

# 性能を保証する分散実行環境のための オンラインコアロケーション手法

竹房あつ子, 中田秀基,  
工藤知宏, 田中良夫

産業技術総合研究所

# 性能保証された分散実行環境の必要性

- クラウドやグリッドなど，遠隔資源の利用が加速
  - 膨大な計算パワー，高性能計算
  - バックアップ，障害復旧
  - 特殊なデバイス，大容量データの利用
  - 遠隔共同作業，高精細遠隔会議

→分散する資源を用いた分散実行環境

- 分散実行環境の要件
  - 欲しいときに欲しい資源(計算機，ストレージ，ネットワークなど)を動的に構成
  - 必要な性能(帯域も含む)を保証←クラウド/グリッドと異なる
    - “光パス”ネットワークで実現可能

→複数資源のコアロケーションが必要

# 分散実行環境構築の試み



- G-lambda(産総研, KDDI研究所, NTT, NICT)と EnLIGHTened Computing(米国)の共同実験 [GLIF2006, SC06]
- 日米間に跨る計算・ネットワーク資源の動的確保

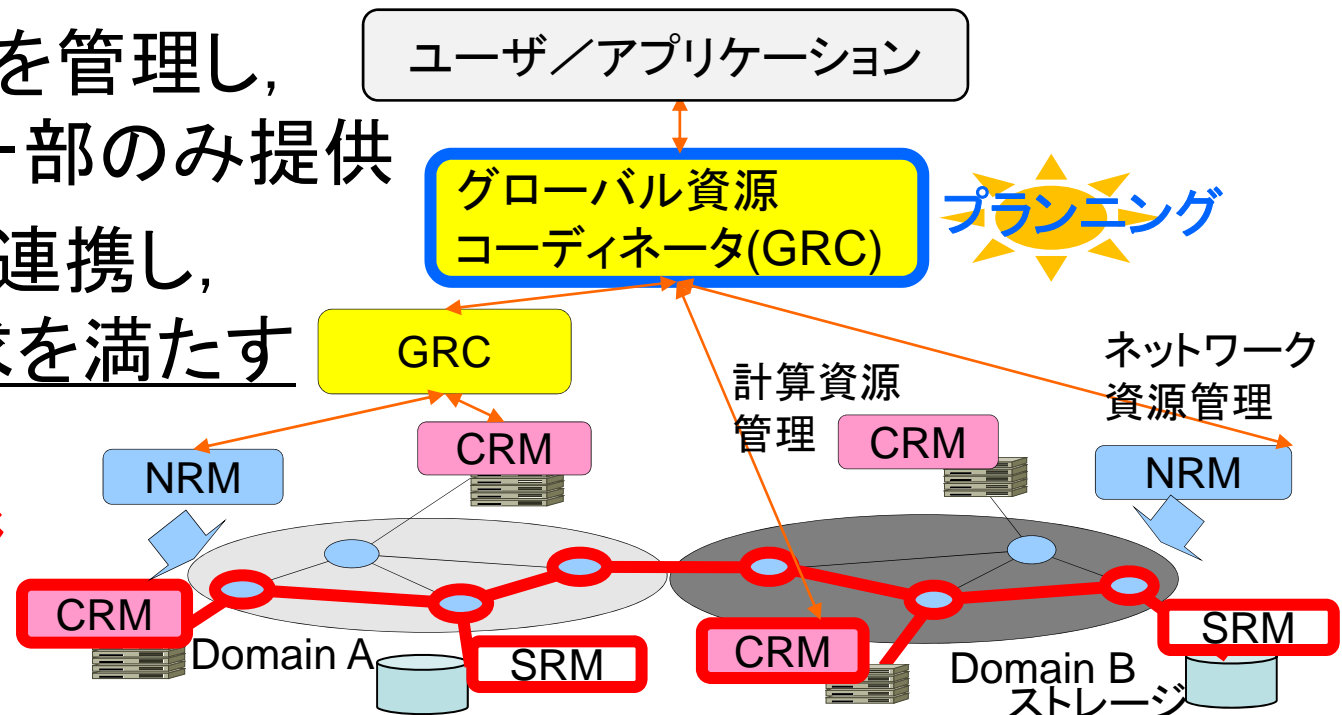
# 前提とするコアロケーションモデル

- 事前予約. オンライン処理(↔バッチ)
- 各資源は複数の組織が提供(商用サービス含む)
- 各資源の資源マネージャ(RM)とグローバル資源コーディネータ(GRC)で構成

- RMが予約表を管理し,  
資源情報は一部のみ提供

- **GRC**がRMと連携し,  
ユーザの要求を満たす  
資源を選択

資源のトポロジ  
も考慮



# 計算資源とネットワークの“帯域”の コアロケーションの課題

- 資源群のトポロジ
  - 複数のネットワーク管理ドメインが存在
  - 計算機だけでなく、その間のネットワークも必要
- コアロケーション方針の反映
  - ユーザの観点
    - 早い時刻に確保
    - 価格の安い資源群を確保
    - 品質優先で確保
  - 資源管理の観点
    - 複数資源提供者に対して平等な負荷の分配
    - 特定の資源への優先割り当て(省エネ, 連携関係)
    - サービスレベルへの配慮

→最適化問題(線形計画(LP))  
にモデル化し,  
資源選択する手法を提案

# オンライン処理における課題

- オンライン処理
  - ウェブからパック旅行の予約手続きをするように、ユーザの期待する資源を確保・提供→**応答時間が重要**
- コアロケーションに要する時間が許容できるか？
  - スケジューリング問題はNP困難
  - 最適化問題(LP)では、問題サイズにより指数的に求解時間が長くなる
  - **LPを用いたコアロケーション手法の妥当性？**

# 本研究の成果

- 性能保証する分散実行環境のための  
オンラインコアロケーション手法の提案
  - コアロケーション問題を最適化問題としてモデル化
  - コアロケーション方針が反映可能な手法の提案
  - 求解時間の短縮手法の提案
- 機能性の評価
  - 計算・ネットワーク資源のコアロケーションが可能
  - 資源管理の観点でコアロケーション方針が反映可能
- 実用性の評価
  - LPソルバを用いた提案手法の求解時間が許容可能

# 発表概要

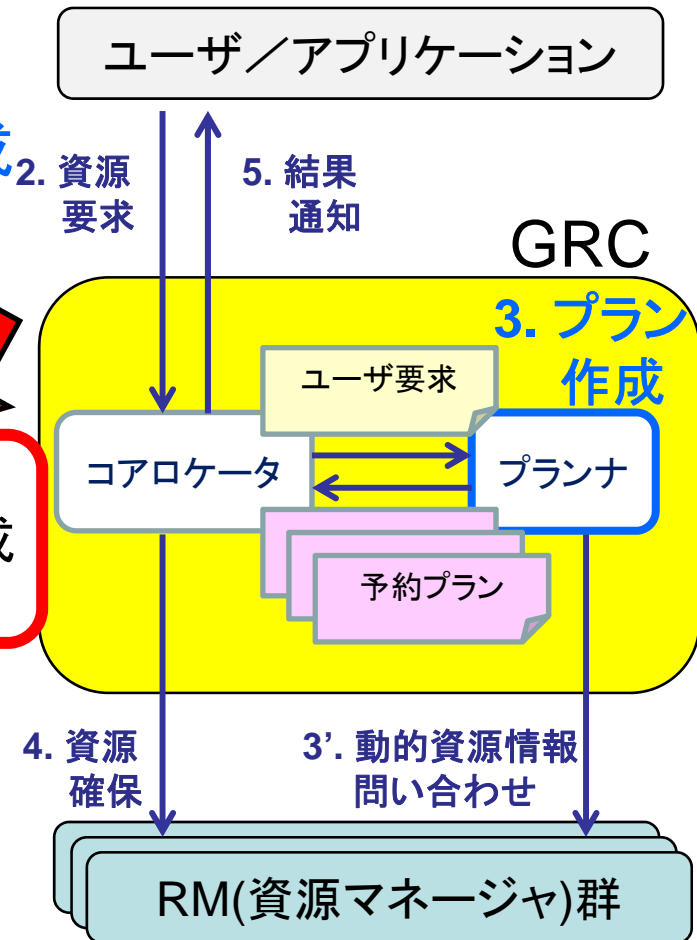
- 提案コアロケーション手法の概要
  - 最適化問題としてモデル化
  - 求解時間の短縮手法の提案
  - コアロケーション方針の反映
- 機能性の評価
  - シミュレーション環境で提案手法の有効性を実証
- 実用性の評価
  - アルゴリズム, ソルバの違いによる求解時間の比較
- 関連研究
- まとめと今後の課題

# 発表概要

- 提案コアロケーション手法の概要
  - 最適化問題としてモデル化
  - 求解時間の短縮手法の提案
  - コアロケーション方針の反映
- 機能性の評価
  - シミュレーション環境で提案手法の有効性を実証
- 実用性の評価
  - アルゴリズム, ソルバの違いによる求解時間の比較
- 関連研究
- まとめと今後の課題

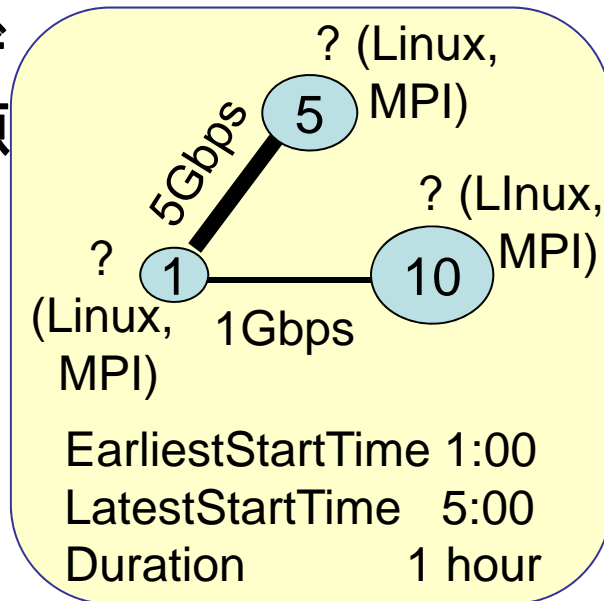
# コアロケーション手法の概要

1. 静的資源情報をあらかじめ取得
2. ユーザの資源要求を受け取る
3. **GRCのプランナが複数予約プラン作成**
  - 3a. 指定した時間帯から予約時間帯候補をN個選ぶ
  - 3b. 関連するRMに対し、N個の時間帯の動的情報を取得
  - 3c. 3bの情報から、**コアロケーションアルゴリズム**を用い、 $n(\leq N)$ 個の予約プランを作成  
→最適化問題にモデル化
  - 3d.  $n$ 個の予約プランの優先順位を決定
4. 予約プランをもとに資源確保
5. 確保成功 / 失敗をユーザに通知  
失敗の場合、ユーザは条件を変えて再度要求

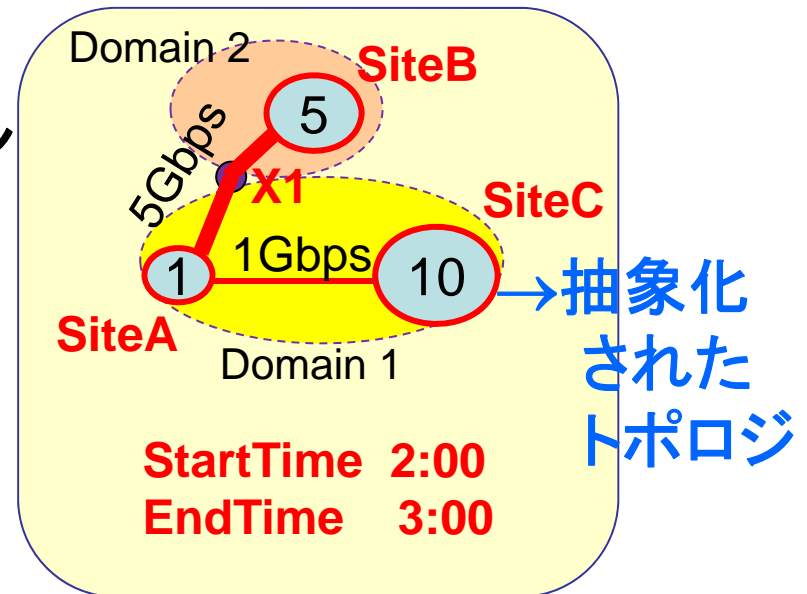
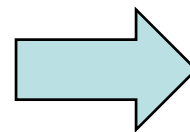


# ユーザの資源要求と予約プラン

ユーザの資源要求



予約プラン



- 資源要求パラメータ

- 計算資源: CPU/コア数(以降CPU数), 属性情報(OS)
- ネットワーク資源: 帯域, 遅延, 属性情報
- 時刻: 直接指定 / 時間帯指定(→予約時間帯候補を選択)

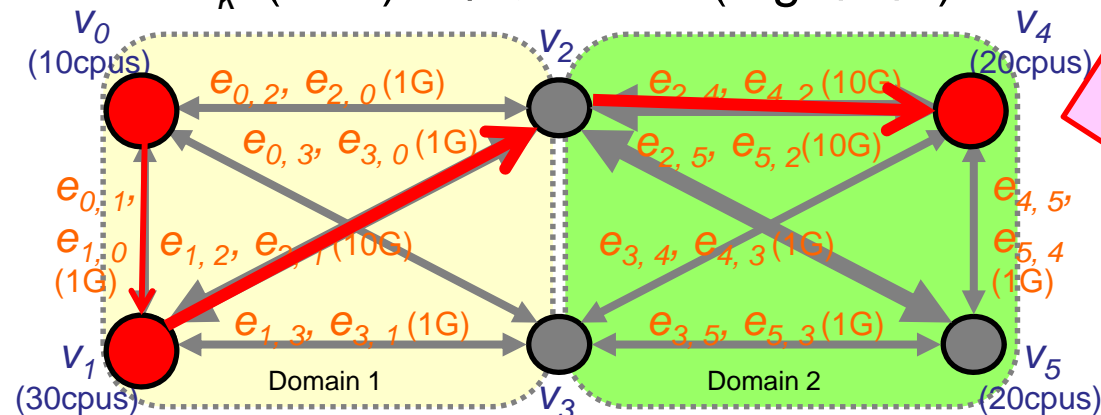
# 資源群と資源要求のモデリング

**資源群**: 有向グラフ(双方向)  $G=(V, E)$

- $V$ :  $G$ の点の集合,  $E$ :  $G$ の枝の集合
- $v_q$ : 計算資源サイトまたはネットワークドメイン交換点
- $e_{o,p}$ : 資源または交換点間のパス ( $o$ : 始点,  $p$ : 終点)

**資源パラメータ** (枝は双方向で共有)

- $wc_i$  ( $i \in V$ ): CPU数,  $wb_k$  ( $k \in E$ ): 帯域
- $vc_i$  ( $i \in V$ ): CPU数の重み(e.g. 価格),  $vb_k$  ( $k \in E$ ): 帯域の重み(e.g. 価格)



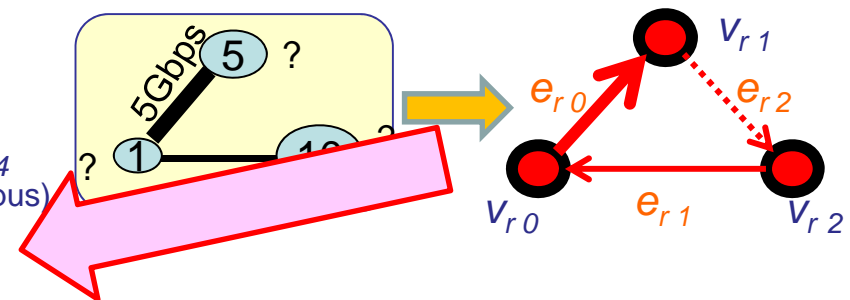
**資源要求**: 完全有向グラフ

$G_r=(V_r, E_r)$

- $V_r$ : 要求資源サイトの集合
- $E_r$ :  $V_r$ 間を結ぶ枝の集合

**資源要求パラメータ**

- $rc_j$  ( $j \in V_r$ ):  $V_r$ に必要なCPU数
- $rb_l$  ( $l \in E_r$ ):  $E_r$ に必要な帯域



# 求める解 = 予約プラン

$X$  (計算機サイト):

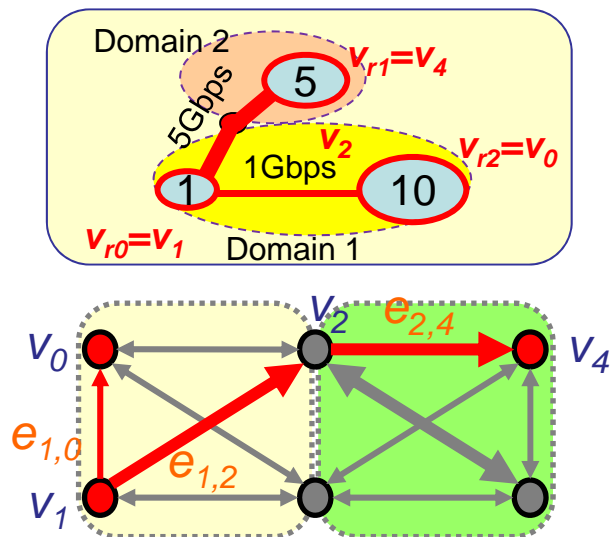
$$x_{i,j} \in \{0, 1\} \quad (i \in V, j \in V_r) \quad (1)$$

$Y$  (ネットワークパス):

$$y_{k,l} \in \{0, 1\}$$

$$(k=(m, n) \in E, m, n \in V, \quad (2)$$

$$l=(o, p) \in E_r, o, p \in V_r)$$



資源  
要求

|       | $V_0$    | $V_1$            | $V_2$ | $V_3$ | $V_4$ | $V_5$ |     |     |
|-------|----------|------------------|-------|-------|-------|-------|-----|-----|
| $X =$ | $v_{r0}$ | 0, 1, 0, 0, 0, 0 |       |       |       |       | 資源群 |     |
|       | $v_{r1}$ | 0, 0, 0, 0, 1, 0 |       |       |       |       |     |     |
|       | $v_{r2}$ | 1, 0, 0, 0, 0, 0 |       |       |       |       |     |     |
| $Y =$ | $e_{r0}$ | 0                | 0     | 1     | 0     | 1     | 0   | ... |
|       | $e_{r1}$ | 0                | 1     | 0     | 0     | 0     | 0   | ... |
|       | $e_{r2}$ | 0                | 0     | 0     | 0     | 0     | 0   | ... |

線形計画法(LP)で求める

# 目的関数と制約条件

## 目的関数

Minimize

$$\sum_{i \in V, j \in V_r} v c_i \cdot r c_j \cdot x_j + \sum_{k \in E, l \in E_r} v e_k \cdot r b_l \cdot y_{k,l} \quad (3)$$

## 制約条件

Subject to

$$\forall j \in V_r, \sum_{i \in V} x_{i,j} = 1 \quad (4)$$

$$\forall i \in V, \sum_{j \in V_r} x_{i,j} \leq 1 \quad (5)$$

$$\forall i \in V, \sum_{j \in V_r} r c_j \cdot x_{i,j} \leq w c_i \quad (6)$$

$$\forall l \in E_r, \sum_{k \in E_r} y_{k,l} \begin{cases} \geq 1 & (r b_l \neq 0) \\ = 0 & (r b_l = 0) \end{cases} \quad (7)$$

$$\forall k \in E, \sum_{l \in E_r} r b_l \cdot y_{k,l} \leq w b_k \quad (8)$$

$$\forall l = (o, p) \in E_r, \forall m \in V, \sum_{n \in V, m \neq n} y_{(n,m),(o,p)} - \sum_{n \in V, m \neq n} y_{(m,n),(o,p)} = \begin{cases} x_{m,o} - x_{m,p} & (r b_l > 0) \\ 0 & (r b_l = 0) \end{cases} \quad (9)$$

(3) 計算・ネットワーク資源の重みを最小化

(4),(5),(6)はX(計算機)の制約

(4) 選択サイトは1つのみ

(5) サイトは1回以上選択されない

(6) 要求CPU数が提供可能

(7),(8)はY(ネットワーク)の制約

(7) 要求パスがある場合  $y_{k,l}$  の総和は1以上, ない場合0

(8) 要求帯域が提供可能

(9)はX,Y両方の制約

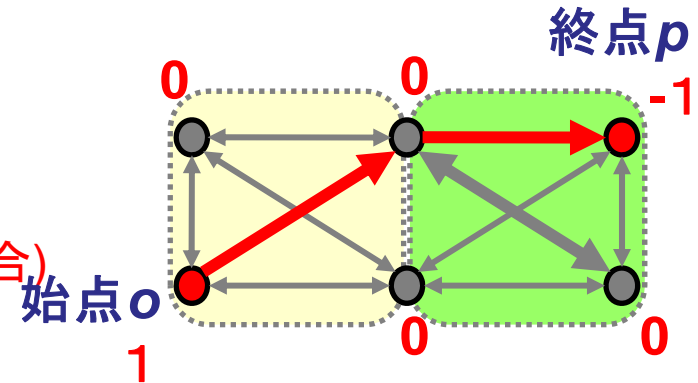
(9) 流量保存則から導く

# 流量保存則の応用

$$\forall l = (o, p) \in E_r, \forall m \in V,$$

$$\sum_{n \in V, m \neq n} y(n, m), (o, p) - \sum_{n \in V, m \neq n} y(m, n), (o, p) = \begin{cases} x_{m, o} - x_{m, p} & (r_{bl} > 0) \\ 0 & (r_{bl} = 0) \end{cases} \quad (9)$$

流出量総和 流入量総和 0/1/-1(要求帯域がある場合)



- 流量保存則

- グラフ上の2点間の経路上の各点において, 流入する流量  $f$  と流出する流量  $f'$  の差が供給量と等しくなる

- 流量保存則の応用

- 要求ネットワーク  $l = (o, p)$  を, 要求サイト  $o$  から  $p$  へのフローと考える
  - 左辺: 流量を1として供給量を算出
  - 右辺:  $m_0, m_1 \in V$  に対し,  
 選択サイトならば  $x_{m_0} = 1, x_{m_1} = 1$

予約なしのコアロケーションでも適用可能

# コアロケーションアルゴリズムの改良

- 最適化問題では, 求める変数の数が増えると求解時間が指数的に増大  
 → 制約条件を追加し, 冗長パスの探索を枝刈り

## 制約条件

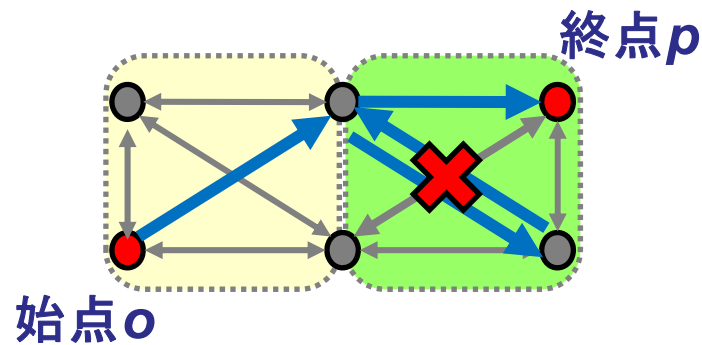
Subject to

$$\forall l \in E_r, \forall m, n \in E (m \neq n), y_{(m,n),l} + y_{(n,m),l} \leq 1 \quad (10)$$

$$\forall l \in E_r, \sum_{k \in E} y_{k,l} \leq P_{\max} \quad (11)$$

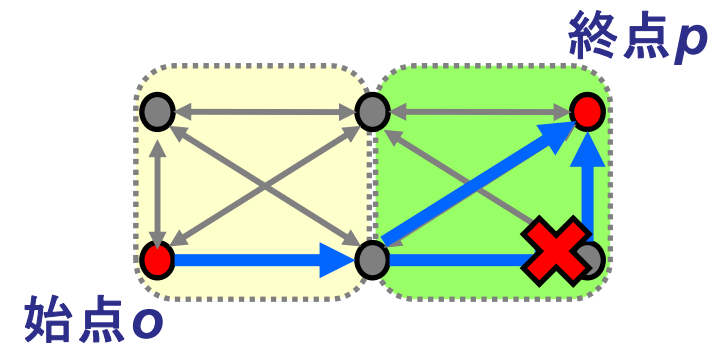
# 新たな制約条件

(10) 1つのネットワークに対し, 双方向の枝を同時に選択しない



(11) 1つのネットワークを構成する枝の数の最大値  $P_{max}$  を指定

$P_{max} = 2$  のとき



→最適解でない場合もある

# コアロケーション方針の反映

- | ユーザ                    | 適用方法                         |
|------------------------|------------------------------|
| (a) 時刻優先               | →手順3dで順位付け                   |
| (b) 価格優先               | →式(3)の目的関数と手順3d              |
| (c) 品質優先               | →式(3)の目的関数の変更と手順3d           |
| <b>資源管理者</b>           |                              |
| (A) 複数RMへの平等な<br>負荷分配  | →資源群のモデリングで重みを追加<br>と目的関数を追加 |
| (B) 特定資源の優先            | →資源群のモデリングで重みを追加<br>と目的関数を追加 |
| (C) ユーザのサービス<br>レベルの考慮 | →手順3bで動的資源情報のフィルタ            |

# 発表概要

- 提案コアロケーション手法の概要
  - 最適化問題としてモデル化
  - 求解時間の短縮手法の提案
  - コアロケーション方針の反映
- 機能性の評価
  - シミュレーション環境で提案手法の有効性を実証
- 実用性の評価
  - アルゴリズム, ソルバの違いによる求解時間の比較
- 関連研究
- まとめと今後の課題

# 機能性の評価

- シミュレーションによる評価
- 提案手法の有効性の検証
  - 予約成功率の調査
- コアロケーション方針の反映
  - 資源管理者の方針  
(A), (B), (C)の適用

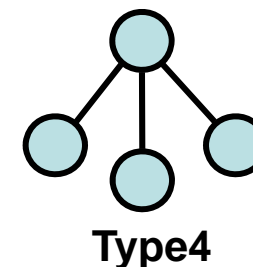
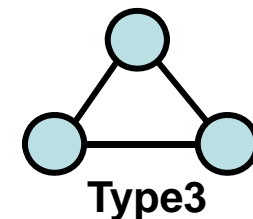
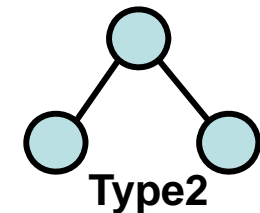
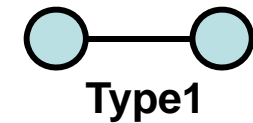
## コアロケーション方針

- ユーザ
  - (a) 時刻優先
  - (b) 価格優先
  - (c) 品質優先
- 資源管理者
  - (A) 複数RMへの平等な負荷分配
  - (B) 特定資源の優先
  - (C) ユーザのサービスレベルの考慮



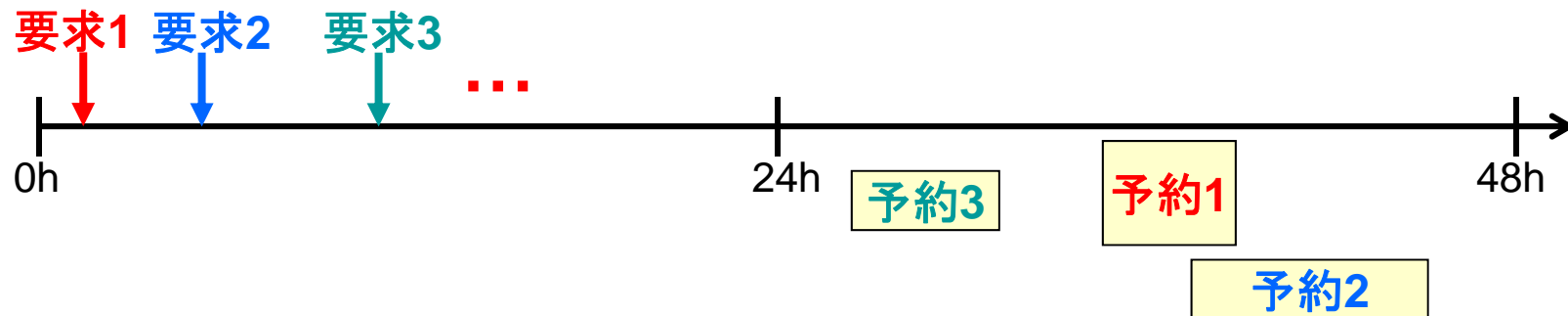
# 評価シミュレーション条件

| 環境設定            |  |
|-----------------|--|
| 資源構成            | GRM=1, NRM=3, CRM=10                               |
| サイト数/ドメイン       | 4/N, 3/S, 3/U (実環境を想定)                             |
| ドメイン交換点         | X1{N, S,U}, X2{N, S}                               |
| CPU数/重み         | N{8, 16, 32, 64}, S{8, 16, 32},<br>U {8, 16, 32}/1 |
| 帯域 [Gbps] / 重み  | ドメイン内=5/5, 交換点接続=10/3                              |
| 資源要求設定          |  |
| ユーザ             | UserA, UserB                                       |
| 資源要求の種類         | <b>Type 1, 2, 3, 4</b> (一様分布)                      |
| 平均要求到着間隔        | ポアソン到着(負荷は10~100[%])                               |
| 予約時間幅 [min] (D) | 30, 60, 120  |
| (LST+D) – EST   | 予約時間幅×3  |
| 予約CPU数          | 全サイトに対し1, 2, 4, 8 (一様分布)                           |
| 予約帯域 [Gbps]     | 全ネットワークに対し, 1                                      |



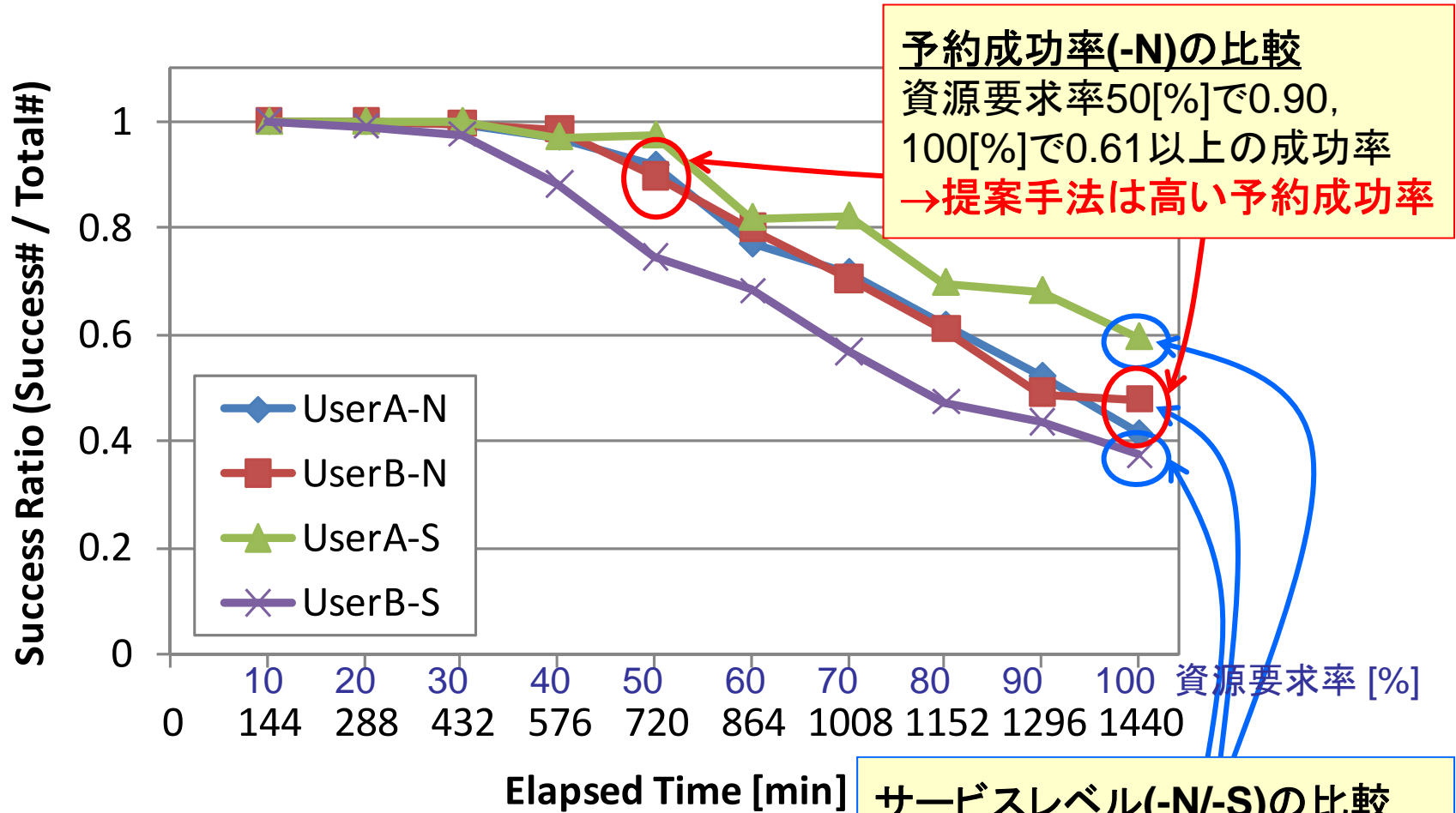
# シミュレーション方法

- 最初の24時間までに各ユーザの資源要求が到着し、次の24時間の時間帯の資源を予約(一様分布)
  - 利用可能な資源は少なくなっていく



- 予約時間帯候補数  $N=10$  (等間隔)

# 予約成功率, (C)サービスレベルの評価

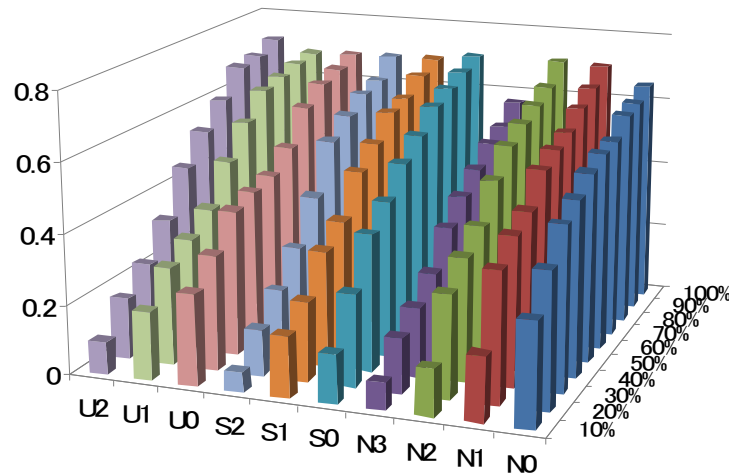


- サービスレベル(SL)の比較**
- UserBは低サービスレベル
  - 利用可能な資源がUserAの半分

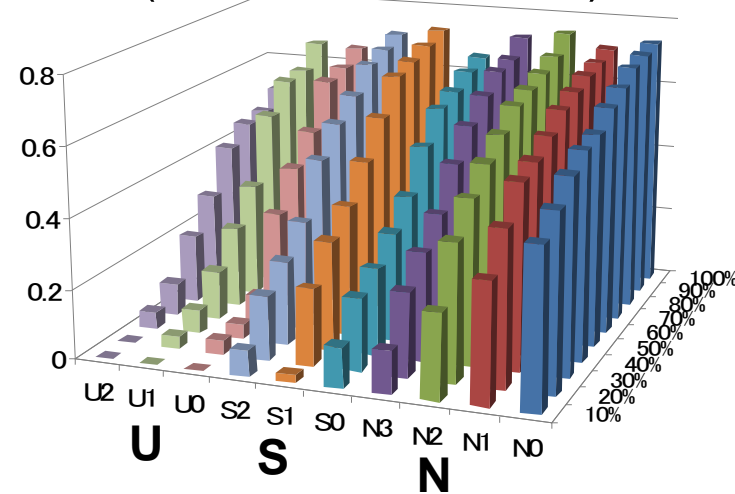
**サービスレベル(-N/-S)の比較**  
 SLなし(-N): UserA, Bは同程度  
 SLあり(-S): 100[%]のとき  
 UserAは0.60, UserBは0.37  
 →ユーザごとのSLを制御可能

# 資源管理者の要求(A)(B)の評価

(A) 平等な負荷分配

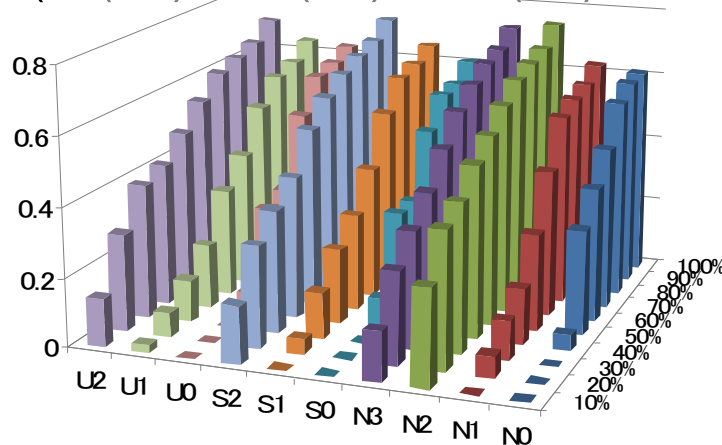


(B1) ネットワークドメインで優先  
(N → S → Uで優先)



(B2) CPU数の大きいサイト優先

(\*3(64) → \*2(32) → \*1(16) → \*0(8))



計算機の重み付けを優先度順に変更

(B1)  $N : S : U = 1 : 10 : 100$

(B2)  $*3 : *2 : *1 : *0 = 1 : 10 : 100 : 1000$

→ 資源管理者の優先度が反映可能

→ 機能性の点でも有効

# 発表概要

- 提案コアロケーション手法の概要
  - 最適化問題としてモデル化
  - 求解時間の短縮手法の提案
  - コアロケーション方針の反映
- 機能性の評価
  - シミュレーション環境で提案手法の有効性を実証
- 実用性の評価
  - アルゴリズム, ソルバの違いによる求解時間の比較
- 関連研究
- まとめと今後の課題

# 実用性の評価

- 求解時間を比較し, オンライン処理として許容できるか検証
- 制約条件の比較
  - 制約条件(10), (11)の有無
- 求解ソルバの比較
  - 線形計画法(LP)のソルバ
    - GLPK (GNU Linear Programming Kit) (フリー、遅い)
  - 充足可能性問題(SAT) のソルバ
    - MiniSat : SATのソルバ
    - Sugar++ : 最適化問題をSATに変換するソフトウェア
- 実験環境
  - CPU: Intel Core2 Quad Q9550 (2.83GHz),
  - OS: CentOS 5.0, kernel 2.6.18 x86\_64, メモリ: 4GB

# LPとSATの違い

- LP(線形計画法)
  - 1次の不等式／等式で目的関数と制約条件が定義された最適化問題
- SAT(充足可能性問題)
  - 乗法標準形(CNF)で与えられた全変数に対し, 全制約条件が満たされていればSAT, 満たされていなければ UNSATと判定
  - Sugar++[丹生]を用い, 最適化問題に対応
    - Sugar[田村]は制約充足問題(CSP)をCNFに変換
    - Sugar++は最適化問題をCSPに変換
      1. 目的関数の上限(/下限)を一時的に固定し、CSPを解く
      2. 1を繰り返し、最適な上限値(/下限値)を探索

# 求解時間の平均値, 最大値, 比率の比較

追加制約の効果大  
オンライン処理でも許容可能

MiniSat-st-1が高速  
ただし, SAT1回のみ

| 手法           | 平均値[sec] | 最大値[sec] | 標準偏差   |
|--------------|----------|----------|--------|
| GLPK         | 0.779    | 8.492    | 1.721  |
| GLPK-st      | 0.333    | 4.205    | 0.700  |
| MiniSat-st   | 12.848   | 216.434  | 27.914 |
| MiniSat-st-1 | 1.918    | 2.753    | 0.420  |

GLPKの方が高性能

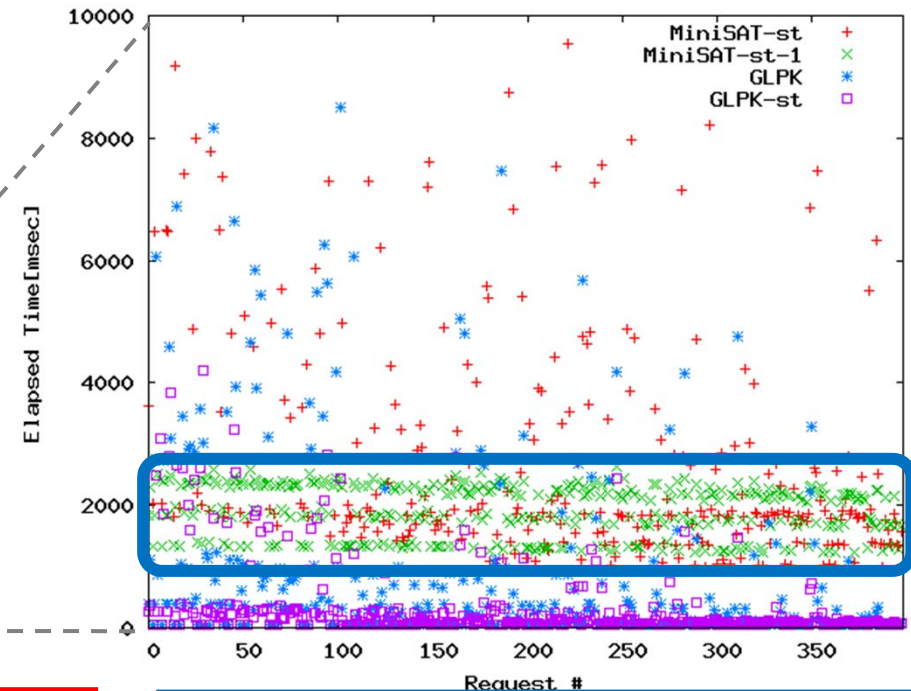
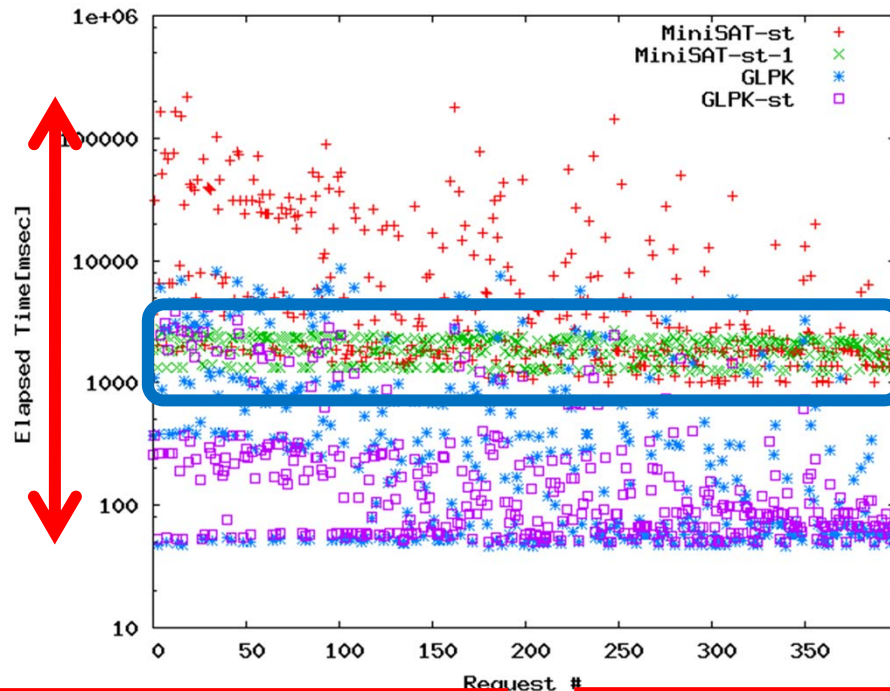
“-st” : 追加制約あり( $P_{max}=2$ ), “-1” : SATの実行回数1回  
解の質 :  $GLPK \geq GLPK-st = MiniSat-st \geq MiniSat-st-1$

# 個々の求解時間の比較

求解時間[sec](ログスケール)

求解時間(0~10[sec])

GLPK / GLPK-st / MiniSat-st / MiniSat-st-1



最適解を求める  
手法はばらつき大

資源数が減ると  
ばらつき小

MiniSat-st-1の求解時間は  
問題サイズに比例  
→多項式時間で求解可能

# 実用性の評価実験の考察

- LPの実求解時間は許容できる
  - LPソルバや計算機の性能が向上
  - LPの求解時間短縮の余地もあり
    - 制約条件を増やすと求解時間は短縮
    - 最適解を求めなければ, 多項式時間で求解可能
- 前提とするコアロケーション問題の性質
  - GRCが階層的に構成, 探索範囲の局所化が可能
  - 求解時間が計算機の台数ではなくサイト数でスケール
  - 遅延, 実行環境, データの所在など, 今後さらに制約条件が増える

→本研究のコアロケーションではLPが適用可能

# 発表概要

- 提案コアロケーション手法の概要
  - 最適化問題としてモデル化
  - 求解時間の短縮手法の提案
  - コアロケーション方針の反映
- 機能性の評価
  - シミュレーション環境で提案手法の有効性を実証
- 実用性の評価
  - アルゴリズム, ソルバの違いによる求解時間の比較
- 関連研究
- まとめと今後の課題

# 関連研究

適切な予約プランでない. 資源数減, 制約多の場合, 探索時間増

- バックトラック法によるスケジューリング[安藤, 合田, 2007]
  - 計算・ネットワーク確保手法を提案. ワークフローにも対応
  - インクリメンタルに予約手続きを行う場合, 多くの資源のブロックが発生. 確保に要する時間も長い
- NorduGridにおけるコアロケーション手法[Elmroth, Tordsson]
  - 予約時間帯をずらしながら, 計算機→ネットワークを探索
  - 制約の多い資源があると, 予約プランの作成に時間がかかる
- Co-reservationのためのグローバル最適化[Röblitz, 2008]
  - 最適化問題に基づく複数資源の同時予約手法を提案
  - ネットワークモデルが単純
  - ソルバにCPLEXを利用しており, 求解時間短縮の考察なし

いずれも, コアロケーション方針の反映は考慮されていない

# まとめ

- 分散実行環境のためのオンラインコアロケーション手法の提案
  - コアロケーション問題を最適化問題としてモデル化
  - コアロケーション方針が反映可能
  - 求解時間の短縮手法の提案
- 機能性の評価
  - 提案アルゴリズムによりコアロケーションが可能
  - 資源管理の観点でコアロケーション方針が反映可能
- 実用性の評価
  - 制約条件の追加により求解時間が短縮可能
  - 提案手法はオンライン処理でも実用レベル

# 今後の課題

- 実環境を考慮したスケラビリティに関する評価
  - 資源に関する現実的な制約条件の追加
  - 他のLPソルバ(近似解など)での求解時間の比較
- 重み付けに関する検討
  - ユーザのコアロケーション方針の反映の検証
    - 経済モデル
    - 多目的最適化への対応
- 実システムへの適用 → SC09でデモ

# 謝辞

- 神戸大学 田村直之先生, 丹生智也様
- 中央大学 藤澤克樹先生, 安井雄一郎様
  
- 本研究の一部は, 科研費(21700047)および情報通信研究機構(NICT)の委託研究「ダイナミックネットワーク技術の研究開発」の助成を受けたものである