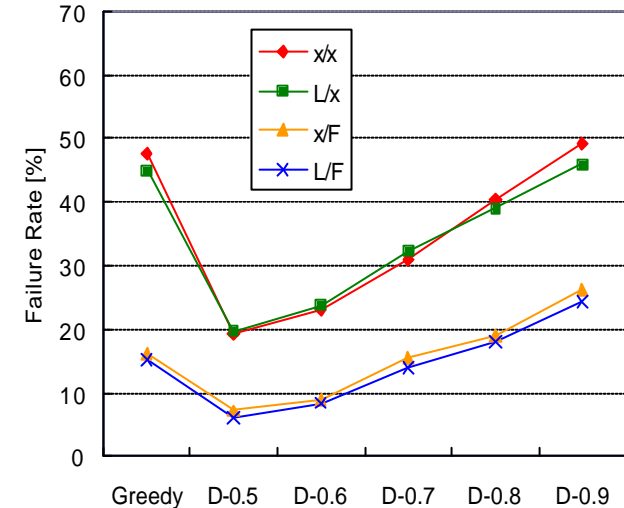
負荷 :高



負荷 :中

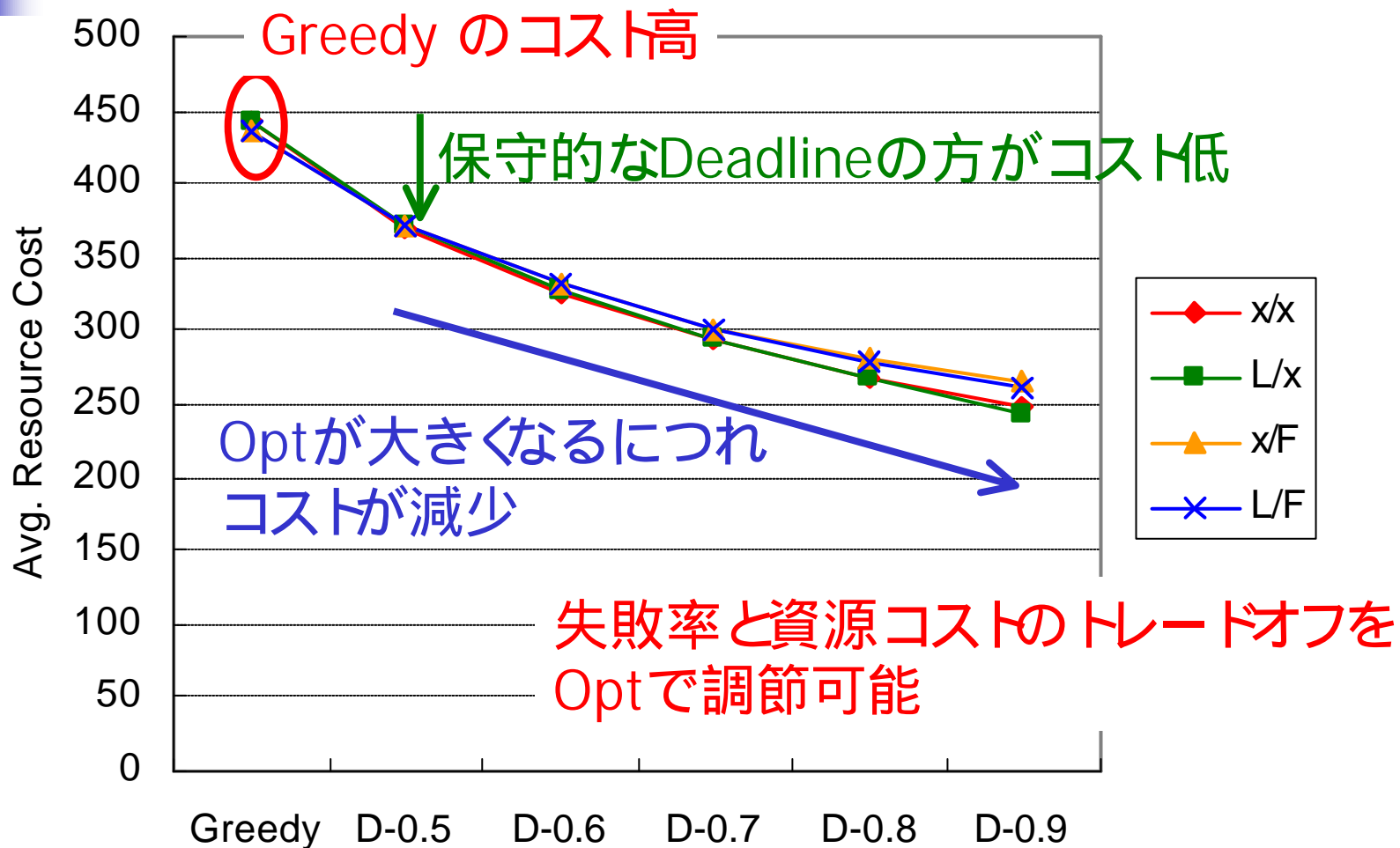


負荷 :低



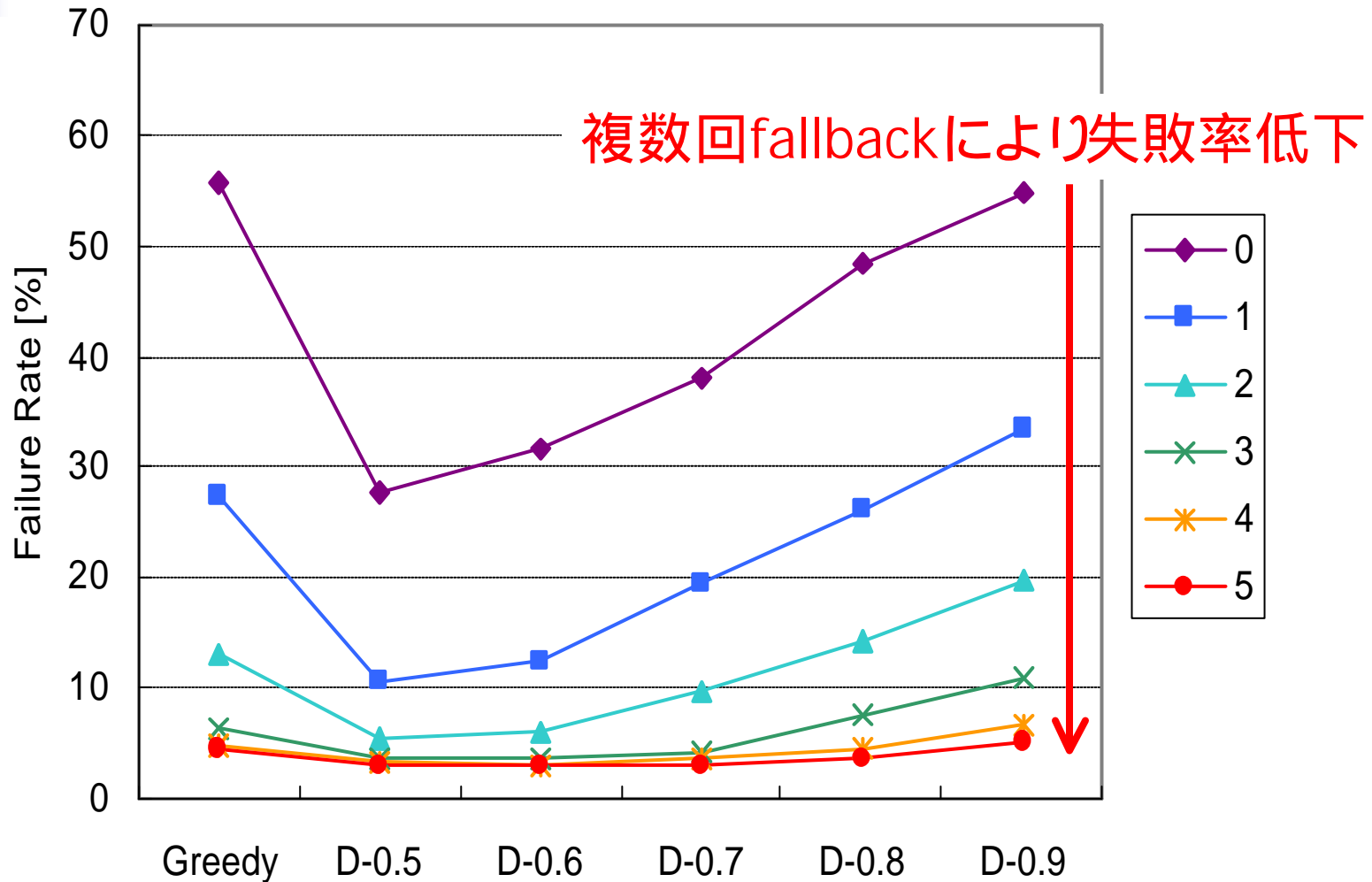
- 負荷が低くなるに連れ失敗率が低下
- 負荷が異なる場合も性能特性は類似

# 平均資源コストの比較



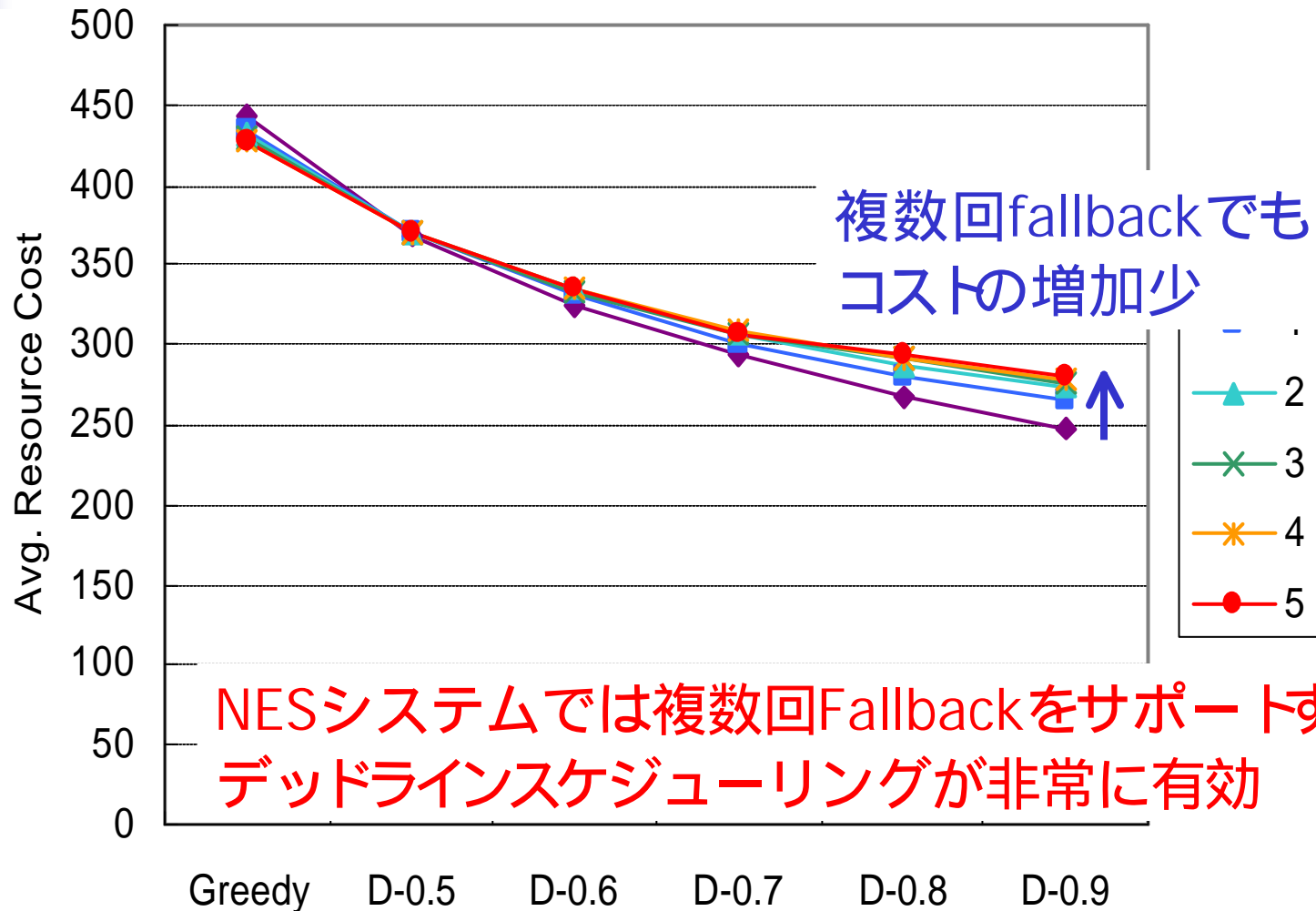
# 失敗率の比較

( $x/F$ ,  $N_{\text{max. fallbacks}} = 0, 1, 2, 3, 4, 5$ )



# 資源コストの比較

( $x/F$ ,  $N_{\max. \text{ fallbacks}} = 0, 1, 2, 3, 4, 5$ )



NESシステムでは複数回Fallbackをサポートする  
デッドラインスケジューリングが非常に有効





# 関連研究

---

## ✎ 経済モデル

- ✎ Nimrod [abramson '00]

- ✎ セルフスケジューリング

- ✎ 単一ユーザからのEPアプリケーションが対象

## ✎ Grid performance evaluation systems:

- ✎ MicroGrid [Song '00]

- ✎ 実計算機上でGlobusベースの仮想Grid環境エミュレーション

- ✎ 評価に実時間以上要す, 制御の困難さなどの問題

- ✎ Simgrid [Casanova '01]

- ✎ トレースベースの離散シミュレータ

- ✎ シミュレーションのためのプリミティブを提供

- ✎ Bricksの提供するネットワークモデルには未対応



# まとめ

---

- ✎ デッドラインスケジューリングアルゴリズムとLoad Correction・Fallbackメカニズムの提案
- ✎ 改良したBricks上で複数クライアント・サーバ環境での性能を調査
- ✎ 評価結果
  - ✎ 失敗率と資源コストのトレードオフが調整可能
  - ✎ Load Correctionは非効果的
  - ✎ 複数回Fallbackをサポートするデッドラインスケジューリングが非常に有効



## 今後の課題

---

- ✎ Bricksのより洗練された経済モデルへのサポート
- ✎ Bricks上での様々な経済モデルの評価
- ✎ 実際のNESシステムへのデッドラインスケジューリングの実装 (Ninf: <http://ninf.apgrid.org/>)