先進的基盤ソフトウェア 1巻1号（通号1号）オンライン版 2007年11月15日発行
ISSN 1882-4196

先進的基盤ソフトウェア

Joint Symposium for Advanced System Software 2007
(JSASS2007)

2007年9月12日(水)・13日(木)

於 名古屋工業大学 (愛知県名古屋市)

JSASS実行委員会

巻頭言

立命館大学　大久保　英嗣

Joint Symposium for Advanced System Software は、2000 年に立命館大学びわ湖草津キャンパスで第 1 回を開催し、今年（2007 年）で 8 回目を迎えました。本シンポジウムは、各学会の研究会とは異なり、より緊密なそしてザックバランな議論を研究室レベルでの交流も含めて行いたいという趣旨で始まりました。丁度 2000 年問題で世界が揺れていた時期にあたり、21 世紀も始まる時期に、何か記念に残る会合を持ちたいという趣旨で、東京農工大学、和歌山大学、立命館大学の有志を中心に開催の運びとなりました。

ジョイントという意味は、そこにあります。すなわち、いくつかの大学の研究室の教員・学生・院生が集まり、まさしく合宿のような形で、様々な交流を行うという趣旨で名付けました。また、シンポジウムとしたのは、学生・院生諸君が、少しあらたまった会合で発表の機会を持ち、それぞれが学会発表や論文執筆につながるようにとの思いで、ワークショップではなくシンポジウムとしました。

本シンポジウムは、各大学持ち回りで幹事となり、企画を練っていただきました。立命館大学で 2 回、東京農工大学で 2 回、和歌山大学で 2 回、龍谷大学で 1 回、そして今年は名古屋工業大学で主催していただきました。例えば、特徴的なのは、和歌山大学で企画して頂いた 2003 年は、白浜の（株）エスアールアイのご好意で会場をお借りできました。海の見える場所で、ゆったりと議論できたことは今も新鮮な記憶として残っています。また、当日の夜の懇親会は、宿泊先の旅館の庭でバーベキュー大会が行われ、白浜町長のご挨拶もいただきました。夜遅くまで、旅館で議論したことが懐かしい思い出となっています。まさしく、全員が同じところに宿泊し議論したことは、参加した学生・院生諸君にもよい経験となったと思います。

さて、本シンポジウムは、システムソフトウェアを主題としてはいますが、研究分野を特に限定はしていません。2000 年からのプログラムも参照して頂いてもお分かりと思いますが、その年々のホットな話題や、一方で OS に関わる基礎的なかつ地道な研究発表もあります。予備的な研究、プロトタイピング、Work in Progress、フルペーパーに近い研究など、分野のみならず進捗において様々

なレベルの発表がありました。各発表者に対しては、鋭い突っ込みと、暖かい励ましを始め、研究の方向性の示唆まで、学会の研究会とは一味違ったやり取りがあります。学生・院生諸君は、研究の進め方、研究に望む姿勢、プレゼンの仕方など、貴重な意見がもらえたのではないかと思っています。一方、教員にとっては、思いもつかないアイデアが出てくることもあります。このように、参加者全員が前向きな議論をし、それなりの成果を挙げてきているからこそ、2000 年以来 8 年も続いてきたのだと思います。

さて、最後に、本シンポジウムの今後ですが、これまでどおりのやり方を維持しつつ、何か少し付加価値を付けていかなければならないと思っています。本会議録の刊行もその 1 つであります。さらに、例えば、IT 関連産業の動向の講演を入れるとか、また、アジアの IT 産業の動向を理解するためのツアーを企画することなども考えられます。また、10 周年企画をすることも考えられます。本シンポジウムが、今後、時代の変遷を経て、どのように変わっていくのか、変わらなければないのかなど、関連の皆様と少し議論してみたいと思っています。

最後に、本会議録の出版にあたって、東京農工大学の並木先生、立命館大学の毛利先生、名古屋工業大学の齋藤先生始め、シンポジウム参加校の先生方には大変お世話になりました。お礼を申し上げます。

2007 年 10 月 10 日記

Joint Symposium for Advanced System Software 2007 (JSASS2007)

2007 年 9 月 12 日(水)・13 日(木)

名古屋工業大学 (愛知県名古屋市)

プログラム

■ 9 月 12 日(水)

○ オープニング: 12:20～12:30

○ セッション 1: 12:30～14:10 セキュリティ，プライバシ保護

○ セッション 2: 14:20～15:35 無線ネットワーク，広域ネットワーク

○ セッション 3: 15:40～16:55 分散ストレージ，ファイルシステム

システムコールの実行順と実行位置に基づく侵入検知システムの実現 ～N-gram利用に関する検討～

槙本裕司[†] 齋藤彰一[†] 松尾啓志[†]

Realization of Intrusion Detection System Based on Pattern and Situation of System Call

Yuji Makimoto,[†] Shoichi Saito[†] and Hiroshi Matsuo [†]

1. はじめに

ゼロデイ攻撃を検出することができるシステムに異常検知 (Anormaly Detection) システムがある．異常検知はプログラムの正常な動作に基づく規則をあらかじめ作成しておき，その規則と異なる動作を異常と判断するシステムである．特定のプログラムにおける異常な動作や, その要因となるセキュリティホールをすべて把握することは不可能に近いが，正常な動作を定義することは可能である．

本稿では，異常検知を用いて効率的にプロセスの異常動作を検出する手法を提案する．

2. 提案手法

本稿では正常な動作および学習と実行ファイルの静的解析により作成した規則を，組み合わせて検知を行う検知システムを提案する．学習により作成した規則はプログラムの，実行されやすい動作を中心とした規則を作成できる．しかし，学習漏れの動作による false positive の発生が問題となってしまう．そこで，学習漏れによる未学習の動作は静的解析により作成した規則により，学習漏れの正常な動作か異常な動作かの判定をおこなう．これは，静的解析では，false positive が出ない規則を作成することができるからである．

なお，学習による規則の作成にはシステムコール番号と，その呼び出し元のユーザ関数の実行アドレスに基づいた N-gram 法を用いる．ユーザ関数の実行位置位置を付加することで，より厳密にプログラムの実行に基づいた規則を作成することできる．そのため，false negative の減少が期待できるが，その分 false positive が増加してしまう．

2.1 規則の概要と正常な動作の確認手順

提案手法では，次の 3 つの規則を作成する．また，それぞれの規則名を括弧内に書き表す．

(1) 正常な動作の学習によって作成した，システムコールの呼び出し順序を表す規則 (呼び出し順規則)

(2) 静的解析によって作成した，システムコールの呼び出し元であるユーザ関数の実行順を表す規則 (ユーザ関数実行順規則)

(3) 静的解析によって作成した，ライブラリ関数の呼び出し関係を表す規則 (スタックリスト確認規則)

これらの規則を用いて，呼び出されたシステムコールが正常な動作によるものか否かを次のアルゴリズムで確認する．

(1) 呼び出し順規則による確認を行う．正しい呼び出しであることが確認できたら (3) へ，そうでなければ (2) へ移行する．

(2) ユーザ関数実行順規則による確認を行う．正しい呼び出しであることが確認できたら (3) へ，そうでなければ異常と判断する．

(3) ライブラリ関数確認規則による確認を行う．正しい呼び出しであることが確認できたなら正常，そうでなければ異常と判断する．

学習済みの動作であれば (2) での確認を行なわないため，検知時間の短縮ができる．

† 名古屋工業大学
Nagoya Institute of Technology

表 1 N の評価

N	3	4	5	6	7	8	9	10
$H_n(X\|Y)$	0.19178	0.17985	0.08669	0.02092	0.00003	0.00003	0.00003	0.00003

表 2 監視による実行時間の増加率

	動作 1	動作 2
侵入検知なし	351.4(1.00)	118.0(1.00)
侵入検知あり: 未学習の動作パターンなし	550.5(1.57)	340.9(2.89)
侵入検知あり: 未学習の動作パターンあり	579.8(1.65)	366.9(3.11)

単位はミリ秒，括弧内は倍率

3. 実験と考察

3.1 実 験 概 要

2 章で述べた提案手法を Linux 上に実装し，実験を行なった．実験環境は Pentium4(3.40GHz)，メモリ 1GB である．侵入検知プログラムを ptrace を用いてユーザプロセスとして実装し，GNU の wc を監視対象とした．なお解析を簡単にするために，wc はライブラリを静的にリンクしてコンパイルしたものを使用した．

3.2 N-gram に用いる N の評価

N-gram では一連の情報の並びから，長さ N の並びのパターンを取り出す必要があり，このときの N は検知精度 (false negative の数) に影響する．そこで適切な N の大きさを決めるための評価を行なった．評価は，N-1 番目までのシステムコールの呼び出しから，N 番目のシステムコールが予測できるかという観点で，次の式を用いて行なった．

$$H_n(X|Y) = -\frac{1}{|D|}\sum_{(x,y)\in D}\log_2 P(x|y)$$

x は n 番目のシステムコール，y は n-1 個のシステムコールのパターン，(x,y) は長さ n のパターン，D はすべてのパターン，$|D|$ はすべてのパターンの数を表す．$P(x|y)$ は n-1 個のシステムコールが y であったときに n 番目のシステムコールが x である確率を表し，この確率が大きいほど呼ばれ得るシステムコールの数が限定され，偽装が困難になります．また，$P(x|y)$ が大きいほど $H_n(X|Y)$ は小さな値となります．

各 N の計算結果を表 1 に示すこの結果より N=7 として侵入検知プログラムのオーバヘッドの測定を行なった．

3.3 侵入検知プログラムのオーバヘッドの測定

wc を各条件の下でで次の 2 つの動作に要する時間を測定した．

- 動作 1: オプションなし
- 動作 2: -l オプション付

測定結果を表 2 に示す．表 2 の「未学習の動作パターンなし」は，規則作成に用いた動作と，検知時の動作が同じ場合である．逆に,「未学習の動作パターンあり」は，規則作成に用いた動作と，検知時の動作が異なる場合である．

動作 1 では，呼び出されたシステムコールの数は 936 個，未学習の動作パターンは 7 個，動作 2 では，呼び出されたシステムコールの数は 935 個，未学習の動作パターンは 6 個であった．どちらの動作の場合も検知に要する時間 200 ミリ秒程で，未学習の動作 1 つの確認に要する時間は 4 ミリ秒程度となった．また，学習済みの動作の確認はシステムコール 1 つあたり約 0.2 ミリ秒程度であるので，未学習の動作の確認に要するオーバヘッドは大きい．なお，呼び出し順規則によって正しいことがが確認されなかった未学習の動作は，すべてユーザ関数実行順規則により，正常な動作であるとが確認された．

4. まとめと今後の課題

正常な動作の学習により作成した規則と，実行ファイルの静的解析ににより作成した規則を用いて，プログラムの動作を確認しながら実行する検知システムを提案した．また，システムコール番号と実行位置情報の組の呼び出し列から N-gram 法を用いて特徴化する手法を提案した．

今後，提案システムでどの程度の検知精度が得られるか，脆弱性が知られているプログラムに適用して検証していく予定である．また，システムコールの呼び出し順を表す規則は，N-gram 法を用いて作成したが，静的解析により規則を作成した場合についても検討していきたい．

システムコールの実行順と実行位置に基づく侵入検知システムの実現 ～N-gram利用に関する検討～

槙本裕司　齋藤彰一　松尾啓志
名古屋工業大学

発表内容

- 背景・目的
- 侵入検知システム、検知方式
- 提案手法の概要
- 提案手法の確認アルゴリズム
- 実験・評価
- まとめ・今後の課題

背景

- 毎年多くの不正アクセスによる被害が発生している
  - 攻撃の巧妙化
  - ゼロデイ攻撃の増加
- プログラムのセキュリティホールを利用した攻撃
  - ➢バッファオーバーフローによる実行フローの操作
    - 情報の改ざん
    - ほかの計算機への不正アクセスの踏み台

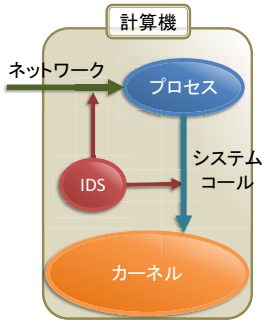
➢このような攻撃を検知するシステムが必要

目的

- ゼロデイ攻撃や未知の攻撃の検知
  - 侵入検知システムによる実現
  - 誤検知が少ない
  - 検知に要する時間が短い
- 検知対象の攻撃
  - バッファオーバーフローを用いた攻撃など
    - 関数の戻りアドレスを書き換え

侵入検知システム(IDS)

- ネットワークや計算機内のイベントを監視し、異常を見つける
- 検知方式
  - シグネチャマッチング方式
  - 異常検知方式



侵入検知システム (シグネチャマッチング方式)

- 明らかになっている攻撃の特徴(シグネチャ)との一致を調べる
  - 一致していたら攻撃を受けたと判断
- 特徴
  - 攻撃に対応したシグネチャを持っていれば高確率で検知できる
- 問題点
  - 攻撃が多様になればその分のシグネチャが必要
  - シグネチャのない攻撃を防ぐことができない

➢未知の攻撃やゼロデイ攻撃を防げない

侵入検知システム (異常検知方式)

- プログラムの正常な動作を定義
- 正常な動作からの逸脱を調べる
- 特徴
  - 未知の攻撃やゼロデイ攻撃も防げる
- 問題点
  - 誤検知が多い
  - 検知に時間がかかる

検知と速度を向上させる手法の提案

提案手法の概要

異常検知方式を使用
- 正常な動作の定義(**規則**)の作成
- プログラムの監視・評価

動作の評価
- システムコール呼び出し時に評価
  - システムコールの呼び出し順
  - 関数の実行順

規則の作成手法
- 正常な動作の学習
- 実行ファイルの静的解析

動作の評価

- 評価に用いる要素

システムコールの呼び出し順

- 正常な動作と異常な動作ではシステムコールの呼び出し順が異なる
- システムコールの履歴を保持

関数の実行順

- 異常な動作は実行ファイルに従わない関数呼び出しがある
- スタックに積まれた戻りアドレスから確認

規則の作成手法

- 正常な動作の学習
  - 実行される確率が高い動作の抽出
  - 規則のサイズが小さい

利点 ➢検知に要する時間が少ない

欠点 ➢誤検知が発生

- 実行ファイルの静的解析
  - 実行ファイルを解析して正常な動作を定義

利点 ➢False positiveが出ない規則が作れる

欠点 ➢規則による確認に時間がかかる

提案手法で作成する規則

呼び出し順規則

- システムコールの呼び出し順の確認
- 正常な動作の学習により作成

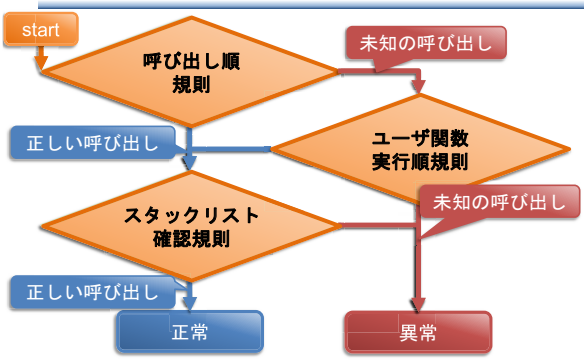
ユーザ関数実行順規則

- ユーザ関数の実行アドレスの遷移の確認
- 実行ファイルの静的解析により作成

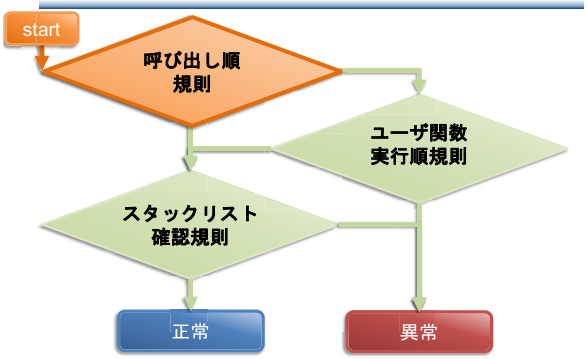
スタックリスト確認規則

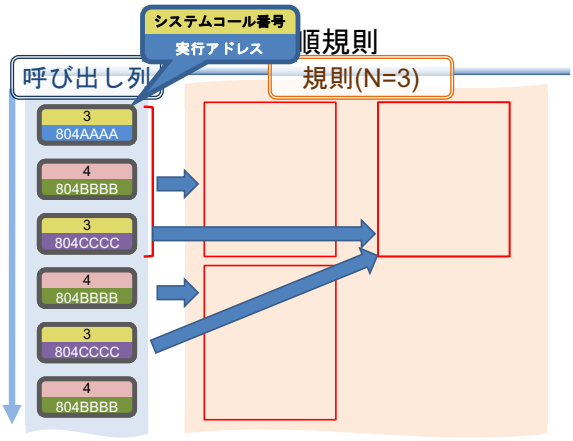
- ライブラリ関数の実行順の確認
- 実行ファイルの静的解析により作成

提案手法の確認アルゴリズム

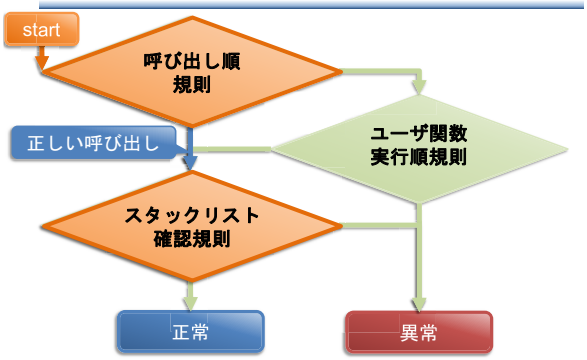


確認アルゴリズムのフローチャート



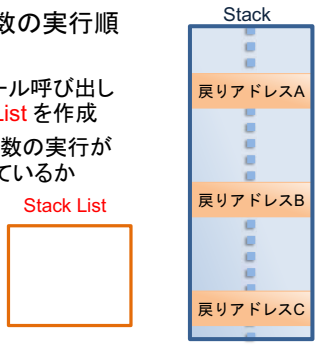


確認アルゴリズムのフローチャート

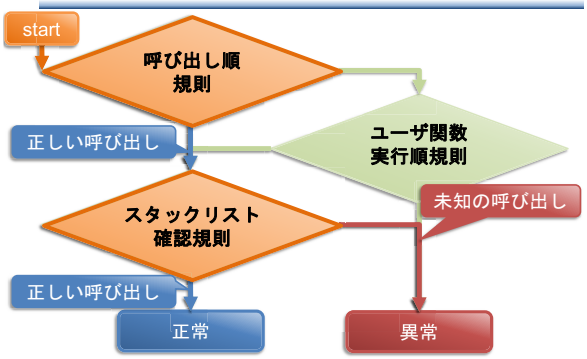


スタックリスト確認規則

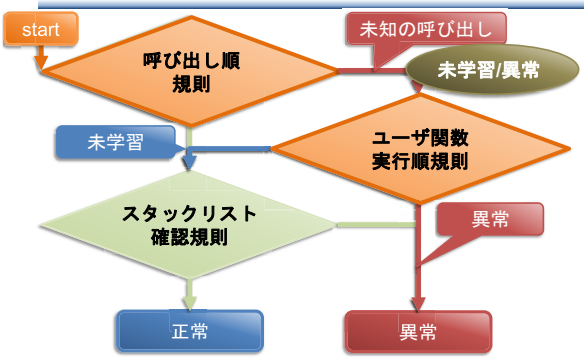
- ライブラリ関数の実行順の確認
  - システムコール呼び出し時のStack List を作成
  - ライブラリ関数の実行が規則に従っているか



確認アルゴリズムのフローチャート

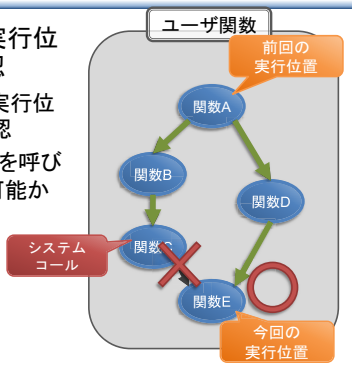


確認アルゴリズムのフローチャート

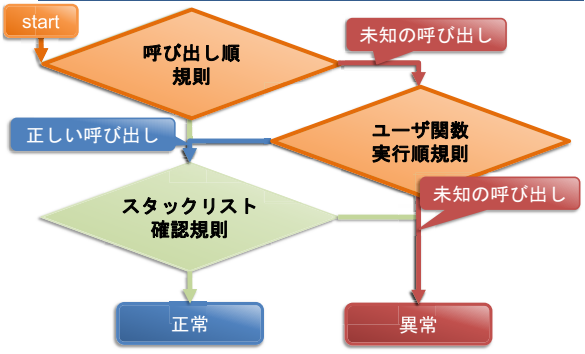


ユーザ関数実行順規則

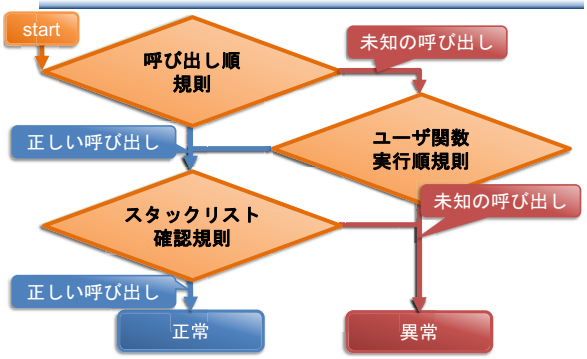
- ユーザ関数の実行位置の遷移を確認
  - ユーザ関数の実行位置の遷移の確認
  - システムコールを呼び出さずに遷移可能か確認



確認アルゴリズムのフローチャート



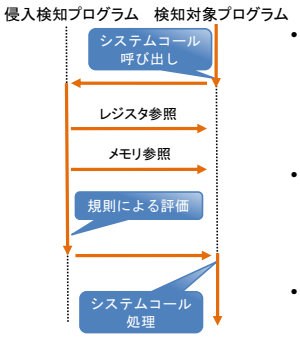
確認アルゴリズムのフローチャート



検知時間の測定

- 実験概要
  - 静的リンクしたwcを使用
  - ptraceを用いた監視
  - wcを検知するのに最適なNの評価
  - 検知に要する時間の測定
    - オプションなしの動作と-lオプション付の動作
- 実験環境
  - Pentium4(3.40GHz)
  - Linuxカーネル2.6.17.8

ptraceを用いた監視手法



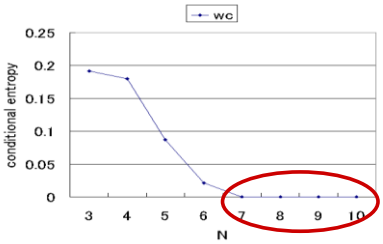
- 別のプロセスで監視する
  - 検知対象プログラムはシステムコールの処理の直前直後で停止
  - レジスタ、メモリの参照
- 欠点
  - メモリリソースを多く必要
  - レジスタ、メモリアクセスに時間がかかる
  - カーネル内実装より遅い
- 利点
  - 実装が容易

N-gramのNの評価

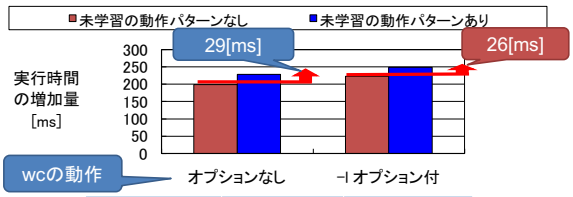
- $H_n(X|Y) = -1/|D| \sum \log_2 P(x|y)$

x: (n)th system call
y: sequence of n-1

|D| : total number of sequence



実験結果



	オプションなし	オプション付
侵入検知なしでの実行時間[ms]	351	118
システムコールの数	936	935
未学習の動作パターン数	7	6

まとめ

- 正常な動作の学習と、実行ファイルの静的解析を用いて規則を作成
  - 呼び出し順規則とスタックリスト確認規則による高速に動作の確認
  - ユーザ関数実行順規則による未学習パターンの判定
- 侵入検知システムでプログラムを監視した時の実行時間の測定
  - wcでの実行時間はシステムコール約935個で約0.2秒の増加
  - 未学習のパターンに対しては1パターンあたり約4ミリ秒の増加
  - ➢未学習のパターンが多いと実行速度の低下が予想される

今後の課題

- カーネル内での実装
- セキュリティホールの知られているプログラムへの適用
- 提案手法での検出精度の評価
- 関数の引数を変える攻撃の検知

コンパイラと OS の連携による 強制アクセス制御向けプロセス監視手法

表 雄仁†

†立命館大学大学院理工学研究科

近年，ネットワーク技術の発展から起きた弊害として不正アクセスが急増してきている．特に，バッファーオーバフロー攻撃などのソフトウェアの脆弱性をついた攻撃は年々増加しており，大きな問題になっている．Linux ではそうした攻撃で管理者権限を奪われてしまうとシステム全体を乗っ取られたことになる．この問題を解決する手段として，強制アクセス制御機能を付加した Secure OS が提供されている．強制アクセス制御とは，あらかじめ定義されたルール（ポリシー）の許容範囲内でのみプログラム実行を可能とする機構である．従って，管理者プロセスに対してもユーザプロセス同様のアクセス制限を付加することが可能となっている．この手法を組み込むことにより，管理者権限を奪われてしまったとしても被害を最小限に抑えることができる．Linux において，ファイルやディレクトリ，ソケットなどリソースへアクセスするためには全てシステムコールが使用される．従って，プロセスが使用するシステムコールを OS 側で管理，制御することで，攻撃されてもシステムへの被害を最小限に軽減できる．この考えに基づき，Secure OS の多くはシステムコール発行時に，アクセス制御を行うことで実行プロセスの監視を行っている．

しかし，既存の Secure OS の多くはアクセス制御を行うためのポリシーを管理者が記述しなければならならず，かつ，その記述が難しいという問題点がある．また，プログラムを事前実行させてポリシーを自動生成する Secure OS も存在するが，実行時に必ずしもプログラムに書かれているすべての処理が行われるとは限らないので，学習漏れが起こりポリシーの記述が正しく行われない可能性がある．従って，管理者がポリシーを適切に記述，もしくは学習させる必要がある．

こうした現状を踏まえ，我々は，コンパイラの解析からポリシーを生成し，生成されたポリシーを基に実行時に動的解析を行うセキュアシステム提案する．コンパイラと OS が連携することでユーザに負担のない高信頼性のシステムを構築することを目標とする．ここで，高信頼性のシステムとは，プロセスの安全性を保証したシステムとする．プロセスの安全性は，プログラムや実行中のプロセスに対する悪意を持ったコードの書き換えなどにより，ユーザの意図に反した方法でリソースにアクセスできないこととする．本システムでは，コンパイラがソースコードを解析することでポリシーを生成するので，ユーザに負担がなく，さらに細かな制御情報をポリシーとして生成できる利点がある．OS がポリシーを基に動的解析を行うことで，コード書き換えやプロセスの乗っ取りによる不正なプログラムの実行，不正なファイルアクセスも防止することができる．OS によるアクセス制御を行うことで，システム管理者にもユーザと同様のアクセス制御を施すことができる．その意味で一般の Secure OS と同等の強制アクセス制御が実現できると考える．

本研究では，上記の考えに基づき，コンパイラの解析情報を基にした強制アクセス制御を行うためのプロセス監視手法についての述べる．本手法は，システムコールの発行を契機として，発行されたシステムコールやその実引数を解析し，コンパイラにより生成されたポリシーを基に検査を行う．さらに，動的解析時にシステムコールがアクセスするファイルパスの限定を行うことにより，より細かいアクセス制御の実現が可能となっている．提案手法の有効性を示すためにユーザプロセスを監視するプロトタイプシステムの実装を行い，提案システムの検討を行う．

A Process Monitoring for Mandatory Access Control by Collaboration of Compiler and Operating System

Yuji Omote†

†Graduate School of Science and Engineering Ritsumeikan University

コンパイラとOSの連携による強制アクセス制御向けプロセス監視手法

名前：表　雄仁
立命館大学大学院　理工学研究科
國枝・桑原研究室
rs011024@se.ritsumei.ac.jp

背景

- 背景
  - 近年，ネットワーク技術の発展の弊害として不正アクセスの急増
    - プログラムの改竄，のっとい，踏み台など
  - バッファオーバーフローなどの脆弱性を利用した攻撃による被害の増大

  - Secure OSが注目
    - 強制アクセス制御
      - セキュリティポリシーによるアクセス制御を管理者にも強制
    - 最小特権
      - ユーザやプロセスが必要最低限の権限のみを付与

既存のSecureOS

- SELinux
  - プロセスがどのリソースにアクセスできるかを詳細に記述可能
  - システムコールの制限
  - 設定が難しく煩雑になりやすい
- TOMOYO Linux
  - プログラムを実行し，そのリソースへのアクセスを学習させポリシーを作成
  - プログラム実行，ファイルシステムへのアクセス権限の自発的放棄
  - 管理者の負担の軽減
  - 学習からもれる場合もある

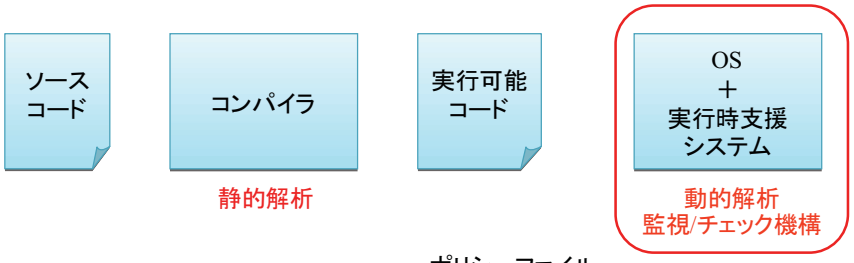
管理者が適切なセキュリティポリシーを記述を要する
・制限が厳しいと，プログラムが動かなくなる可能性がある
・制限が緩いと，セキュリティの脆弱性となる可能性がある

提案システムの目的と特徴

- プロセスの乗っ取りやプログラムの改竄による不正アクセスを防止
  - プログラムの正規の挙動を保障
    - コンパイラは正確な処理の流れを把握できる
    - プロセスに対して細かな制限をすることが可能
    - ユーザの負担を削減することが可能

- コンパイラとOSの連携により，ユーザに負担のない高信頼システムの構築
  - ただし，ソースプログラムは正しいものとする

システム概要



● 静的解析情報埋め込み
- システムコール発行情報
- ファイルアクセス情報
- トレース情報

ポリシーファイル
として埋め込む

- 不正なデータアクセス/
  ファイルアクセスを排除
- 不正なプロセスの実行禁止

動的解析

- システムコールに着目しプロセスを監視，制御する
  - 各命令の実行はOSは関知しない
    - 例外：リソースへのアクセス，ページフォルト，割り込み
  - リソースへのアクセスを制限することにより，被害を最小限することが可能
    - コンパイラにより，発行するシステムコールなどの制限が可能
    - 実行時の解析を行いやすい
- プロセス監視手法
  - システムコールの検査
  - システムコールの引数の動的解析および検査

システムコールの検査

- ポリシーファイルに記述されているシステムコールかを検査
- システムコールが発行される順序が一意に決定する場合は順番を考慮する

システムコールの引数の動的解析および検査

- システムコールの引数

```
execve(char *filename , char *argv[], char *envp[]);
```

  - 静的解析では，定数，値域，未定義と評価される
    - 値域：取り得る値の範囲もしくは集合
    - 未定義：静的解析では値の絞り込みができなかった場合
- システムコールの引数の動的解析
  - 定数，値域の場合はパターンマッチングによる照合
  - 未定義は全て許可

システムコールの順序を気にしていないので，複数回同じシステムコールが呼ばれると，不要なアクセスパスも許可してしまう

システムコールの引数の動的解析および検査

- 複数のシステムコールの同一変数使用による値の固定化

```
void file_copy( *from_name, *to_name)
{                                          システムコール間で同じ変数が使われる
      int form_fd, to_fd;                  かつ，範囲内での書き換えがない
      char buff[BUFSIZE];
      int rcount, wcount;

      form_fd = open(from_name, O_RDONLY);
      to_fd   = open(to_fd,  O_WRONLY | O_CREATE | O_TRUNC, 0666);
      while ((rcount = read(form_fd, buff, BUFSIZE) > 0 ) {
            wcount = write(to_fd, buff, rcount);
      }
      close(form_fd);
      close(to_fd);
}
```

システムコールの引数の動的解析および検査

- 複数のシステムコールの同一変数使用による値の固定化

```
void file_copy( *from_name, *to_name)
{                                          システムコール間で同じ変数が使われる
      int form_fd, to_fd;                  かつ，範囲内での書き換えがない
      char buff[BUFSIZE];
      int rcount, wcount;

      4     = open(from_name, O_RDONLY);
      to_fd   = open(to_fd,  O_WRONLY | O_CREATE | O_TRUNC, 0666);
      while ((rcount = read( 4    , buff, BUFSIZE) > 0 ) {
            wcount = write(to_fd, buff, rcount);
      }
      close(    4    );
      close(to_fd);
}
```

システムコールの引数の動的解析および検査

- ディレクトリの遡りを一部禁止
  - 許可されていないパスを通り，他のファイルパスへのアクセスを禁止
  - 例：

```
execve(char *filename , char *argv[], char *envp[]);
    許可されているfilename -> /bin/* | /home/user/bin/*
```

- filename = “/home/user/bin/../../../bin/ls”

“/” ⟶ 許可されていないパス

コンパイラが生成するポリシーファイル(許可リスト)

```
syslist {
    main {
        execve  “/bin/ls”       argv    NULL
    }
    file_copy {
        open    from_name       0       0
        open    to_name         577     0666
        read    from_fd         buff    1024
        write   to_fd           buff    rcount
        close   from_fd
        close   to_fd
    }
}
```

システムコールを発行するユーザ関数

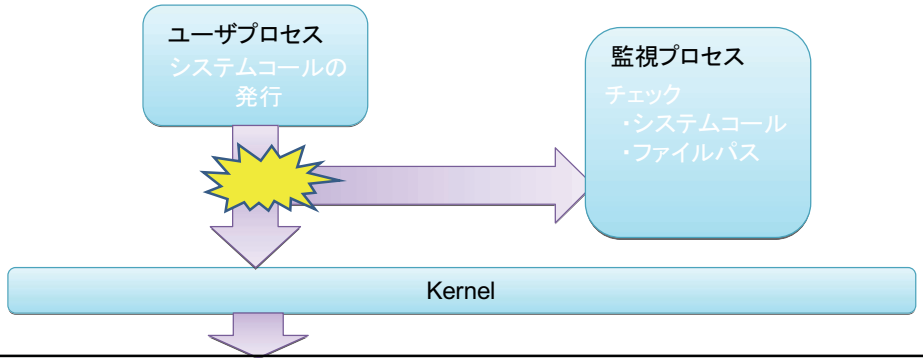
発行されるシステムコール

システムコールの引数リスト

- プログラムが呼ぶシステムコールの限定
  - 不要なシステムコール発行権限を放棄
- ファイルパスの限定
  - プログラムがアクセスできるファイルアクセスパス，実行パスの限定
- 管理者にもコンパイラが生成した制御情報
  - 強制アクセス制御の実現

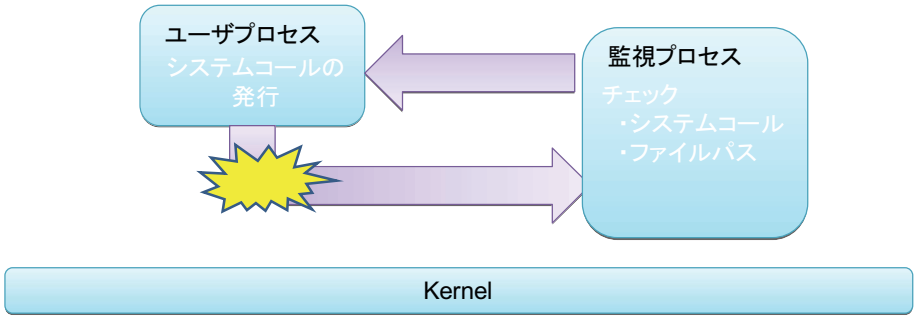
プロトタイプ

- ユーザプロセスを監視する実行時システムとして構築
  - ユーザプロセスを監視する監視プロセス
  - システムコールを起点として，ユーザプロセスの解析



プロトタイプ

- ユーザプロセスを監視する実行時システムとして構築
  - ユーザプロセスを監視する監視プロセス
  - システムコールを起点として，ユーザプロセスの解析



検討項目

- シンボリックリンク使用などによるファイルパスの変更
  - 現状は，ファイルパスが正しい状態で呼ばれているかの検査しか行っていない
- 引数値が未定義の場合のセキュリティの確保のための検討
  - 未定義は，すべての値を取り得る．つまり，そのシステムコールに制限はかけられていない状態
- アドレス改竄に対する対処
  - 現状は，許可されているシステムコール，ファイルパスの範囲内の攻撃には弱い

今後の課題

- 精度評価
  - False Positive:
    - 正しいプログラムが異常動作と判断される場合
  - False Negative:
    - プロセスが乗っ取られた場合でも，検出できない場合
- 検討項目の改善
- 本手法のOSへの実装

参考文献

- SELinux
  - SELinux, http://www.nsa.gov/selinux/
- 原田季栄, 保理江高志, 田中一男:
  “TOMOYO Linux –タスク構造体の拡張によるセキュリティの強化 Linux-”
  Linux Conference 2004

コンパイラと OS の連携による強制アクセス制御向け静的解析

森山　壱貴[†]

[†]立命館大学大学院　理工学研究科

近年，報告されるソフトウェアの脆弱性を狙った攻撃で，比較的高い割合を占めるものはバッファオーバーフローによる攻撃であり，高い権限で実行されているプロセスが攻撃されると，悪意のあるコードがその権限で実行される危険性がある．そのため，重大なセキュリティ侵害を引き起こす可能性が高い．こうした脅威からコンピュータの被害を最小限に抑えるため，OS レベルでの強固なセキュリティ対策が求められている[1]．

現状の問題の一つとして，ファイルへのアクセス権をユーザが任意で設定可能なことが挙げられる．このため，多くのセキュア OS は強制アクセス制御機構を備えている．強制アクセス制御とは，各リソースに対し正常なアクセスを定義したポリシーを予め用意し，それに基づいたアクセス制御を全プロセスに強制する機構である．これにより，バッファオーバーフローなどの攻撃によってプログラムの制御を不正に奪取された場合でも，そのプログラムが必要としないファイルへのアクセスを禁止できる．すなわち，脆弱性の影響がシステム全体へ拡散することを抑制して，被害を限定化・局所化でき，リスクを最小に抑えることができる．しかし，アクセス制御を定義するポリシーの記述が複雑になるという欠点を抱えており，高い安全性を得るためには，煩雑なアクセス制御設定をアプリケーションごとに行う必要があるため活用されているとは言えない[2]．この煩雑さを軽減する対策として，対象プロセスを実行し，アクセスされたファイルや他のプロセスとの通信，発行されたシステムコールなどのログを取ることによってプログラムの挙動を解析し，それを基にポリシーを自動生成(学習)する機能が提案されている．しかし，完全なポリシーを生成できるわけではなく，その解析結果を雛形としても，必要十分なポリシーの作成は，なお困難な作業である．

本研究では，コンパイラと OS の連携によって強制アクセス制御を行う手法を提案する．コンパイラの静的解析技術により，システムコールに着目してソースコードから正常なアクセスを表現するポリシーを自動生成し，OS はポリシーに基づいて実行時の動的解析と監視機構により強制アクセス制御を実現する[3]．双方の組み合わせによりポリシーの記述作業を削減し，ヒューマンエラーを減少させることによって信頼性の高いシステムを構築することを目指す．プログラムの挙動をその原型であるソースコードからコンパイラの静的解析技術を用いて解析することにより，実行コードから挙動を解析した場合と比較して，詳細なアクセス制御情報を生成することが可能となる．また，システムコールに着目することで，例え侵入されても被害の影響範囲を最小限にし，プログラムの脆弱性に対するリスクを大幅に低減することができる．

本発表では，生成するポリシーを検討し，コンパイラの静的解析技術を用いたポリシーの生成手法について述べる．

Static Analysis for Mandatory Access Control by Collaboration of Compiler and OS

Ikki Moriyama[†]

[†]Graduate School of Science and Engineering, Ritsumeikan University

参考文献

[1] 総務省：セキュア OS に関する調査研究会報告書，http://www.soumu.go.jp/s-news/2004/040428_1.html.

[2] 情報処理振興事業協会セキュリティセンター：オープンソースソフトウェアのセキュリティ確保に関する調査，http://www.ipa.go.jp/security/fy14/reports/oss_security//part1.pdf.

[3] 表雄仁，森山壱貴，桑原寛明，毛利公一，齋藤彰一，上原哲太郎，國枝義敏：コンパイラと OS の連携による強制アクセス制御向けプロセス監視手法，Computer Security Symposium 2007 (CSS2007)，2007(投稿中)．

Summer Joint Symposium 2007 for Advanced System Software

コンパイラとOSの連携による 強制アクセス制御向け静的解析

立命館大学大学院理工学研究科
高性能計算機ソフトウェアシステム研究室
（國枝・桑原 研究室）
森山 壱貴

発表内容

- 背景と目的
- 関連研究
- 概要と提案手法
- 設計と実装
- 課題と問題点
- まとめ

研究背景 1

- コンピュータセキュリティに関する事件が増加
  - 情報漏洩やクラッキング
  - ソフトウェアの脆弱性を狙った攻撃
- バッファオーバーフロー攻撃への対策
  - ライブラリ：安全な関数に置換
  - コンパイラ：検知コードの挿入
  - OS：スタック領域でのコード実行を禁止
  - 完全に防御可能ではない

研究背景 2

- SELinux などのセキュアOSが注目
  - セキュリティポリシーの定義が必要
  - 適切なポリシーの記述が困難
- セキュアOSのポリシー生成機能
  - プログラムの実行からログを採取
  - ログを基にポリシーに変換
    - 必要十分なポリシーとは言えない

研究目的

- セキュリティの強化
  - 煩雑な記述を必要としない安全なシステム
  - プログラムの意図しない動作を抑止
- 情報漏洩やデータの改竄を防御
  - リスクを最小に抑える
  - バッファオーバーフローなどによるプロセスの乗っ取りから，情報資産を保護

関連研究

- ソースコードの静的解析技術
  - パターンマッチング
  - 構文解析技術
- コンパイラによるセキュアコード生成技術
  - 検知コードの挿入
  - メモリ領域の検査
- システムコールによる異常検知
  - N-gramに基づく手法
  - 引数のモデル化
  - スタック情報を利用した手法

概要

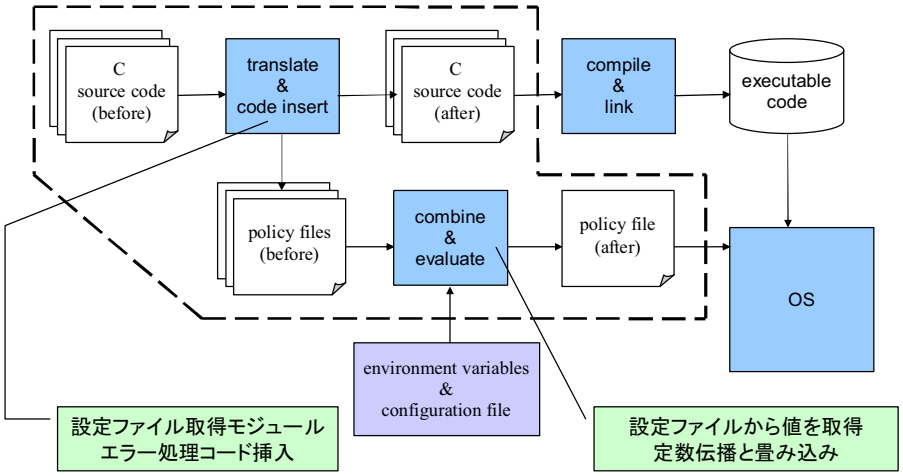
- システムコールに着目した強制アクセス制御
  - 攻撃に利用されると非常に危険
    - ファイルの読み書き
    - ネットワークへのアクセス
  - 脆弱性のあるプログラムが実行された際のリスクを大幅に低減
- ソースコードから強制アクセス制御情報を生成
  - ソースコードを基に正常な動作を定義
  - システムコールの発行・アクセス範囲

提案手法

- コンパイラによる静的解析技術
  - システムコールの種類と発行位置
  - システムコールの引数に取られる値の範囲
- OSによる動的解析と監視機構
  - システムコールに着目した動的解析
  - 不正なシステムコール実行の禁止

両者の組み合わせにより，信頼性の高いシステムを構築

システムの構成



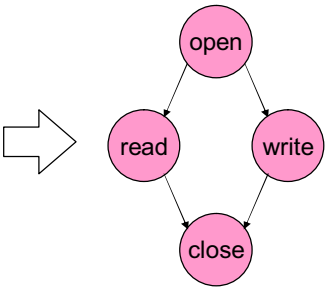
静的解析技術を用いたポリシー生成

- システムコールの種類と発行位置の解析
  - 発行シーケンスと制御フローとの関連付け
  - 種類・発行位置の特定
  - ライブラリコールとの対応付け
- システムコールの引数に対するデータフロー解析
  - 引数の範囲抽出・正規化
  - 引数の定義・使用に対する追跡情報

システムコールの種類と発行位置の解析1

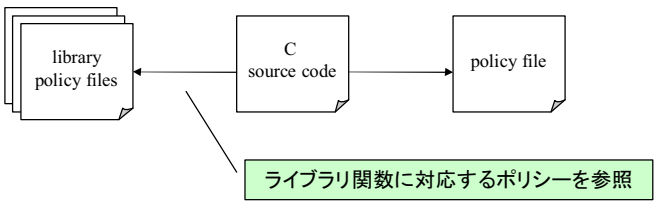
- 関数の呼び出し順序をグラフ化
- 発行シーケンスと制御フローとの関連付け

```
fd = open(…);
if(…)
  read(…);
else
  write(…);
close(…);
```



システムコールの種類と発行位置の解析2

- ライブラリコールとの対応付け
  - ライブラリ関数に含まれるシステムコール
  - 各ライブラリのポリシーを生成
  - ソースコード中のライブラリ関数を対応するポリシーに変換して生成



システムコール許可リスト

プログラム全体で実行可能なシステムコール

file_copy 関数で実行可能なシステムコール

```
syslist {
  main {
    execve  "/bin/ls"  argv  NULL
  }
  file_copy {
    open   from_name  0
    open   to_name    577   0666
    read   from_fd    buff  1024
    write  to_fd      buff  rcount
    close  from_fd
    close  to_fd
  }
}
```

引数の範囲抽出・正規化

- 定数伝播と畳み込み
  - 静的に解析可能なアクセスパス・範囲の抽出
- 引数の取り得る範囲を正規化

```
strcat(path, "/etc/");
if(…) {
  strcat(path, argv[1]);
} else {
  strcat(path, "passwd");
}
```

path → "/etc/"(* | "passwd")

引数の追跡情報

- 引数に使用される変数の定義・使用について追跡
  - 各システムコールと各引数の関連を解析
  - 一連の処理を行うシステムコール群を識別

```
from_fd = open(from_name,O_RDONLY);
to_fd = open(to_name, O_WRONLY|O_CREAT|O_TRUNC, 0666);
while((rcount = read(from_fd, buff, BUFFSIZE)) > 0)
  if((wcount = write(to_fd, buff, rcount)) != rcount) {
     perror(to_name); exit(1);
}
close(from_fd);
close(to_fd);
```

現状の課題

- 現状のプロトタイプ
  - コンパイラのフロントエンドと基礎レベル最適化
    - ANSI-C準拠のフロントエンド
    - 定数伝播と畳み込み
  - システムコール許可リスト
    - ファイル・関数単位でのシステムコールの種類と引数リスト生成
    - システムコールの種類は実行やファイル・ネットワークアクセスに関連したシステムコールを名前を基に識別
- システムコール発行シーケンスと制御フローとの関連付けが課題

現状の問題点

- 現状の問題点
  - ポリシーの表現方法
    - システムコールの許可リスト・発行位置
    - 引数の正規化・追跡情報
  - ソースコードに記述されたシステムコールを対象
    - ライブラリ関数が呼び出すシステムコールのポリシーも必要
  - ライブラリ関数単位でのポリシー生成
    - 文法拡張やマクロが多く，解析が困難
    - 主要なライブラリ関数のポリシーを手動で生成

今後の課題

- ポリシーの再構築
  - 環境変数や各アプリケーションの設定ファイルを基に引数の取り得る範囲を絞り込む
- データフロー解析の高機能化
  - 引数（文字列）に対するデータフロー解析の精度向上
- OS側のオーバヘッドを削減
  - エラー処理に対するシステムコールの表現

まとめ

- 背景と目的
  - バッファオーバーフロー攻撃
  - セキュアOSのポリシー記述が困難
- 関連研究
  - 静的解析技術・コンパイラによるセキュアコード生成
  - システムコールによる異常検知
- 概要と提案手法
  - コンパイラとOSの連携によるシステムコールに着目した強制アクセス制御
  - ソースコードから強制アクセス制御情報を生成
- 設計と実装
  - システムコールの種類と発行位置の解析
  - システムコールの引数に対するデータフロー解析
- 課題と問題点
  - ポリシーの表現方法
  - ソースコードに記述されたシステムコールを対象
  - ライブラリ関数単位でのポリシー生成

参考文献

1. D. Wagner, D. Dean：Intrusion Detection via Static Analysis：In Proceedings of the 2001 IEEE Symposium on Security and Privacy. pp.156-168, (2001/05).

2. 阿部洋丈, 大山恵弘, 岡瑞起, 加藤和彦：静的解析に基づく侵入検知システムの最適化, 情報処理学会論誌：コンピューティングシステム, Vol.45 No.SIG3(ACS 5), (2004/03).

3. 総務省：セキュアOSに関する調査研究会報告書, http://www.soumu.go.jp/s-news/2004/040428_1.html

4. 情報処理振興事業協会セキュリティセンター：オープンソースソフトウェアのセキュリティ確保に関する調査, http://www.ipa.go.jp/security/fy14/reports/oss_security/part1.pdf

5. バッファオーバフロー対策技術に関する報告書. http://itslab.csce.kyushu-u.ac.jp/ssr/tatara report.html

Thin Client における USB メモリを介した情報漏洩の防止手法

岩永 真幸 †
† 立命館大学大学院理工学研究科

1 はじめに

近年，情報の漏洩事件が頻繁に発生している．情報の漏洩事件の原因は，端末の盗難，内部の人間による誤操作や管理ミス，正当なアクセス権限を持つものによる情報の持ち出し，外部犯による情報の不正な持ち出し，ウイルスやワームによる漏洩などである．現在発生している情報漏洩事件の多くは，内部の人間による誤操作，端末の盗難，正当なアクセス権限を持つ者による情報の持ち出しである．従来のセキュリティ技術は，外部からの攻撃からデータを保護するものであるため，これらの原因による漏洩を防止することができない．

このような背景から，研究室では，Privacy-Aware OS *Salvia*[1] を開発している．*Salvia* は，内部の人間による誤操作や不正行為による情報漏洩を防止することを目的としている．

また，セキュリティの観点から，近年 Thin Client システムが注目されている．Thin Client システムは、アプリケーションやファイルをサーバ側で保持しているシステムである．そのため，クライアント側にデータが存在せず，端末の盗難，クライアント端末での不正行為による情報の漏洩を防止することが可能である．

そこで，Thin Client システムの上で *Salvia* を構築することで，より強固なセキュリティを実現しようとしている．しかし，Thin Client システムの上で *Salvia* を利用することによって，今まで想定していた *Salvia* の利用状況が変化する．Thin Client システムで *Salvia* を適応することで，ユーザの位置とデータの位置が変化する．Thin Client システムでは，処理を行う端末の位置とユーザが使用する端末の位置が異なる．*Salvia* ではコンテキストに基づいてデータ保護を行っているが，位置コンテキストがユーザの位置と異なってしまう．データの位置変化は，クライアント端末で外部記憶装置を利用した場合に，外部記憶装置のデータがネットワークの向こう側に存在するために発生する．通常の外部記憶装置のデータとクライアント側で利用する外部記憶装置のデータではデータの位置が変化するため，情報漏洩の危険性が変化する．

Thin Client システムにおける外部記憶装置は，*Salvia* としても解決しなければいけない課題であるが，Thin Client システムが持つセキュリティの問題でもある．Thin Client システムのセキュリティの利点は，クライアント側にデータが存在しないことで，データが漏洩しないことである．しかし，クライアント側で外部記憶装置を利用すると，外部記憶装置を介して情報が漏洩する危険性がある．そこで Thin Client のクライアント側で最も利用される外部記憶装置である USB メモリからの情報漏洩を防止する手法を提案する．

本稿では，2 章で外部記憶装置を介した情報漏洩について述べ，3 章で外部記憶装置からの情報漏洩を防ぐために企業がどのような対策を行っているかについて述べ，4 章で USB メモリからの情報の漏洩を防止するための手法について述べ，5 章でまとめとする．

2 外部記憶装置を介した情報漏洩

Thin Client システムは，近年セキュリティの観点から注目され始めている．Thin Client システムは，データやアプリケーションがサーバ側に存在し，処理をサーバ側で行う．そのため，クライアント側にデータが存在しないために不正な処理を行ってデータを持ち出すことを防ぐ．また，サーバ側ですべて管理しているため，セキュリティ対策を統一でき，サーバ側でセキュリティ対策を行うことで情報漏洩を防ぐことが可能である．

しかし，Thin Client 端末の中には外部記憶装置を利用することが可能なものも存在し，その外部記憶装置によってサーバからデータを持ち出すことが可能である．外部記憶装置を利用可能な Thin Client システムは，外部記憶装置が普及し，外部記憶装置の利用が不可能な状態では業務などに支障をきたすために，必要とされている．よって，Thin Client システムの外部記憶装置からの情報漏洩を防止することは重要であると考えている．

3 外部記憶装置からの情報漏洩を防ぐための企業の対策

Thin Client 端末で外部記憶装置が利用することが可能にすると，情報漏洩の可能性が発生する．そのため，Thin Client 端末を販売している企業は，さまざまな技術を用いて対策を行っている．以下に実例を示す．

3.1 富士通の Portshutter

富士通の Portshutter[2] は，USB 接続の機器や PC カードの制御が可能である．この機能は，利用可能なデバイスを個々に設定することが可能である．これにより，会社で支給した USB メモリや PC カードのみ利用可能といったことが可能となるため，会社外の人間が外部記憶装置を利用して情報を持ち出すことを防止することが可能である．

3.2 デバイスの制御

この仕組みは，ユーザや端末によってデバイスの利用の可否を設定することが可能である．企業などでは，重要なデータにアクセスする必要があるユーザが決まっている場合が多い．そのため，重要なデータにアクセスする必要のないユーザのデバイス利用を禁止したり，重要なデータにアクセスする必要のないユーザのみが利用する端末のデバイスの利用を禁止することで，情報の漏洩を防止することが可能である．

3.3 認証の強化

ユーザ ID やパスワードが洩れると，悪意のある人間に外部記憶装置を使ってデータを持ち出されてしまう可能性がある．そこで，指紋認証や認証デバイスなどで認証を強化することでなりすましを防ぎ，情報の漏洩を防ぐことが可能である．

4 USB メモリを介した情報の漏洩を防止する手法の提案

3 章で述べた方法は，外部の人間による漏洩を防ぐことは可能であるが，内部の人間による情報を漏洩させることは防止できない．また，デバイスの制御のようにユーザや端末によってデバイスの利用を禁止すると，利用できないユーザが業務に支障をきたす可能性がある．そこで，外部記憶装置の中でも Thin Client システムで最も利用される USB メモリに着目し，すべてのユーザが USB メモリを利用可能であり，重要なデータが USB メモリから持ち出されることを防ぐ手法を提案する．

Thin Client システムでは，USB メモリがクライアント側に差し込まれると USB メモリ内のファイルの情報を送信する．そして USB メモリ内のファイルへアクセスするとアクセスされたファイルのデータがサーバへ送信される．USB メモリへファイルを書き込むと，書き込まれたファイルの情報と USB メモリ内のファイルの情報を合わせてクライアントへ送信する．その後，書き込まれたファイルのデータをクライアントへ送信する．このことから，サーバ内部のデータの持ちだしを防止するために書き込みが行われた時に発生する 2 つのデータの送信を失敗させることで，データの漏洩を防止する．

クライアント側の USB メモリの中にデータを書き込んだ時の処理の流れを図 1 に示す．クライアント側で USB メモリが差し込まれている状態で，I/O デバイスを使って USB メモリにデータの書き込みを行うと，クライアント側は割込みを受けると，その種類とどのような操作が行われたのかを解析し，サーバに送信するためのデータを作成する．そして，I/O デバイスの操作情報としてサーバ側に送信する．サーバ側は受信した操作情報に対応する処理を行う．操作は USB メモリへのファイル書き込みであるため，サーバは上記で説明したファイルの情報とファイルのデータを送信する．

この処理の流れの中で重要なファイルを保護するため

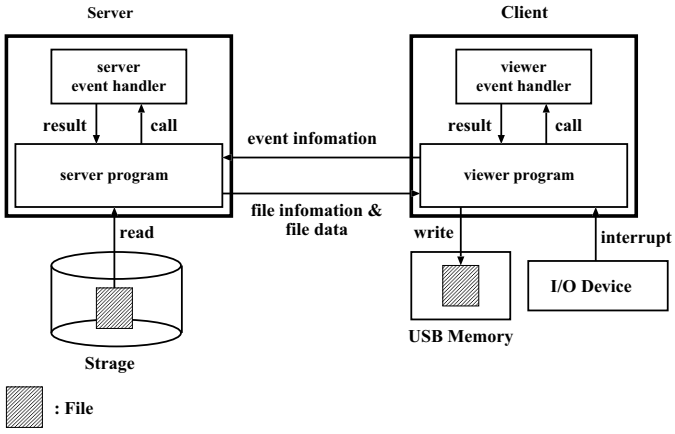


図 1　USB メモリへの書き込み時の処理

に，まず重要なファイルには，Thin Client 端末の USB メモリへの書き込みを禁止するフラグを付加する．そして書き込みが発生した時には，書き込まれるファイルをチェックし，書き込みを禁止するフラグが付加されていれば，2 つのデータの送信を失敗させる．

このような手法によって，すべてのユーザが USB メモリを利用することが可能であり，USB メモリを介した重要なファイルの漏洩を防止することが可能となる．

5 おわりに

本稿では，Thin Client システムと *Salvia* を協調させることによって強固なセキュリティを構築することについて述べ，そのための課題の一つとして外部記憶装置からの情報漏洩を防止する必要性について述べた．そして，外部記憶装置の中でも USB メモリに着目し，Thin Client システムでの USB メモリを介した情報漏洩を防止する手法について述べた．

参考文献

[1] 鈴来和久，一柳淑美，毛利公一，大久保英嗣 : Privacy-Aware OS *Salvia* におけるデータアクセス時のコンテキストに基づく適応的データ保護方式, 情報処理学会論文誌: コンピューティングシステム (ACS13). vol.47, No.SIG3, pp.1–15 (2006 年 3 月).

[2] FMV シンクライアントによるセキュリティ強化，情報漏洩対策の実現, FUJITSU JOURNAL. vol.32, NO.3, pp.2–7 (2006 年 3 月)

Thin Clientにおける USBメモリを介した 情報漏洩の防止手法

立命館大学 理工学研究科
毛利研究室
M1　岩永 真幸

はじめに

- 背景
- 目的
- Thin Clientについて
- 現在取り組んでいる課題
- 手法の提案
- 今後の課題
- おわりに

背景

- 近年、情報漏洩事件が多発している
  - 計算機や記憶媒体の盗難、紛失
  - 誤操作や管理ミスなどの内部の人間によるミス
  - 内部の正当なアクセス権限を持つものによる不正行為
  - 外部犯による不正な情報持ち出し
  - ウイルス
- プライバシアウェアOS ***Salvia***を開発している
  - 正当なアクセス権限を持つものによるミスや不正行為による データ漏洩を防止することを目的としている
- セキュリティの観点からThin Clientが注目されている
  - 計算機の盗難や不正行為による持ち出しによるデータ漏洩を 防止する
- ***Salvia***とThin Clientの協調による、より強固なセキュリティを 実現

Thin Clientのセキュリティ

- 端末からデータの持ち出しができない
  - 盗難によるデータの持ち出しを防ぐ
- アプリケーションを制限する
  - インストールするアプリケーションを管理できる
- セキュリティ対策を統一する
  - ウイルスチェック
  - ウイルス定義の更新
  - アプリケーションのバージョンアップ

Thin Clientの導入形式

- 画像転送方式
  - サーバ側ですべての処理が行われ、クライアント側に画像が送信される
  - クライアント側にデータが存在しない
- ネットワークブート方式
  - クライアント側が必要なデータをサーバ側から受信し、処理を行う
  - 実行時にデータがクライアント側に存在する

課題点

- Thin Clientシステムでの*Salvia*の状況変化
  - ユーザの位置変化
    - 位置を取得するサーバと利用するクライアントの位置が違うため、位置コンテキストをそのまま利用できない
  - データの位置変化
    - 外部記憶装置のデータがネットワークの先に存在する
    - ネットワークブート形式では、データがクライアント側に送信される
- Thin Clientのセキュリティ問題の解決
  - クライアント側の外部記憶装置から、データが漏洩する

現在取り組んでいる課題

- クライアント側の外部記憶装置から、データが漏洩する
  - 外部記憶装置へデータを書き込む
    - 重要なデータが書き込まれると漏洩の危険性がある
- 情報の漏洩を防ぐためには、外部記憶装置が利用できないのが理想である
- 業務上必要なため、外部記憶装置が利用可能なThin Client端末が存在する

企業の対策

- 富士通のPortshutter
  - 指定した特定メーカーの特定の外部記憶装置のみ使用可能なように設定可能
- デバイスの制御(日立など)
  - 端末やユーザによってデバイスの利用の可否を設定可能
- ユーザ認証の強化
  - 指紋認証、認証デバイスの利用など

問題点

- 外部記憶装置の利用者が限定される
  - 重要なデータを利用できないユーザに、外部記憶装置の利用を禁止すると業務に支障がでる
- 内部犯による情報漏洩
  - 利用できる外部記憶装置の特定や認証の強化では、防ぐことができない
  - 情報の漏洩事件は、内部犯が多い

開発環境

- ミレニアムシステムズ社のThin Client端末を利用している
  - 記憶装置はUSBメモリ
  - プロトコルはRDP
- Thin Clientの導入形式は、画像転送形式

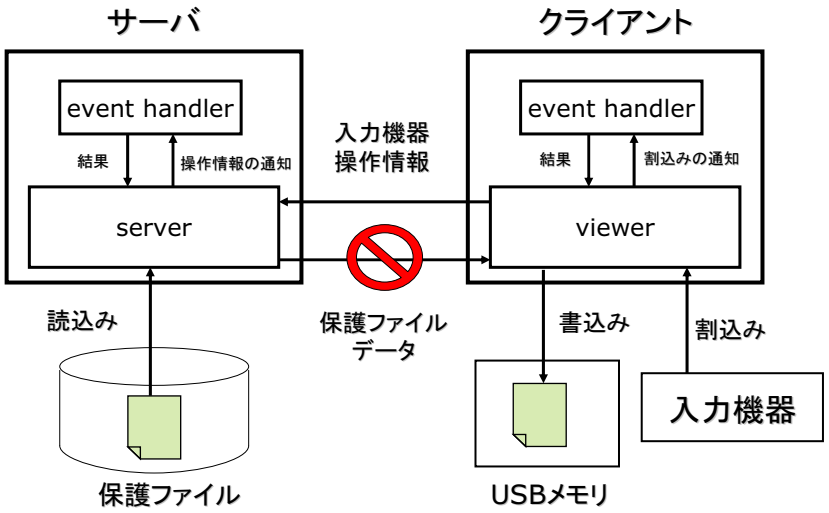
Thin Clientの調査(1/2)

- Thin Clientの動作を調査
  - パケット解析
  - VNCの解析
- サーバ
  - 画像データは定期的に送信する
    - 送信する画像データは、差分のみである
  - 音声データはヘッダをつけて送信される
- クライアント
  - マウス、キーボードの操作情報は、割込みが発生した時のみそれぞれヘッダをつけて送信する

Thin Clientの調査(2/2)

- ファイルの読み込み
  - USBメモリを挿すとUSBメモリの情報が送信される
    - デバイスの名前
      - Generic USB Flash Disk 0.00
    - OSやバージョン，ファイルシステム
      - MSDOS4.0 FAT16 など
    - ファイルの情報
      - ファイル名、アクセス権限など
  - ファイルにアクセスするとサーバにデータが送信される
- ファイルに書き込みを行う
  - 書き込みされたファイルを追加したファイル情報を送信する
  - 書き込むデータを送信する

手法の提案



今後の課題

- 外部記憶装置のデータをどのように扱っているかを調査する
  - rdesktopのソース解析
- システム構成を決定する
- プロトタイプの作成

おわりに

- 背景
- 目的
  - Thin ClientとSalviaとの協調
- Thin Clientについて
  - セキュリティ
  - Thin Clientの導入形式
- Thin ClientとSalviaを協調するための課題
- 現在取り組んでいる課題
  - 外部記憶装置からのデータ漏洩を防止する
  - 企業の対策と問題点
  - Thin Clientの調査結果
  - 手法の提案
- 今後の課題
  - Thin Clientシステムでの外部記憶装置の扱い
  - システム構成
  - プロトタイプの作成

確率的ルーティングアルゴリズム ARH の MANET への適応手法

岩田　元†　松尾　啓志†

† 名古屋工業大学大学院工学研究科 〒 466-8555 名古屋市昭和区御器所町
E-mail: †gensan@matlab.nitech.ac.jp, ††matsuo@nitech.ac.jp

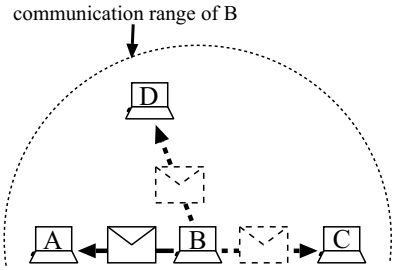


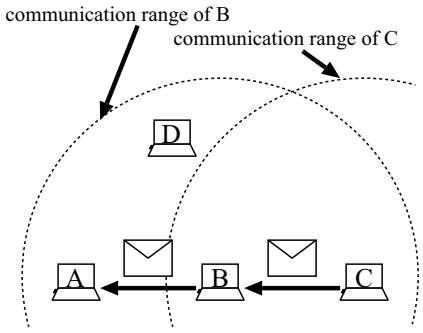
図 1　パケットの送信
Fig. 1　Sending packet.

1. はじめに

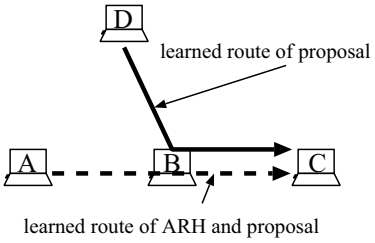
近年，無線端末の急速な普及により，MANET（モバイルアドホックネットワーク）[1] に期待が寄せられている．MANET では，無線基地局を介して通信を行う従来の無線ネットワークとは異なり，端末どうしが直接通信を行うことにより，通信設備を設置することなく無線ネットワークを構築可能である．そのため，災害地，僻地での利用やセンサーネットワーク，車々間通信などの大規模ネットワークから，屋内での利用や携帯型ゲーム機における通信対戦などの小規模ネットワークまで，様々な用途への適用が考えられており，MANET ルーティングプロトコルが多数提案されている [2]〜[5].

また，強化学習を応用した確率的手法を，MANET のような頻繁にトポロジが変化するネットワークでのルーティングに用いることが考えられている．中でも，ARH（Ant Routing with routing history）[6] は，経路の履歴を用い，使用した経路と逆向きの学習を行うことで効率的な学習を実現している．

ARH は，動的なネットワークへの適応を目的としたルーティングアルゴリズムである．しかし，ARH は，移動端末間の不安定な無線リンクによって構成される MANET 特有の環境を考慮していない．例えば，MANET の中でも，複数の端末が 1 個のチャンネルを共有し，無指向性の電波を用いて通信を行うネットワークでは，各端末は近隣端末が送信したパケットを傍受することが可能である（図 1）．また，MANET 環境では，パケットの受信が非常に不安定な領域（グレーゾーン）[7] が存在する．



(a) Forwarding packet.



(b) learned route.

図 2　経路の学習
Fig. 2　Route learning.

我々は，上記のような MANET 特有の環境を考慮した，ARH に対する改良手法を提案してきた [8]．しかし，文献 [8] では小規模な MANET を用いた実証実験しか行っておらず，詳細な検討がなされていなかった．

本稿では，我々が提案してきた手法を，シミュレーションを用いて評価する．

2. ARH

ARH は，強化学習を応用したルーティングアルゴリズムであり，中継端末を確率的に選択し，固定的な経路を確立しない．各端末は，パケットを受信した際に受信したパケットと逆向きの経路が使用されるように，中継端末の選択確率を学習していく．

経路選択　ARH では，端末 y がパケットを送信する場合，中継端末を，自身が持つルーティングテーブルに記録された確率値 $\mathrm{P}_y(d,z)$（y が，宛先が d であるパケットの中継端末として近隣端末 z を選択する確率）にしたがって，近隣端末から確率的に選択する．

経路の学習　経路の学習は受信したパケットと逆向きに行う．例えば，端末 A が図 2（a）に示す経路で転送されたパケットを受信した場合，端末 A は図 2（b）の破線の経路を学習する．

3. MANET に適応させるための改良方式

本章では，我々が提案してきた改良方式について述べる．

Hello パケットの相乗り　改良方式では，データパケットに Hello パケットを相乗りさせる手法を用いる．この手法では，各端末は，近隣端末が送信するデータパケットを傍受し，Hello パケットと同等に処理することで，一定期間パケットを送信していない場合にのみ Hello パケットを送信すれば良いため，制御メッセージの削減が期待できる．

リンクの安定性確認　改良方式では，Hello パケットの相乗りに加えて，Hello パケットに対して到達通知パケットを返し，Hello パケットが届いた端末を管理する手法を用いる．この手法では，各近隣端末から返送されてくる到達通知パケットから，一定割合以上の Hello パケットが届いているかどうかを調べることにより，各近隣端末が自身の安定した無線通信範囲内に存在しているかどうかを判定することができる．また，Hello パケット同様，到達通知パケットをデータパケットに相乗りさせることで，制御メッセージ数の増加を抑制した．

経路の学習　パケットを送信・中継する端末の近隣端末（図 2 の端末 D）は，送信されたパケットを傍受することができる．そこで，改良方式では，傍受したパケットも受信したパケットと同様に経路の学習に用いる（端末 D が図 2（b）の実線の経路を学習する）ことで，パケットを中継・受信した端末だけでなく，パケットを送信・中継した端末の近隣端末にも他の端末までの経路を学習させ，学習効率を高めている．

再送　改良方式では，到達通知を ACK として用い，パケットの再送を行っている．さらに，再送するときに，以下の処理を行っている．

（1） 送信に失敗したリンクに対して，負の学習を行う．
（2） 再送パケットの中継端末を再選択する．

これらの処理は，パケットが正しく送信できず，再送を行う場合，再送前に用いたリンクは信頼できない可能性があるため，再送前に用いたリンクの使用を抑制するために行う．本稿では，クロスレイヤアプローチを用い，MAC 層の再送機構に改良を施すことで，再送手法を実現した．

4. シミュレーション評価

提案手法を glomosim [9] に実装し，シミュレーション評価を行い，提案手法と，改良前の ARH，ARH に Hello パケット相乗りを追加した手法 1，手法 1 にリンクの安定性確認を追加した手法 2，手法 2 に学習の効率化を追加した手法 3，AODV を比較した．シミュレーション時間は 70 秒間とし，シミュレーション開始 5 秒後から 60 秒間，1 個の端末が別の端末に対して UDP パケットを用いて RTT の測定を行う．測定は 546 バイトのパケットを用い，秒間 30 回行った（したがって，通信帯域は往復で 256kbps となる）．

4.1 実　験　1

Random Way Point における移動速度を変化させ，AODV，ARH，手法 1〜3，提案手法における測定パケットの到達率を測定した．

試行を 100 回行い，中央値を求めた結果をスライド 15 枚目に示す．スライドから，手法 1 は ARH と比較して到達率が低下していることがわかる．これは，Hello パケットの相乗りにより，Hello パケット受信間隔が短くなり，リンクの切断検知が遅れるためである．

手法 2 では，移動速度が小さな場合は到達率が向上した．このことから，移動速度が小さな場合には到達通知が有効であるといえる．一方で，移動速度が大きな場合には，物理リンクが切断されても，古い到達通知が残ってしまうために，リンク切断検知に遅れが生じてしまい，到達率が低下する．

手法 3 を用いることで，到達率は向上した．このことから，学習の効率化を行うことで，宛先までの経路を効率的に発見できるようになり，ネットワークの動的な変化に対する適応力が向上したといえる．

提案手法では，更なる到達率の改善が見られた．したがって，提案手法では，信頼できないリンクの使用を抑制し，リンクの切断が原因で起こる再送においてもパケットを正しく転送できるといえる．しかし，端末が停止している状態（移動速度が 0m/s）では，到達率は低下した．これは，端末が停止している状態では，リンクの切断はほとんど発生せず，再送の原因は衝突が大半を占めるため，負の学習を行うことでルーティングテーブルの破壊を招いてしまうためである．

AODV では，固定的な経路を用い，MAC 層から送信失敗の通知を受け取って経路を無効化し，経路の修復を行う．したがって，経路が存在する間はパケットロスはほとんど起こらず，パケットロスの多くが経路修復中に発生するため，確率的に経路を選択する ARH と比較して高い到達率となった．

しかし，提案手法を用いることで，端末が移動する場合には，AODV と比較しても高い到達率を達成できることがわかる．

4.2 実　験　2

Hello パケットの最低送信間隔を変化させ，ARH，手法 1〜3，提案手法における測定パケットの到達率および制御メッセージ数を測定した．試行は 100 回行い，到達率は中央値を，制御メッセージ数は平均値を求めた．

実験結果をスライド 16 枚目に示す．

スライドから，手法 1 は ARH と比べて制御メッセージ数が削減されたことがわかる．このことから，Hello パケットの相乗りにより，制御メッセージ数を削減できたといえる．削減効果は，Hello パケットの送信間隔を短くすると相乗りできない Hello パケットが増加するため，Hello パケットの送信間隔が長いほど大きい．

手法 2 を用いると，Hello パケットの送信間隔が長い場合に，手法 1 と比較して制御メッセージ数は増加した．これは，Hello パケットの送信数が少ない場合には，他のパケットに相乗りできない到達通知が多くなるためである．

手法 3 では，手法 2 と比較して到達率が向上した．一方で，学習の効率化により，送信されるパケット数が減少し，Hello パケットや到達通知の相乗り数が減少してしまうため，制御メッセージ数は増加した．

提案手法では，手法 3 と比べて，到達率，制御メッセージ数ともに改善された．これは，手法 3 ではリンクの切断によって再送に失敗するパケットが正しく転送されるようになったためと考えられる．

4.3 実　験　3

提案手法を用いて，負の学習における学習率（Δp）を変化させ，4.1 節と同様に測定パケットの到達率の中央値を測定し，負の学習の効果を検証した．この時，$\Delta p = 0$ は負の学習を行わずに，再送時に中継先の再選択のみを行うことを意味し，$\Delta p = 1$ は負の学習により，選択確率を 0 にすることを意味する．

実験結果をスライド 17 枚目に示す．スライドから，端末が移動しない状態では，$\Delta p = 0$ の場合に到達率が最大となることがわかる．これは，4.1 節で説明したように，負の学習を行うことでルーティングテーブルの破壊が発生するためである．

端末が移動する状態では，$\Delta p = 0.1$ が最良となった．このことから，端末が移動する場合には，負の学習により，再送を効率良く行えるといえる．しかし，Δp を増加させると，ルーティングテーブルの破壊が発生し，到達率は低下した．

5. まとめと今後の課題

本稿では，我々が文献 [8] で提案した手法に対して，シミュレーションによる評価を行った．実験結果から，提案手法を用いることで，大規模な MANET においても，制御メッセージ数を削減させながら到達率を向上でき，AODV と比較しても良い到達率が得られることを示した．

しかし，文献 [8] で提案したリンク確認手法では，古い到達通知の影響により，精度の高い確認ができていないことがわかった．今後，より精度の高いリンク確認手法を検討する必要がある．

また，相乗りさせた Hello パケットや到達通知が失われたり，正常なパケット交換を妨げる場合があるなど，MAC プロトコルとの連携が取れていないこともわかった．今後，MAC 層に対して改良を施し，ARH と MAC 層がより緊密な連携を行う必要がある．

さらに，大規模なネットワークでは，学習速度が経路長に依存する問題がある．今後，提案手法に対して，経路長に依存しない強化学習手法を適用することを検討している．

文　献

[1] IETF MANET Working Group http://www.ietf.org/html.charters/manet-charter.html.

[2] Johnson, D. B., Maltz, D. A. and Hu, Y.-C.: *The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks*, IETF Internet-Draft (Jul. 2004).

[3] E.Perkins, C., Belding-Royer, E. M. and R.Das, S.: *Ad hoc On-Demand Distance Vector Routing*, IETF RFC 3561 (Jul. 2003).

[4] Clausen, T. H. and Jacquet, P.: *Optimized Link State Routing*, IETF RFC 3626 (Oct. 2003).

[5] Toh, C.-K.: Associativity-Based Routing For Ad-Hoc Mobile Networks, *Wireless Personal Communications*, Vol. 4, No. 2, pp. 103–139 (Dec. 1997).

[6] 斎藤享，松尾啓志：動的な環境下における履歴情報を用いた確率的ルーティング，電子情報通信学会技術研究報告. SST，スペクトル拡散，Vol. SST2001-168, pp. 289–296 (Mar. 2002).

[7] Lundgren, H., Nordstrm, E. and Tschudin, C.: Coping with communication gray zones in IEEE 802.11b based ad hoc networks, *5th ACM international workshop on Wireless mobile multimedia (WoWMoM 2002)*, ACM Press, pp. 49–55 (2002).

[8] 岩田元，松尾啓志：確率的ルーティングアルゴリズムＡＲＨを用いた無線ＬＡＮ環境におけるストリーミング配信実験，情報処理学会研究報告,2006-MBL-37, pp. 91–96 (2006).

[9] UCLA Parallel Computing Laoratory: Glomosim. http://pcl.cs.ucla.edu/projects/glomosim/.

確率的ルーティングアルゴリズム ARHのMANETへの適応手法

名古屋工業大学
岩田 元
松尾 啓志

背景

- MANET（モバイルアドホックネットワーク）が注目されている
  - 固定的なインフラを必要としない
  - 一時的なネットワークを容易に構築可能
  - MANETルーティングアルゴリズムが必要

- 強化学習を応用したルーティングアルゴリズム
  - 確率的に経路を選択
    - 選択する経路を強化学習により学習
  - 動的な変化に対する適応力に優れる

目的

- ARHをMANETに適応させる手法を提案してきた
  - 小規模な環境での実験・評価しか行っていない

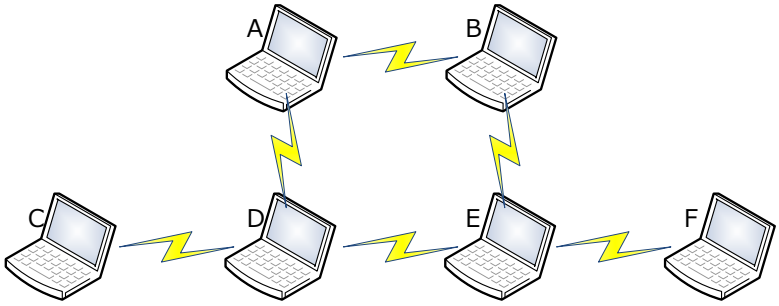
- 大規模なネットワークを用いて提案手法を評価する
  - シミュレーションを用いる
  - 詳細に考察する

ARH: Ant Routing with routing history

3

ARH(Ant routing with routing history)

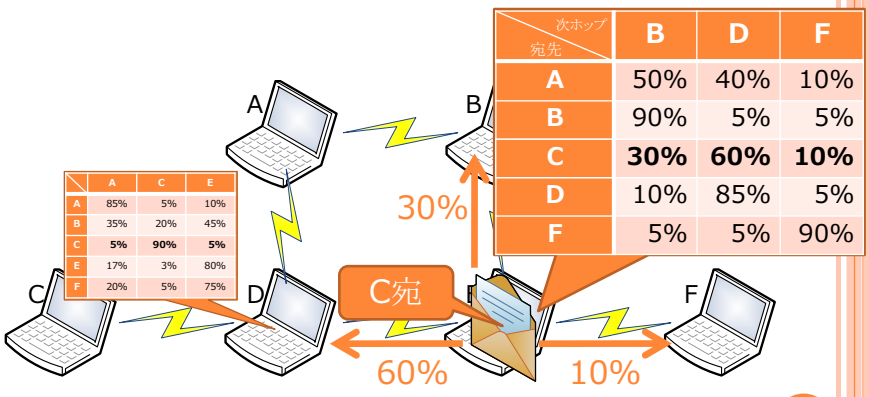
- 強化学習を応用
  - 確率的に経路を選択
- 使用した経路と逆向きの経路を学習



4

経路の選択

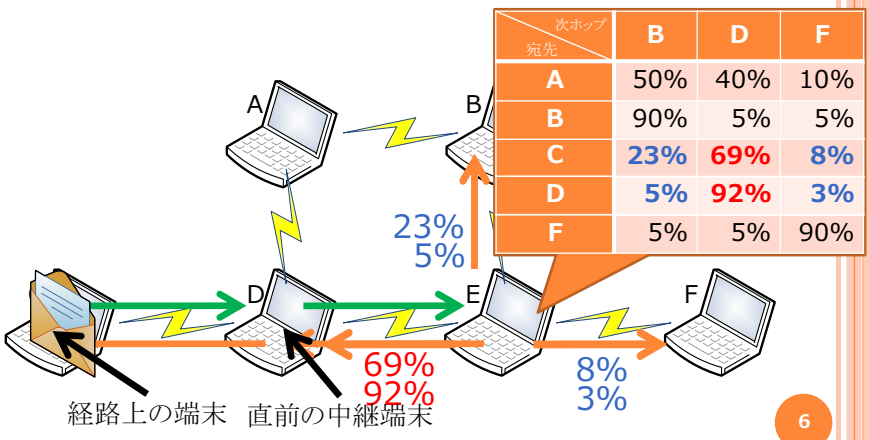
- 中継端末をルーティングテーブルに従って確率的に選択



5

経路の学習

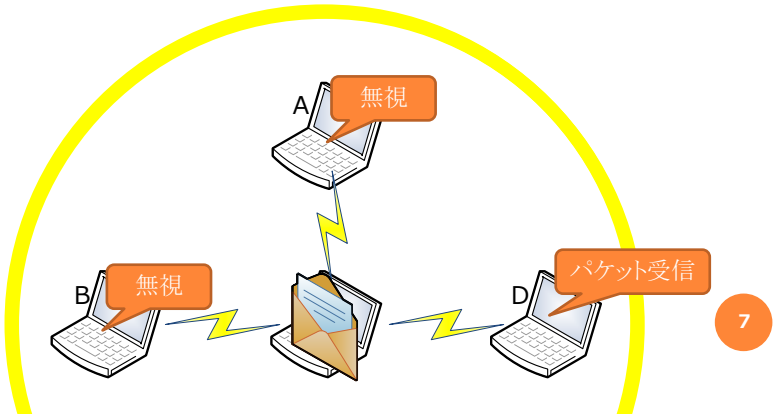
- 受信したパケットと逆向きの経路を学習



6

MANET環境の特徴

- 近隣端末は送信されたパケットを傍受できる
- 電波到達領域にグレーゾーンが存在



7

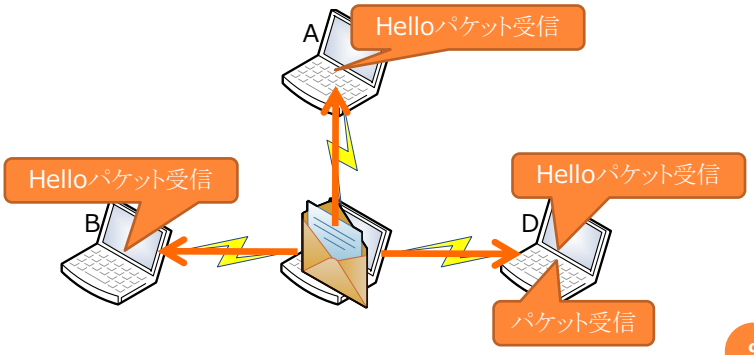
提案手法

- Helloパケットを他のパケットに相乗りさせる
  - 制御メッセージ数の削減
- リンクの安定性を確認する
  - パケット到達率の向上
- 傍受したパケットを学習に利用する
  - 学習の効率化
- 再送を行う
  - パケット到達率の向上

8

Helloパケットの相乗り

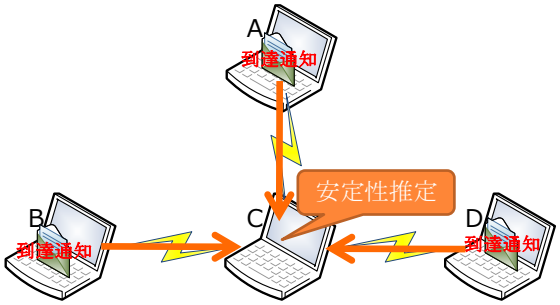
- 傍受したパケットをHelloパケットとして用いる



9

リンクの安定性確認

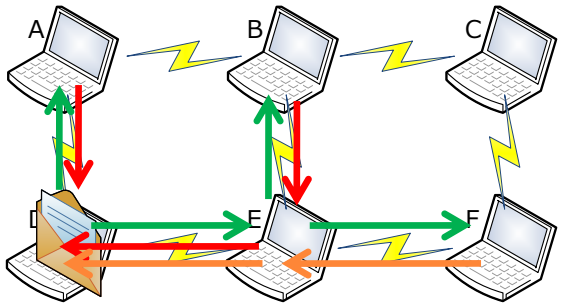
- Helloパケットに対して到達通知を返す
  - 到達通知は他のパケットに相乗りさせる
- 到達通知からリンクの安定性を推定する



10

学習の効率化

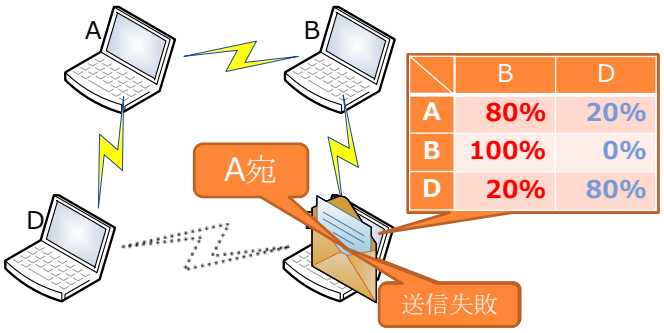
- 傍受したパケットも学習に用いる



11

再送の効率化

- 再送毎に中継端末を選択
- 負の学習を行う
- MAC層の再送機構を改良することで実現



	B	D
A	80%	20%
B	100%	0%
D	20%	80%

12

シミュレーション評価

- glomosimを使用
- 実験環境

シミュレーションエリア	1km×1km
ノード数	100台
送信半径	約200m
モビリティ	Random Way Point
移動速度	15m/s
シミュレーション時間	70秒間

- 1個の端末が別の端末に対してRTTを測定
  - UDPを使用
  - シミュレーション開始5秒後から60秒間
  - 30回/秒
  - 往復で256kbps

13

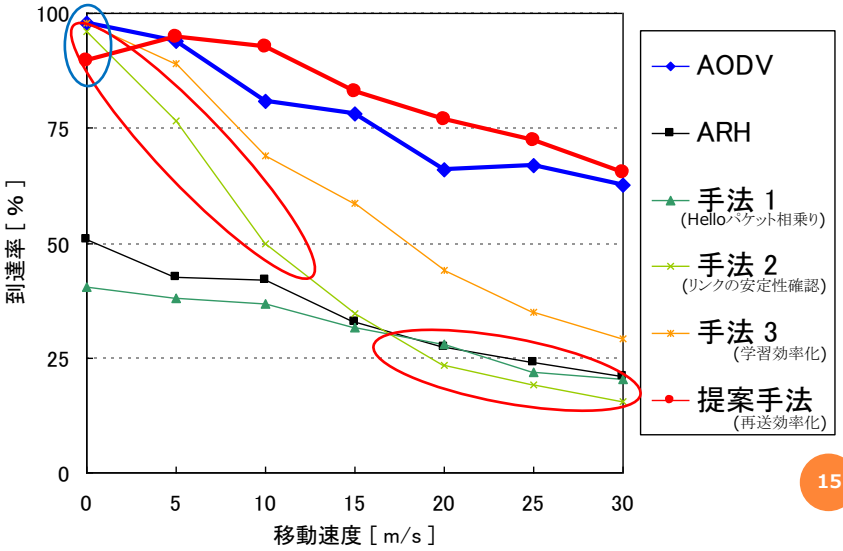
評価

- 試行は100回
- 以下の項目を測定
  - 測定パケットの到達率(中央値)
  - 制御メッセージ数(平均値)
- 以下の手法を比較
  - AODV
  - ARH
  - 手法1：ARH + Helloパケット相乗り
  - 手法2：手法1 + リンクの安定性確認
  - 手法3：手法2 + 学習の効率化
  - 提案手法：手法3 + 再送の効率化

14

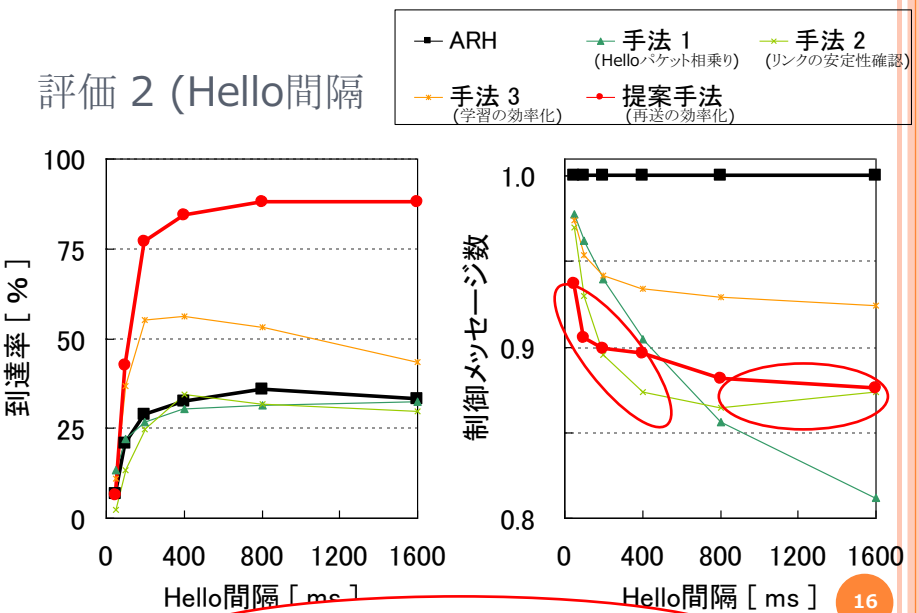
AODVと比べて最大12%向上

評価 1 (移動速度)



15

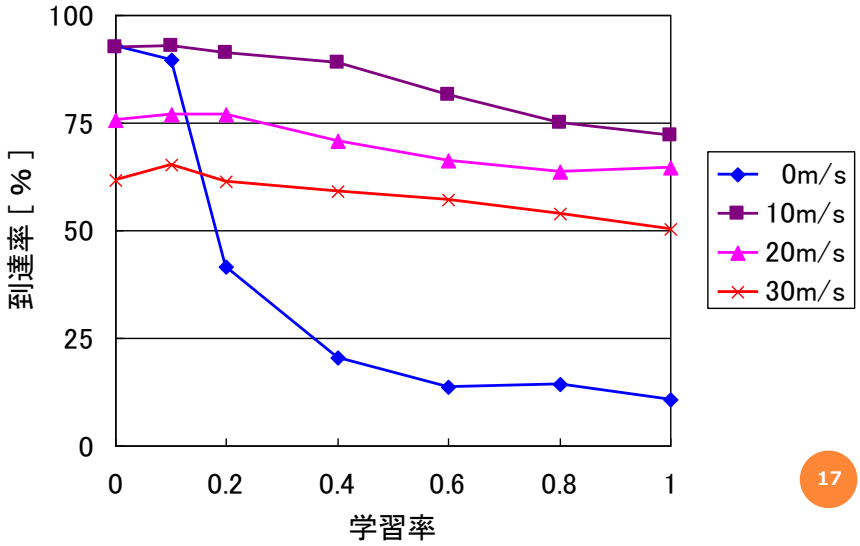
評価 2 (Hello間隔



16

制御メッセージ数：最大12%削減
到達率：　　55%向上

評価 3 (負の学習)



17

まとめと今後の予定

- 以下の手法を大規模なネットワークを用いて評価
  - Helloパケットの相乗り
  - リンクの安定性確認
  - 学習の効率化
  - 再送の効率化
  - 提案手法では、到達率の向上と制御メッセージ数削減を両立
    - AODVと比較しても到達率向上

- 今後の予定
  - より精度の高いリンク確認手法の提案
  - MAC層と緊密な連携を行うこと
  - 経路長に依存しない強化学習手法の適用

18

複数の無線基地局を用いた QoS 制御システムにおける適応的接続制御アルゴリズムの考察

水野 邦彦 †
† 立命館大学大学院理工学研究科

1 はじめに

無線技術の発展は，IEEE802.11a, g, n などの高速通信を実現し，無線 LAN が利用されている場所も増えている．また, その適用場面も，メール送受信や Web の閲覧だけでなく，動画像のストリーミング，VoIP(Voice over IP) などの音声通信，ユビキタスコンピューティングなど，大きな広がりを見せている．

しかし，このようなコンテンツの増加と大容量化，QoS(Quality of Service) 制御の必要性の視点から解決しなければならない課題は多い．例えば，現在普及している無線 LAN では，QoS に関して十分な考慮がされていないため，動画や音声のようなリアルタイム性の高い通信や帯域を多く必要とする通信を行う際に，品質が低下する場合がある．また無線 LAN では，基地局と端末が，特定の周波数であるチャネルを共有して通信を行うため，1 台の端末がチャネルを使用している間，同一のチャネルを使用する他の端末は，送信を見合わせることとなる．よって，1 台の基地局に対して多数の端末が接続された場合，輻輳が発生し，各端末の QoS 要件を満たせなくなる可能性がある．これを解決するために複数の基地局を配置する場合があるが，端末が特定の基地局に集中し，全体としてのスループットが向上しなかったり，端末毎に通信状況が大きく異なってしまったりするという問題がある [1]．そこで，我々は，複数の基地局を配置することによって全体としてのスループットの向上を狙い，さらに，QoS を考慮しながら，基地局と端末の接続を動的に制御するシステム [2] を提案している．

2 想定環境

本システムにおける想定環境を図 1 に示す．本システムでは，ある特定のエリア内に複数の基地局を配置する環境を想定している．ただし，適応的に基地局と端末の接続を制御するために，各基地局の通信可能な範囲がそれぞれ重なるように配置する．また，各基地局は 1 つ以上の無線 NIC が装着されており，互いに干渉しないようにチャネルを割り当てることとする．一方で，各基地局は，同じネットワークセグメント上に Ethernet で接続される．この Ethernet 上では，端末上で動作するプロセスによる通信と，各基地局同士が協調に必要なデータの通信が行われる．

エリア内には，複数の無線端末が存在する．各端末の位置によって，接続可能な基地局群やそれらの電波状況は異なる．端末は，一般的に，最も電波状況の良い無線基地局に接続するため，1 章で述べたように特定の基地局に端末が集中する場合がある．本システムでは，そのような場合に，基地局から端末へ，基地局を切り替えるよう指示する．端末は，それに従うことで QoS を維持することができる．1 では，AP2 に CL1 と CL2 が集中していた場合に，CL1 を AP2 へ切り替えることでそれを実現している例である．

3 設計方針

本システムでは，QoS に影響を与えるパラメータとして帯域に着目し，帯域要求を満たすことで QoS 満足度が最大になるとする．しかし，帯域は，端末の電波状況や基地局のトラフィック状況に影響を受ける．そこで，全基地局の中から代表となる基地局を 1 台選出し，代表基地局においてこれらの値の変動をまとめて監視し，適応的に QoS 制御を行う．代表基地局では，各端末の接続可能な基地局における電波品質から最大通信速度を求め，要求帯域を保証するために必要な割付時間割合を求める．その時間割合の中で，帯域要求を受け入れ可能な基地局において，帯域を予約する．帯域要求が受け入れ不可能の場合は，帯域予約を行っているプロセスが終了し，時間割合を確保できるまでベストエフォートによる通信とする．以上を実現するための，基地局と端末におけるシステム要件を次に示す．

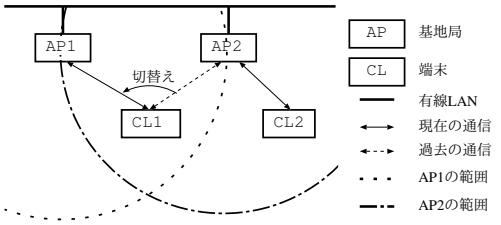


図 1 QoS 制御システムが想定する環境

■帯域予約 各ユーザが利用するプロセスによって必要とする帯域は異なるため，プロセス毎の要求帯域を接続先となる基地局において予約する必要がある．代表基地局は，要求帯域の予約可能な基地局を選択し，その基地局のパケットスケジューラに帯域を予約する．基地局において，要求帯域を満たないプロセスもパケットスケジューラに帯域を予約する．そして，そのプロセスの予約割合は，要求帯域のあるプロセスの予約割合に応じて変動し，ベストエフォートによる通信とする．

なお，本システムを実装していない端末が接続された場合を考慮し，基地局と通信を行っている全端末のプロセスを監視することで, その端末のプロセスもパケットスケジューラに帯域を予約する．このプロセスの予約割合も変動し，ベストエフォートによる通信とする．

■フロー制御 帯域が変動する中で，予約された帯域を保証する仕組みが必要となる．そのため，基地局と端末の双方のパケットスケジューラで帯域を予約することで，基地局から端末への通信だけでなく，端末から基地局への通信にもフロー制御を行い，予約帯域を保証する．

■代表基地局の選出 各基地局の情報や各端末の情報をまとめて監視するために，代表となる基地局を選出する必要がある．各基地局に同一ネットワークセグメントの IP アドレスを割り当て，その IP アドレスが最も大きい基地局が代表基地局とする．

■通信状況の収集 各基地局と各端末において，フロー制御行うために，刻々と変化する状況を常に監視する必要がある．基地局は，接続されている全端末の全プロセスが送受信するパケットを監視し，通信状況を把握する．端末は，端末情報として，全プロセスが送受信するパケットや接続可能な基地局の電波品質を監視し，定期的に基地局へ送信する．そして，各基地

局は，基地局自身の情報と各端末情報を代表基地局に送信する．

■適応的接続制御アルゴリズム　QoS 満足度を最大化させるためには，代表基地局で各端末の電波状況，各基地局のトラフィック状況と予約帯域を考慮して，よりユーザの要求帯域を満たすことの可能な基地局と端末の組合せを求める必要がある．端末が接続先を切り替えることで，電波状況，トラフィック状況の改善，予約帯域の分散が可能となる．

■接続先基地局の切替えメカニズム　端末が接続先の基地局を変更する場合，端末は，適応的接続制御アルゴリズムによって選択された基地局に切り替える必要がある．端末は，接続先の基地局の MAC アドレスを設定することで，ローミング通信による切替えを行う．

4　適応的接続制御アルゴリズム

代表基地局は，端末の接続可能な基地局の電波品質から，接続先基地局での予約帯域の割付時間割合を求め，予約が可能か判断する．ユーザからの要求帯域を $R[bps]$，端末の電波品質を $Q[\%]$，電波品質から求まる最大通信速度を $f(Q)[bps]$，割付時間割合を $C[\%]$ とし，式 $C = \frac{R}{f(Q)} \cdot 100$ によって帯域の予約が可能か判断する．各基地局の C の値が 100% を超えない限り，帯域の予約が可能である．

適応的接続制御アルゴリズムでは，この割付時間割合を使用して，端末の接続先を決定し，端末の接続切替先の基地局に予約帯域を移行させ，端末に対して接続先基地局の切替えを指示する．これによって，端末の QoS 満足度をシステム全体で最大化させることが可能となる．しかし，実際には，切替え処理によって一時的な接続断が発生するため，その前後で遅延や帯域の低下が発生してしまうことが予想できる．また，電波品質の揺らぎや新たに参加する端末に対応するため，基地局ごとの合計割付時間割合は，なるべく低く抑えることが望ましいと考えられる．以上より，本システムでは，アルゴリズム評価の観点として，基地局毎の合計割付時間割合の中で最大の値 (以下，最大割付割合と記す) と，接続先基地局の切替え回数を考える．ここで総当たりによって接続先決定を決定する方法があるが, 計算回数が膨大となるため, 適当では無いと考えられる．そこで, 2 つの評価観点のそれぞれに着目したアルゴリズムを検討している．

両方のアルゴリズムに共通する基本方針は次のとおりである．

- 割付時間割合の合計値がより高い基地局から低い基地局へ切り替える．
- 切替え元基地局における合計割付時間割合と比較して，切替え後の切替え先基地局における合計割付時間割合が大きい場合は，切替えを行わない．

上記を基に，最大割付割合を低く抑えることを目的とするアルゴリズム (A) と，切替え回数を少なく抑えることを目的とするアルゴリズム (B) を作成した．次に, それぞれの動作を示す．

(A)　最大割付割合を低く抑えることを目的としたアルゴリズム
1. 各端末が最良の電波状況となる基地局に接続された状態を基とする．
2. より割付時間割合が大きい基地局に接続されている端末で，切替えによって増加する端末の占める割付け時間割合が，最小である基地局への切替えを候補とする．
3. 切替えを実行したと仮定し，状態の再計算を行う．
4. 切替え可能な組合せが無くなるまで (2)，(3) を繰り返す．
5. 最終的に求められた組合せに従い，切替えを実行する．

(B)　切替え回数を少なく抑えることを目的としたアルゴリズム
1. アルゴリズム実行時の，各端末と各基地局の組合せを基とする．
2. 各基地局の合計割付時間割合が，任意に設定可能な閾値以上であるか確認し，そうでない場合はアルゴリズムを終了する．
3. 基地局内でより割付け時間割合が大きい端末について，合計割付時間割合がより小さい基地局への切替えを候補とする．
4. 切替えを実行したと仮定し，状態の再計算を行う．
5. 各基地局の合計割付時間割合が閾値以下となるか，切替え可能な組合せが無くなるまで (3)，(4) を繰り返す．
6. 最終的に求められた組合せに従い，切替えを実行する．

5　評価

シミュレーションによって各適応的接続制御アルゴリズムの性能評価を行った．基地局 3 台, 端末 10 台の環境を想定し, 10% から 45% の範囲で割付時間割合をランダムに与え，何も処理を行わない場合, 最大割付割合が最も低い場合との比較を行った．その結果, 最大割付割合の平均は, 処理をしない場合が 139.79%, 最も低い場合が 65.87% であったのに対し, アルゴリズム (B) が 101.97%, アルゴリズム (A) が 68.02% となった．また, 切替え回数の平均は．最大割付割合が最も低い場合 6.67 回, アルゴリズム (A) が 6.72 回, アルゴリズム (B) が 2.92 回となった．このように, アルゴリズム (A) では, 最大割付割合が最も低い場合と近い結果が得られた．アルゴリズム (B) では, 切替え回数が低く抑えられ, 最大割付割合も, 処理をしない場合と比較して改善した．連続して使用する場合には, 切替え回数の面でアルゴリズム (B) が, 状態を大きく改善する場合には, アルゴリズム (A) が有効であると考えられる．

6　おわりに

本稿では, 複数の無線基地局を用いた QoS 制御システムについて述べた．また, 本システムにおいて用いられる適応的接続制御アルゴリズムについて述べ, その評価を行った．今後は, 2 種のアルゴリズムの使い分けについて検討し, アルゴリズムの改良を行う．その後, 本システムの実装を完了し, システム全体の評価を行う．

参考文献

[1] Anand Balachandran, Geoffrey M. Voelker, Paramvir Bahl, P. Venkat Rangan: Characterizing user behavior and network performance in a public wireless LAN, ACM SIGMETRICS Performance Evaluation Review, pp. 195–205, 2002.

[2] 伊藤 淳，水野 邦彦，毛利 公一: 複数の無線基地局を用いた QoS 制御システムにおける通信チャネル制御方式，情報処理学会研究報告 2007-MBL-40/2007-UBI-13，Vol. 2007，No. 14，pp. 91-98，2007.

複数の無線基地局を用いた QoS 制御システムにおける 適応的接続制御アルゴリズムの考察

立命館大学大学院 理工学研究科
毛利研究室
水野 邦彦

1

発表内容

- 背景
- 複数の無線基地局を用いたQoS 制御システム
- システム概要
- システム要件
- 適応的接続制御アルゴリズム
- 評価
- おわりに

2

背景

- 無線LANではQoS(Quality of Service)の考慮が不十分
  - 干渉や減衰などにより一定とならない帯域
  - リアルタイム性の高い通信,帯域を多く必要とする通信の品質低下
- 無線LANでは特定の周波数であるチャネルを共有して通信
  - 同一のチャネルが使用されている間, 端末は送信を待機
  - 1つの基地局に多数の端末が接続された場合,
    待ち時間が長くなり各端末のQoS要件を満たせない

- 複数の無線基地局を用いたQoS制御システム
  - 複数基地局・複数チャネルを用い,適切な接続制御によって
    要求に合わせた資源配分を行うシステム

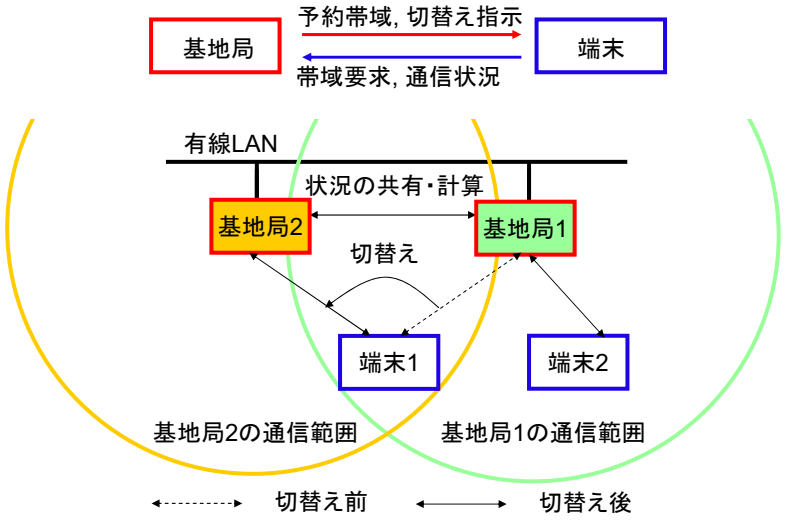
3

複数の無線基地局を用いたQoS制御システム

- QoS満足度の総和を最大化
  - 各端末のプロセスごとに存在
  - 要求する帯域をどれだけ満たしているか
- 複数のチャネルを用いる
  - 1つのチャネルを用いた場合と比較して、多くの帯域を使用可能
- 複数の無線基地局を配置する
  - 端末は, 電波状況の良好な基地局や
    端末のQoS要件を満たす基地局を選択することが可能
- 基地局と端末が協調する
  - 通信状況と要求帯域に応じて各端末の接続先を選択し,
    要求帯域を満たすことが可能

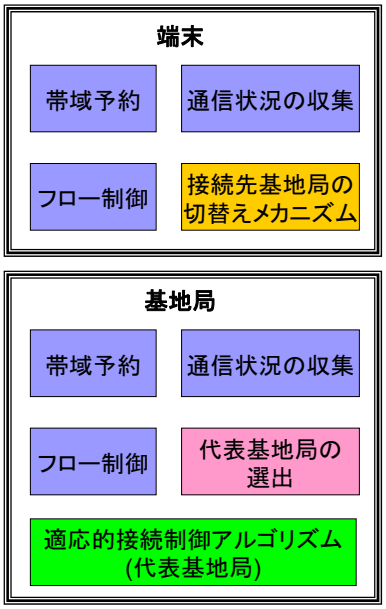
4

システム概要



5

システム要件

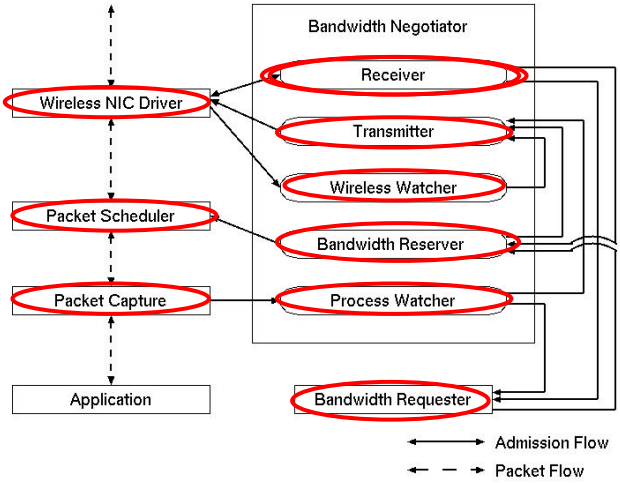


- 帯域予約
  プロセスごとの帯域予約
- フロー制御
  予約された帯域を保証
- 通信状況の収集
  フロー制御のため, 通信状況, 要求帯域, 電波状況を監視
- 代表基地局の選出
  各端末の情報や各基地局の情報をまとめて監視する基地局
- 適応的接続制御アルゴリズム
  端末と基地局の組み合わせを求め, QoS満足度を最大化
- 接続先基地局の切替えメカニズム
  アルゴリズムによって選択された基地局に切り替える

6

ソフトウェア構成

- 端末のソフトウェア構成

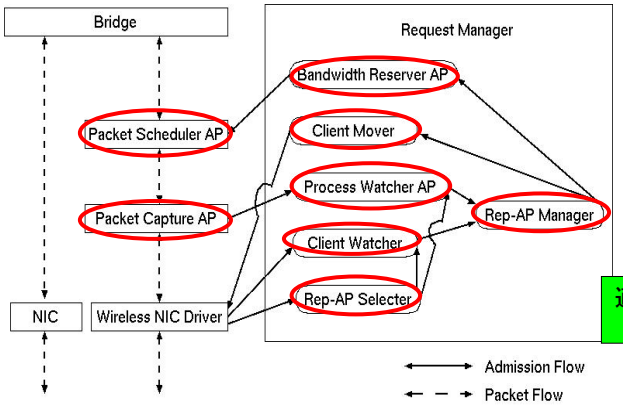


帯域予約
フロー制御
通信状況の収集
接続先基地局の切替えメカニズム

7

ソフトウェア構成

- 基地局のソフトウェア構成

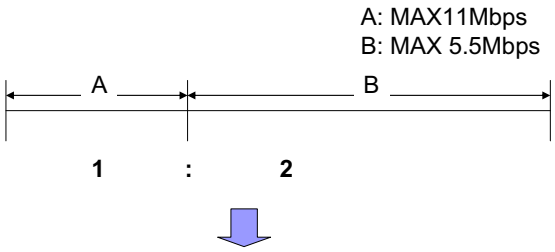


帯域予約
フロー制御
通信状況の収集
代表基地局の選出
適応的接続制御アルゴリズム(代表基地局)

8

資源配分

- 無線LANの通信速度は電波品質に影響を受ける
- 他の端末の通信速度によってスループットが変化する
  - 図に示すA, Bが機会均等で通信を行った場合,
    それぞれの占める時間の割合は1:2となり,
    通信速度は双方とも約3.7Mbpsとなる



- 時間を単位として各端末に資源配分を行う

9

割付時間割合

- 端末の帯域要求が満たされる単位時間当たりの割合
- 各端末と基地局の組合せの電波品質から要求帯域を満たす割付時間割合を求め, それを基に接続先を決定
  - ユーザからの要求帯域[bps] = R
  - 端末の電波品質[%] = Q
  - 電波品質から求まる最大通信速度[bps] = $f(Q)$
  - 割付時間割合[%] = C

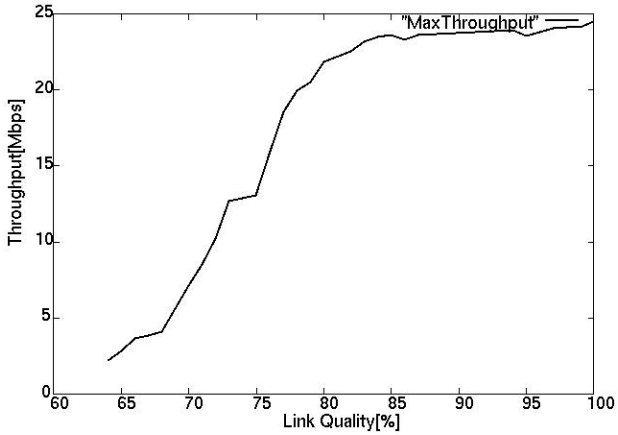
$$C = \frac{R}{f(Q)} * 100$$

10

割付時間割合

- IEEE802.11a (Intel PRO/Wireless 2915ABG) における電波品質とTCP通信での最大通信速度の関係

$f(Q)$



Q

11

適応的接続制御アルゴリズム

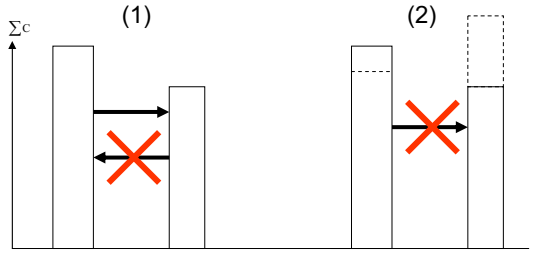
- 総当りによる接続先決定は適当でない
  - 端末数n, 基地局数mとして m^n 通りの組合せが考えられる
  - 計算数を抑えつつ適切な組合せを決定するアルゴリズムが必要
- アルゴリズム評価の観点
  - 基地局ごとの合計割付時間割合の中で最大の値（最大割付時間割合）が低い
    - 電波品質の揺らぎや新規に参加する端末に対応するため
  - 無線基地局の切替え回数が少ない
    - 切替えによって一時的な接続断が発生するため
- 評価観点を考慮した以下の2つのアルゴリズムを検討
  - A) 最大割付割合を低く抑えることを目的としたアルゴリズム
  - B) 切替え回数を少なく抑えることを目的としたアルゴリズム

12

アルゴリズムに共通する基本方針

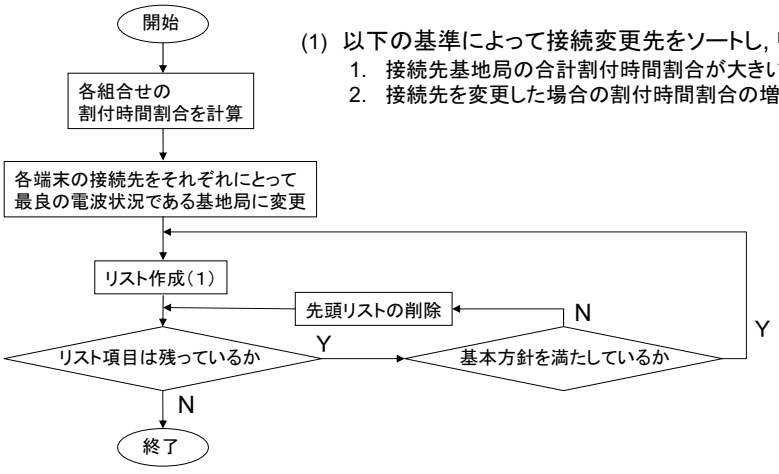
(1) 合計割付時間割合が大きい基地局から
小さい基地局に対して切替え

(2) 切替え元の
合計割付時間割合 $\leq$ 切替え後の切替え先の
合計割付時間割合
となる場合は切替えない



13

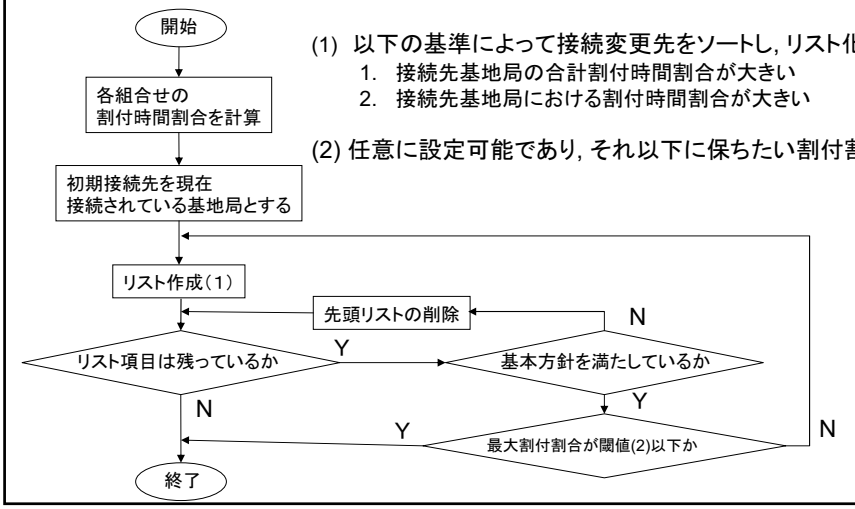
(A):最大割付割合を低く抑えるアルゴリズム



(1) 以下の基準によって接続変更先をソートし, リスト化する
1. 接続先基地局の合計割付時間割合が大きい
2. 接続先を変更した場合の割付時間割合の増加が小さい

14

(B):切替え回数を少なく抑えるアルゴリズム



(1) 以下の基準によって接続変更先をソートし, リスト化する
1. 接続先基地局の合計割付時間割合が大きい
2. 接続先基地局における割付時間割合が大きい

(2) 任意に設定可能であり, それ以下に保ちたい割付割合を表す

15

評価

- シミュレーションによる評価
  - 最大割付割合と切替え回数を評価項目とする
- 比較対象としてアルゴリズム(best)を考える
  - 総当りによって求める,最大割付割合が最も小さく,
    かつ切替え回数がより少ない組合せ
- パラメータ
  - 端末数:10台
  - 基地局:3台
  - 割付時間割合:10%～45%の範囲でランダム
  - 試行回数:1000回
  - (B)の閾値: 0.8

16

結果

- 平均最大割付割合
  - (A):68.02%
  - (B):101.97%
  - (best):65.87%
  - (処理を行わない場合)139.79%
- 平均切替え回数
  - (A):6.72回
  - (B):2.92回
  - (best):6.67回
- 平均リスト個数
  - (A)13.06個×2.86回のループ
  - (B)7.26個×3.31回のループ
  - (best)3^10=59049個

17

各接続の割付時間割合が等しいモデル

各組合せにおける割付割合C(%) $C=\frac{R}{f(Q)}*100$

	基地局1～5
端末1～10	25

- 初期接続先として以下の2種を検討する
  - α : 全ての端末が基地局1に接続されている
  - β : 各基地局に対して均等に接続されている

18

各接続の割付時間割合が等しいモデル

初期状態	アルゴリズム	切替え回数[回]	最大割付割合[%]
α	(A)	8	50
α	(B)	7	75
α	(best)	8	50
β	(A)	8(期待値)	50
β	(B)	0	50
β	(best)	0	50

- 初期状態がαの場合, 各アルゴリズムは問題無く働いている
- 初期状態がβの場合, 最大割付割合の面では(A)(B)共に最善であるが, (A)は切替え回数が平均8回となり余計な切替えが発生している

19

合計割付時間割合が等しく 状態の悪い接続先に接続されているモデル

各組合せにおける割付割合C(%)と初期接続先 $C = \frac{R}{f(Q)} * 100$

	基地局1	基地局2	基地局3	初期接続先
端末1	25	50	75	基地局3
端末2	75	25	50	基地局1
端末3	50	75	25	基地局2
端末4	25	50	75	基地局3
端末5	75	25	50	基地局1
端末6	50	75	25	基地局2

- 各基地局の合計割付時間割合が150%と等しい
- 各端末の初期接続先がそれぞれにおいて 最も状態の悪い接続先となっている

20

合計割付時間割合が等しく
状態の悪い接続先に接続されているモデル

アルゴリズム	切替え回数[回]	最大割付割合[%]
(A)	6	50
（B）	0	150
(best)	6	50

- アルゴリズム(A)は，アルゴリズム(best)と等しい結果が得られた
- アルゴリズム(B)は，切替えを実行することが不可能であり，
最大割付割合が150%となっていることから，QoS要求を満たせなかった

21

考察

- アルゴリズム(A)
  - 計算回数を抑えつつ(best)と近い結果が得られた
    - 最大割付割合を低く抑えるという目的において十分である
  - 通常運用にあたっては，切替え回数が多いことが問題となる
    - 良い接続先が選択されている場合でも再計算を行うため
- アルゴリズム(B)
  - 切替え回数が低く抑えられる
  - 割付時間割合に余裕のある場合，
最大割付割合もほぼしきい値以下となる
  - 各基地局における合計割付時間割合に差が無い場合，
接続先に問題があっても状況を改善できない可能性がある
- 2つのアルゴリズムを使い分ける
  - 通常はアルゴリズム(B)で運用
  - 状況が改善しきれない場合にアルゴリズム(A)を用いる

22

おわりに

- 複数の基地局を用いたQoS制御システム
  - 複数基地局・複数チャネルを用い，
    端末の要求に応じ適応的に接続を制御しQoS保証を行う
- 適応的接続制御アルゴリズム
  - 最大割付割合を低く抑えるアルゴリズム
    - 現在の状態によらず最善の電波状況である基地局から割付割合の増加が少ない端末を移動
    - 最大割付割合は十分低いが，切替え回数が多い
  - 切替え回数を少なく抑えるアルゴリズム
    - 現在の状態を基とし多くの割付割合を占めている端末を移動
    - 切替え回数は少ないが，状態によっては最大割付割合が高くなる
  - 2つのアルゴリズムを使い分けることにより，互いの欠点を補える可能性
- 今後の課題
  - 実機を用いてのシミュレーション結果の確認
  - 2つのアルゴリズムの切替え条件の検討
  - 端末の切替え条件に対し，他のパラメータによる補正の検討

23

広域ネットワークの統計情報の解析および可視化

芝 公仁† 寺元 慎二†
†龍谷大学理工学部

1 はじめに

インターネットにおいて，ネットワークの効率的な利用やサービスの質の向上を実現するためには，通信するノードや使用する経路を適切に選択し通信を行う必要がある．このとき，良いネットワークリンクを選ぶためには，各リンクのパケット伝送遅延や帯域幅などの特性を予め知っておく必要がある．このような情報の収集を目的に，インターネットのネットワーク特性を測定する研究が多く行われてきている．これらの研究では，インターネット上のすべてノード間のリンクの測定が困難であることから，通常，一部のリンクのみを測定し，その結果をもとに他のリンクの特性を推定することが行われる．既存の研究の多くは，できるだけ正確な測定・推定を行うことに注力しており，測定・推定の結果を有効に活用する手法については十分に検討されていない．

インターネットを対象とした測定･推定の結果は大量のデータとなり，これを理解することは容易ではない．そこで，本研究では，広域ネットワークの測定･推定結果を解析するための可視化手法について検討し，可視化ツールの構築を行った．本可視化ツールは次の特徴を持つ．

- 多数のノードが物理的に広範囲に広がって形成されるネットワークに対応可能である．
- 多くの環境で動作可能であり，多様な目的に対応できる柔軟な可視化機能を有する．

以下，本稿では，これらの特徴を持つ可視化ツールを用いた，広域ネットワークのノード間特性の可視化について述べる．

2 ネットワーク特性の測定

本研究で構築した可視化ツールが可視化するデータは，PlanetLab[1] 上で動作する S3（Scalable Sensing Service）[2] で収集されたものである．PlanetLab は分散コンピューティングのためのネットワークテストベッドであり，企業や大学などが提供する計算機を利用して，インターネット上にオーバーレイネットワークを構築している．PlanetLab のノードは世界中に広がっており，それらがインターネットによって接続されている．このような特徴から，PlanetLab のノードを標本とし，それらのノード間のリンクを調べることによって，インターネットの特性を調べようとする試みが行われてきている．

Visualization of Link Qualities of a Large Network
Masahito Shiba† and Shinji Teramoto†
†Faculty of Science and Technology, Ryukoku University

S3 は，このような試みの一つであり，PlanetLab のノードを利用してインターネットの測定を行っている．また，S3 では，ノード間での測定の結果を元に，推定も行う．例えば，パケット伝送遅延時間であれば，すべてのノード間のリンクで測定するのではなく，一部のリンクでのみ測定を行い，その結果をもとに測定していないリンクのパケット伝送遅延時間を推定する．

S3 が測定・推定する特性には，パケット伝送遅延時間に加え，ネットワーク帯域幅やパケットロス率がある．S3 は，定期的に，測定・推定したすべてのデータから一つのスナップショットを作成する．スナップショットは 1 日に 6 回作成されるが，各スナップショットには 15 万から 20 万程度のノード間リンクの特性に関する情報が含まれる．

3 可視化

3.1 地図上への表示

本研究では，地図上でノードの位置を確認できるように，Google Maps を利用して，S3 の測定・推定データを表示する可視化ツールを構築した．本ツールは，以下の 3 つの特徴を持つ．

- ノード間の物理的な位置関係を確認することができる．
- 指定するノードを基準として測定・推定の結果を確認することができる．
- 多くの環境で利用可能である．

S3 での測定・推定に使用されるノードは世界中に広がっており，計測・推定データの理解には，各ノードの物理的な位置に関する情報が重要になる．本ツールでは，地図上でノードの位置を確認することができ，ノード間の物理的な距離を直感的に理解することが可能である．また，国境や海をまたぐリンクであるかなどの情報を容易に理解することもできる．

また，ユーザが指定するノードを基準として，計測・推定データを表の形で確認することもできる．この表は，任意のパラメータをキーとして並び替えることができるため，ユーザは簡単に良いリンクを抽出し地図上で確認することが可能である．

本ツールは JavaScript で実装されており，主要な環境のほとんどで実行可能であるといった利点もある．さらに，ツール本体となる Web ページや JavaScript のプログラム，可視化される測定・推定データは，Web サーバ

から取得することが可能であり，ユーザは Web ブラウザさえあれば，本ツールを使用することができる．

3.2 詳細な情報の表示

本研究では，前節で述べたものに加え，測定･推定されたデータをより詳細に解析，表示するための可視化ツールも構築した．本可視化ツールは，Smalltalk 環境の一つである Squeak 上で動作する．Squeak はプラグインにより Web ブラウザ上で動作させることができ，本ツールも主要な Web ブラウザが使用できる多くの環境で利用可能である．

インターネットのノード間リンクの測定・推定データは，ネットワーク管理者やインターネットアプリケーション開発者にとって有用であると考えられるが，大量にあるデータのうち，何を必要とするのかやそれらに対する見かたはそれぞれ異なる．本可視化ツールでは，データの表示方法をユーザが自由に決定できるようにすることで，多様な利用のされかたに対応するための柔軟性を実現している．すなわち，グラフや表などを表示の単位とし，ユーザがこれらを自由に可視化ツールのウィンドウ上にレイアウトできるようにしている．本ツールでは，このような表示の単位となるものをビューと呼んでおり，ビューには表 1 に示すような種類がある．

Histogram of all links では，すべての測定･推定データを対象に，ヒストグラムの形式で表示を行う．すべてのリンクが表示の対象となるため，どの程度の遅延時間のリンクが多いかなど，ネットワーク全体の状態を確認することが可能である．これに対して，Filtered link graph は，測定・推定されたリンク特性のうち，指定された条件を満たすものを表示するビューである．表示はグラフの形で行われるため，各リンクのパラメータの大小を容易に確認することができる．

ネットワーク全体を対象としたものではなく，特定のノードを基準にして，リンク特性の情報を表示することもできる．Link table は，測定･推定されたリンク特性のうち，指定されたノードをリンク元とするものを表の形で表示するビューである．遅延時間や帯域幅などをキーとしてソートすることができ，これによって，どのノードに対して良いリンクを持つかなどを簡単に確認することができる．

また，Spiral graph，Closest node set graph，Grid graph を用いると，リンク特性をグラフの形で確認することができる．Spiral graph は，指定されたノードから他のノードへのリンクを，良いものから順にらせん状にグラフ表示するビューである．Spiral graph を用いると，特定のノードが持つ良いリンクを抽出して確認することができる．Closest node set graph は，指定されたノードと，当該ノードに近いノードのいくつかをグラフ表示するビューである．本グラフでは，各ノード間の距離が各リンクの遅延時間や帯域幅をもとに計算され，この距離に応じてグラフ上に各ノードが配置される．ユーザは，Closest node set graph を用いることで，ノード間のネットワーク的な距離を容易に確認することができる．また，Grid graph は，指定されたノードを基準に，最も近いノードが隣接するようにグリッド上にノードを並べ表示するビューである．Grid graph では，ノードの配置が単純であるため，Closest node set graph に比べ，多くのノードを表示する場合に適している．

表 1　ビュー

View name	Description
System information	システムの情報を表示
Node information	ノードの静的な情報を表示
Histogram of all links	全リンクの測定・推定結果をヒストグラムで表示
Filtered link graph	指定された条件を満たすリンクを表示
Node list	測定・推定に使用されているノードの一覧
Link table	リンク特性のデータを表示
Spiral graph	良いリンクをらせん状に表示
Closest node set graph	良いリンクを持つノード間の関係を表示
Grid graph	最も良いリンクを持つノードを並べて表示
Histogram	リンク特性のデータをヒストグラムで表示
Node correlation	ノード間の相関関係を表示

4 おわりに

本稿では，広域ネットワークの解析を目的とした，ノード間リンクの特性情報を可視化する手法について述べた．インターネットのように，多くのノードが広範囲に広がるネットワークでは，ノードの位置も重要な情報になってくるが，本研究で構築した可視化ツールでは，ノードを地図上に表示できるため，ユーザは容易にノード間の位置関係を理解することができる．また，測定・推定されるデータは膨大な量であり，このデータに対してユーザごとに必要とするものは異なるが，本可視化ツールは，全データあるいは特定のノードを基準としたデータを，多様な形式でグラフ化することができるため，様々な可視化の要求に柔軟に対応することが可能である．

参考文献

[1] PlanetLab: http://www.planet-lab.org/.

[2] Yalagandula, P., Sharma, P., Banerjee, S., Basu, S. and Lee, S.-J.: S3: a scalable sensing service for monitoring large networked systems, *INM '06: Proceedings of the 2006 SIGCOMM workshop on Internet network management*, New York, NY, USA, ACM Press, pp. 71–76 (2006).

広域ネットワークの統計情報の解析および可視化

龍谷大学理工学部
芝 公仁
寺元 慎二

1

発表内容

1. 研究背景
   - ネットワーク特性の取得
2. 関連研究
3. 可視化するデータ
   - S3
4. 可視化・解析
5. まとめ

2

研究背景

- インターネット
  - 多数のサーバが様々なサービスを提供している
  - 同様のサービスを複数のホストで多地点から提供
- 適切なホストを選ぶことが重要
  - パケット伝送遅延時間の小さいホスト
  - ネットワーク帯域幅の大きいホスト

サーバの選択はクライアントが行えばよいが，
選択のための判断材料を提供する必要がある．

→ ネットワーク特性の計測

3

研究背景

- ネットワーク特性の取得
  - 計測には時間がかかるため，予め計測しておく
  - 常に変化しているため，それなりの頻度で
  - 測定自体がネットワークに負荷をかける
  - 少ない測定処理で正確に → 推定の技術で可能に
- 計測・推定されたデータの可視化・解析
  - 膨大なデータを理解
  - データの検証
  - 本来の目的である計測データの活用

4

関連研究

ネットワークディメンジョニング

- Pathchirp, Lighthouse, Netvigator
- ネットワーク特性を見積もる
- 多くがパケット伝送遅延を対象としている

可視化

- OpenView, Total Monitoring Environment
- 特定のネットワークの監視・管理
- リアルタイムにパケットを解析
- 特定リンクへの負荷, 特定のアプリケーション

5

可視化

- 広域ネットワークのノード間リンクの情報を可視化
  - 情報の解析
  - 結果を活用し, 本来の目的を達成する

想定する利用者

- ネットワークの管理者
- インターネットアプリケーションの開発者

可視化するデータ

- PlanetLab: インターネットの標本
- S3: ネットワークの測定・推定

6

S3

- S3 (Scalable Sensing Service)
  - PlanetLabのホスト上で動作
  - ノード間のリンクの特性を計測する
  - 計測結果をもとに，推定も行う

測定・推定データ

  - リアルタイムではない
  - 15万から20万リンクのデータがスナップショットとして出力される
  - 1日6回程度の測定・推定

7

S3

- パケット伝送遅延
  - measured_latency
  - nv_estimated_latency
- ネットワーク帯域幅
  - bottleneck_capacity
  - bottleneck_available_bandwidth_pathchirp
  - bottleneck_available_bandwidth_spruce
- パケットロス率
  - tulip_loss_rate

8

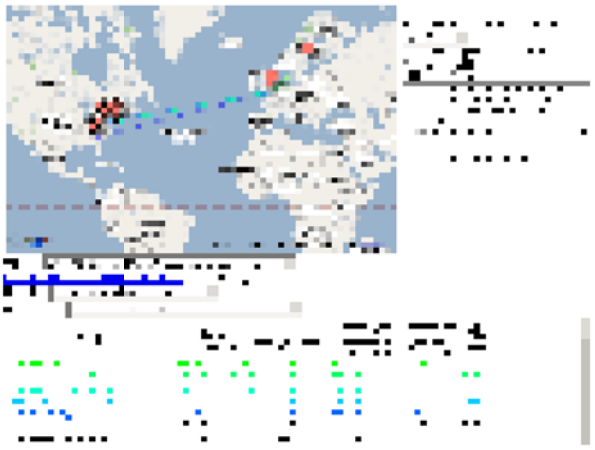
地図上への表示

- ノードの物理的な位置が重要になる
  - 地理的に近いサーバは有用
  - 経由するルータの数
- 良いリンクがどこにあるか
  - パケット伝送遅延
  - 帯域幅

9

地図上への表示

- Google Mapsを利用
  - JavaScript
  - Webブラウザさえあればよい
- 指定したホストに関するリンクを一覧できる
- リンクの特性を地図上で確認することが可能



有用なリンクを地図上で確認可能

10

より詳細な表示・解析

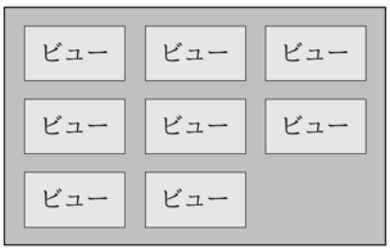
- 特徴
  - 特定のホストを基準とした可視化
    - 詳細な解析
  - 目的に応じた形式でグラフ化
    - 様々な用途に対応
  - 多くの環境で使用可能
    - Squeakによる実装
- ビューを単位とした表示
  - リスト，グラフ
  - 生成・削除，並び替え

11

より詳細な表示・解析

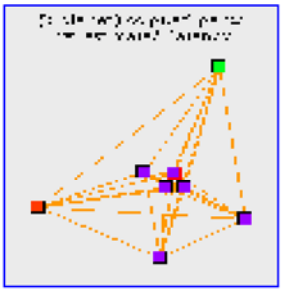
ウィンドウにビューを配置

- 自由にレイアウト



グラフ

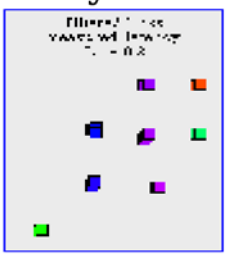
- ホストの物理的位置
  - 経度を色相として色で表す
- リンクを良さ
  - 線の長さ，太さで表す

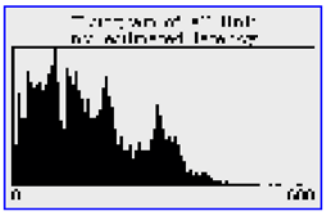


12

ビュー

- スナップショット全体を対象
  - Host list
  - Histogram
  - Filtered links
  - System information





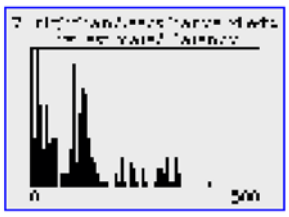
System information

Sites	361
Hosts	753
nodeinfo	283860
nodeinfo loader	completed
Repository URL	nil
Http proxy	127.0.0.1:8080
Layout server	127.0.0.1

13

ビュー

- 特定のホストを基準するビュー
  - Histogram
  - Correlation
  - Host information



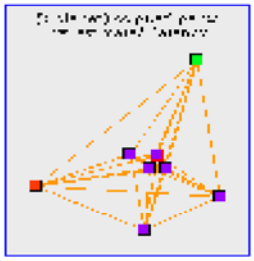
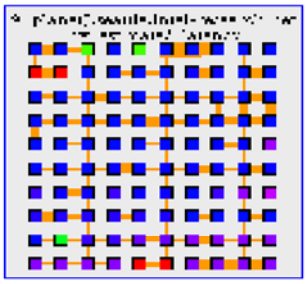
7: righthand.eecs.harvard.edu

Site	Harvard University
IP	140.247.60.126
Model	Dell Precision 340
Mac	00:08:74:40:3C:51
Boot CD	n
Version	2.0
Node ID	78

14

ビュー

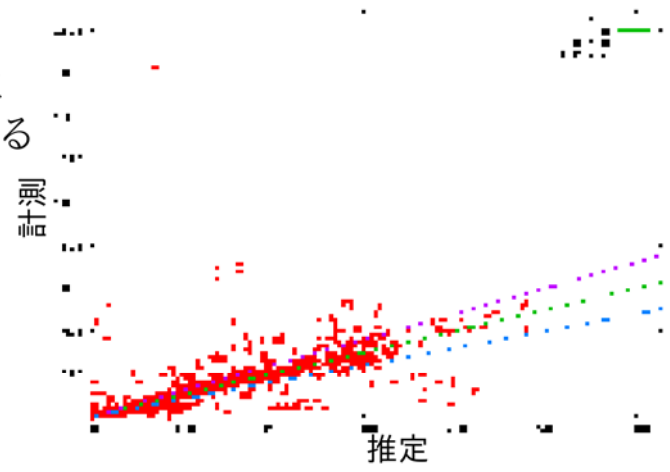
- 特定のホストを基準するビュー
  - Cloesest host set
  - Spiral graph
  - Grid graph


15

解析

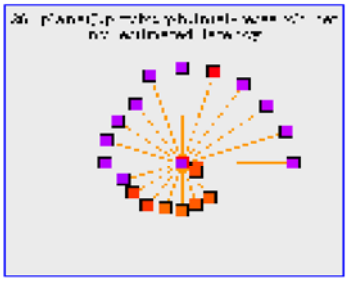
- データの検証
  - 推定の正確さ
  - 一部あきらかに不正なものもある



16

解析

- 物理的な位置
  - 経度との関連が強い
- 相関
  - 物理的な位置によらず，多くのホストが正の相関関係を有する



17

まとめ

- 広域ネットワークでのノード間リンクの特性を可視化
  - ネットワーク上の資源を効率的に利用するために重要な情報
  - 可視化によって，実態の理解を助ける
  - 時間による変化に関しても考慮する必要がある

18

P2P データポット: センサネットワーク向け分散型マイクロストレージアーキテクチャ

藤崎 友樹
立命館大学大学院理工学研究科

1 はじめに

近年，センサ技術の発展により，無線によるセンサネットワークの構築が可能となり，さまざまな環境にセンサを適用することが可能となった．現在，環境観測を行うセンサアプリケーションは，アプリケーションとセンサネットワークから構成されているものが主流である．アプリケーションは，一般的な PC 上で動作し，センサネットワークと 1 対 1 に対応することでセンシングの要求と結果の蓄積を行っている．センサネットワーク上に存在するノードは，それぞれ無線通信機能を持ち，マルチホップ通信によって長距離の通信を行う．ノードは，それぞれが必要な場合にのみ動作するよう，効率的なスケジューリングを行うことで電力消費を最低限に抑え，限られた電力資源の中で長期にわたる動作が可能なよう設計されている．

これらのシステムにおいて，広範囲にわたる大規模な環境観測を行おうとした場合，その規模に従ってマルチホップ通信の中継による消費電力が増大し，ノードの生存期間が短縮されてしまうという問題が発生する．また，高精度での環境観測を行う場合，センサノードを稠密に配置する必要があるため，導入や維持に掛かるコストが大きく，観測対象が移動や増減する場合など動的な変化する環境での利用が困難であり，拡張性や柔軟性に乏しいといった問題があった．

これらの問題に対し，基地局およびストレージを小型のデバイスとして移動可能にすることで，ノードと基地局の距離を短縮し，中継回数を減少させることができると考えられる．また，基地局の固定配置が不要となることで，従来より柔軟なシステム構成が可能となり，センサネットワークの適用可能な環境を拡大することができると考えられる．

以上の背景から，現在，我々は，移動体によるセンシングの基盤技術に関する研究を行っている．本システムでは，センサネットワークにおけるストレージに柔軟性を持たせ，センサネットワークの利用をより容易にし，移動体センシング，広範囲のセンシング，複数に分散した環境のセンシングといった，従来より高度な利用を可能とする．また，移動するストレージが相互にアドホック通信を行い，自律的にデータの共有や複製を行うことで，より柔軟かつ容易なデータ利用を実現し，自律的にネットワークを構成することで，アプリケーションからの問い合わせに対する位置透過的性を実現する．

本研究におけるセンサネットワークシステムは，3 層によって構成される．アプリケーションとセンサネットワークの間に無線アドホック通信を用いた P2P ネットワークを構築することで，柔軟なシステムの構成を可能し，事前にインフラや設定を必要をしないアドホック性，動的な環境に対応可能な柔軟性，観測対象の位置を問わず統一的な操作が可能な位置透過性を実現する．

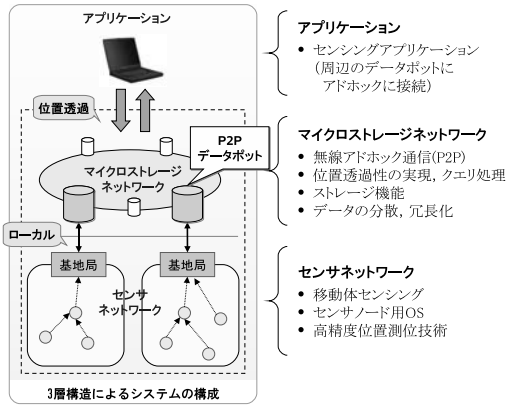


図 1　3 層構造によるシステムの全体構成

2 P2P データポット

2.1 3 層構造によるシステムの構成

本システムの全体構成を図 1 に示す．本研究で提案するプラットフォームは，アプリケーション，マイクロストレージネットワーク，センサネットワークの 3 層によって構成される．

本プラットフォームは，従来のセンサアプリケーションにおけるアプリケーションとセンサーネットワークの間に，複数のセンサーネットワークを取りまとめる新たな層（マイクロストレージネットワーク）を構成する．マイクロストレージネットワークは，アドホックに接続された複数のノードによって構成され，ネットワーク上に存在する個々のノードを P2P データポットと呼ぶ．

2.2 P2P データポットの概要

P2P データポットは，センサネットワークとの通信機能，ストレージ機能，無線通信機能とアプリケーションに対するゲートウェイ機構を備え，それぞれが無線アドホック通信により自律的に P2P ネットワークを構築し，複数のセンサーネットワークにおけるデータフローを適切に管理することで，柔軟性と位置透過性を実現する．

P2P データポットは，対応するセンサネットワークのストレージとしての機能を持ち，センシングによって得られた環境情報の蓄積を行う．また，センサネットワークのノード情報や実行中のクエリの管理といったセンサネットワークの管理を行う．

さらに，それぞれの P2P データポットは，アプリケーションに対するゲートウェイ機能を持ち，統一的なイン

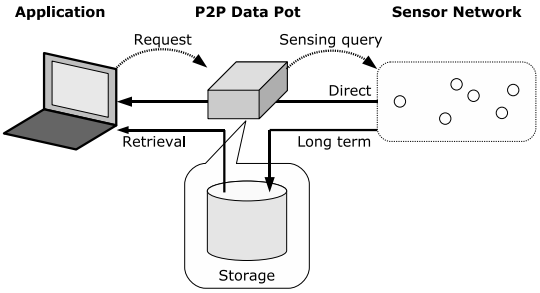


図 2　問い合わせの種類ごとのセンシング結果の流れ

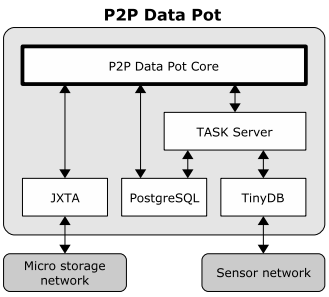


図 3　P2P データポットのソフトウェア構成

タフェースを提供する．これによりネットワーク上に存在するどの P2P データポットからでも要求の発行とその結果を得ることができる．P2P データポット同士の通信と同様，アプリケーションも事前の設定を必要とせず P2P データポットとアドホックに通信が可能である．

P2P データポットは，実装の簡略化のため P2P ネットワークの構成や探索の機構自体は既存の実装を利用して構成する．各種の問合せや情報は，P2P ネットワーク上にアドバタイズメントとして告知して伝達する．

2.3　問合せの分類

本システムにおいては，アプリケーションからの要求を長期クエリ，集計クエリ，直接クエリ，管理クエリの 4 種類のクエリとして定義する．これらの要求の流れを図 2 に示す．長期クエリは，数日から数ヶ月といった長期的なセンシング要求であり，その結果はストレージに蓄積される．集計クエリは，長期クエリによって得られた結果を回収するための要求である．直接クエリは，現在の状態を連続的に観測するためのセンシング要求であり，その結果は蓄積されない．管理クエリは，センシング状態やネットワーク状態の監視を行うためのシステム全体に対する要求である．

2.4　メタデータの管理

P2P データポットでは，クエリ操作において必要なデータポットおよびセンサーノードの特定，クエリの一貫性の確保，位置透過性の実現のため，それぞれに必要な情報をメタデータとして管理する．各 P2P データポットは，アドホック通信によって確認したネットワーク上の他の P2P データポットの情報，自身が管理しているセンサーネットワーク上に存在するセンサーノードの情報，各データポット上で実行されているクエリおよび蓄積されたデータに関する情報を保持する．

3　実装

3.1　ハードウェア構成

これまでに述べた P2P データポットの設計を基に，P2P データポットのプロトタイプ実装を行っている．P2P データポットは， SH4 を搭載した Linux Box での動作を予定しているが，プロトタイプ実装においては，無線 LAN を搭載した PC を利用して実装を行っている．また，センサネットワークを構成するキットとして Crossbow 社の Mote-Kit2400J を用いており，センサ基地局である MIB510 とシリアル接続を行い実験を行っている．

3.2　ソフトウェア構成

P2P データポットのソフトウェアは， Linux 上で実装され，実装を簡潔に行うため既存のソフトウェアを利用している．ソフトウェアの構成を図 3 に示す．現在の実装では， OS に Vine Linux 4.1 (kernel-2.6.16) と IPv6 関連のソフトウェアを導入し，ストレージに使用するデータベースとして PostgreSQL 8.1.5, センサネットワークに TinyOS 1.1.15 および同梱の TinyDB [1] パッケージ, TASK [2] (Tiny Application Sensor Kit) サーバを使用している．また， P2P ネットワークの構成には Java 版の JXTA [3] 実装 (jxta-j2se) 2.5 RC1 を使用している．これらのソフトウェアを利用し，P2P データポットの機能を実装するのが P2P Data Pot Core である．

無線 LAN によるアドホックネットワークは， IPv6 におけるリンクローカルアドレスを用い，事前のアドレス設定を不要にしている．また， JXTA ネットワークにおいても IPv6 のリンクローカルマルチキャストアドレスを使用し，アドバタイズメントの伝播を行っている．

4　おわりに

本稿では，本研究で提案する 3 層構造でのプラットフォームと P2P データポットについて述べ，現在の実装状況と今後の実装予定について述べた．今後も実装を継続し，基本的な動作を行う実装の完成しだい平行して実験を行っていく予定である．これらの実装を完了したのち，評価を行い修士論文とする予定である．

参考文献

[1] Sam Madden, Michael J. Franklin, Joseph M. Hellerstein and Wei Hong: TinyDB: An Acqusitional Query Processing System for Sensor Networks, ACM TODS, 2005.

[2] Buonadonna, P. and Gay, D. and Hellerstein, J.M. and Hong, W. and Madden, S.: TASK: Sensor Network in a Box, Proceedings of the Second IEEE European Workshop on Wireless Sensor Networks and Applications, 2005.

[3] Sun Microsytems, Inc.: Project JXTA. http://www.sun.com/software/jxta/

P2Pデータポット

センサネットワーク向け分散型
マイクロストレージアーキテクチャ

藤崎友樹

立命館大学大学院 理工学研究科
大久保・横田研究室
fujisaki@sol.cs.ritsumei.ac.jp

発表内容

- はじめに
  - 研究背景
  - 移動体ストレージ
- P2Pデータポット
  - 機能・設計
- 実装
  - ソフトウェア，ハードウェアの構成
  - 現在の進捗，今後の予定
- おわりに

はじめに

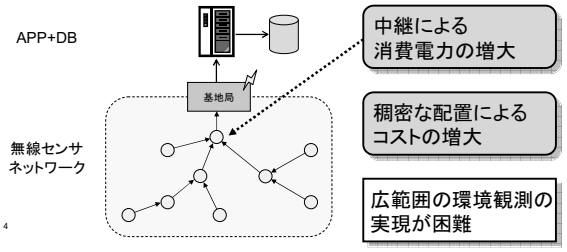
- センサー技術の発展
  - 無線によるセンサネットワークの構築

⇩

  - さまざまな環境でのセンサアプリケーションの利用
    - 斜面観測による防災システム：
      広範囲・長期の環境観測とセンシングデータの蓄積
    - 工場機械の状態監視：
      リアルタイムにデータを収集・管理

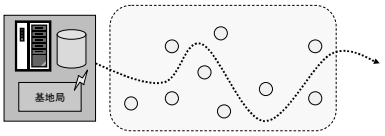
一般的なセンサネットワーク

- 無線通信機能を持つセンサノードがアドホック通信
  - 効率的なスケジューリングで限られた電力資源を効率的に利用
- センシングしたデータをストレージへ蓄積



ストレージを移動

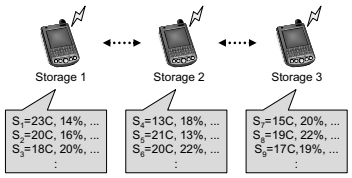
- ◆ 問題点: 基地局・ストレージが固定されている
  - ➢ PDA等の小型デバイスとして移動できれば
    - ✓ ノード間通信，中継回数の減少
    - ✓ 固定配置が不要でより柔軟なシステム構成が可能



5 APP+DB 無線センサネットワーク

ストレージ同士が通信することで…

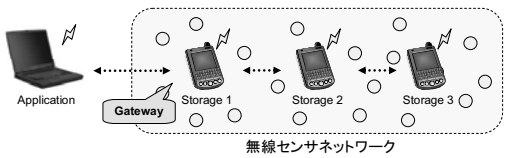
- ✓ ストレージ間でデータの共有・複製
  - ● 無線LANを用いたアドホック通信
  - ● 問い合わせのための統一的なインタフェース
    - ➢ データの利用を柔軟かつ容易に



6

マイクロストレージネットワーク

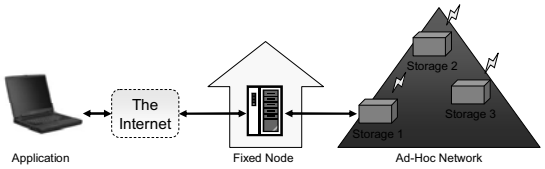
- ● ストレージ間で自律的にネットワークを構成
- ● 問い合わせを位置透過的に処理
  - ◆ 仮想的な1つのセンサネットワークとして動作
  - ✓ 事前に整備されたインフラが不要
  - ✓ 観測対象の移動・増減などの動的な環境変更に強い



7 無線センサネットワーク

移動体と静的環境の組合わせ

- ● ネットワークインフラが整備可能な環境との組合わせを可能にすることで…
  - ● インターネット経由での遠隔地からの監視
  - ● 複数の環境を統一的な手法で管理



8 Application Fixed Node Ad-Hoc Network

研究目的

- センサネットワークの
  より柔軟で高度な利用を可能とする
  - センサネットワークの利用をより容易に
    - 事前にインフラや設定を必要としないアドホック性
    - イニシャルコスト・ランニングコストの削減
  - 移動体のセンシングや
    広範囲・複数に分散した環境のセンシング
    - 動的な変化のある環境にも対応可能な柔軟性
    - ノードの位置を問わず統一的な操作が可能な位置透過性

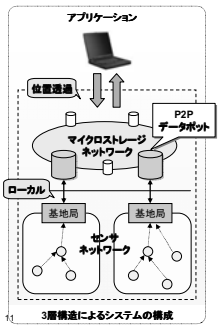
9

利用例

- 山の斜面の長期的な観測
  - ネットワークインフラの構築が難しい環境での観測
    (有線では落雷によって破損しやすい)
  - 無線での長期に渡った観測、遠隔地からの管理
- デパート内の環境監視
  - 頻繁に環境が変化する環境での観測
  - 低コストで柔軟な構成が可能

10

システム全体の構成



3層構造によるシステムの構成

アプリケーション
- センシングアプリケーション
  （周辺のデータポットにアドホックに接続）

マイクロストレージネットワーク
- 無線アドホック通信(P2P)
- 位置透過性の実現，クエリ処理
- センシングデータの蓄積，分散と冗長化

センサネットワーク
- 移動体センシング(パーソナルセンシング)
- センサノード用OS
- 高精度位置測位技術

11

P2Pデータポット

= ストレージネットワークを構成する各ノード
- センサネットワークのデータを蓄積・インデックス化
- 無線LANによるアドホック通信によって
  自律的にP2Pネットワークを構成
- アプリケーションに対するゲートウェイ

➢ 複数のP2Pデータポットが協調動作
- 複数のセンサネットワークを抽象化し，ゲートウェイに
  よって統一的なアクセス機構を提供する

12

P2Pデータポットが提供する機能

- P2Pデータポット - センサネットワーク間
  - ✓ それぞれのデータポットがストレージを持ち，センシングによって得られたデータを蓄積（データベース，メタデータ管理）
  - ✓ センサネットワーク上のノード情報を把握（目的ノードの探索）
- P2Pデータポット - アプリケーション間
  - ✓ 相互に事前設定なくアドホックに通信可能（アドホック通信）
  - ✓ いずれかのデータポットへ問合せるだけで，目的のデータを得ることが可能（抽象化，目的データの探索，ゲートウェイ機構）

13

P2Pデータポットの設計

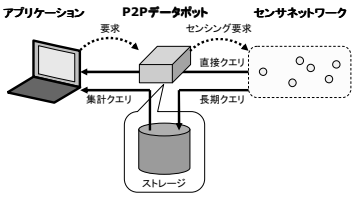
- P2Pネットワークの構成やストレージ: 既存の実装を利用
- 電源資源は，ある程度潤沢に利用可能であることを想定

- 問合せの分類
  - アプリケーションから発行される問合せの分類
- メタデータの管理
  - データ探索・管理に必要なデータ
- 位置透過性の実現
  - 対象とするデータおよびセンサノードの探索機構

14

問合せの分類

- 主な要求を4種のクエリで実現
  - 長期クエリ: センシング値のストレージへの継続的な蓄積の指示（長期の環境観測など）
  - 集計クエリ: 長期クエリによる結果の集計・回収
  - 直接クエリ: センシング値の直接取得（蓄積しない）
  - 管理クエリ: ネットワークの状態取得および管理



15

メタデータの管理

- 各P2Pデータポットがそれぞれ保持
- 問い合わせ対象となるデータポットやセンサノードの検索，ネットワークの管理に使用
  - データポットテーブル:
    自身および他のデータポットに関するメタ情報.
    （データポットのID，位置情報，管理センサノード数など）
  - センサノードテーブル:
    自身が管理するセンサネットワークに存在するセンサノード群の情報．（ノードID，位置情報，搭載センサ種別など）
  - クエリテーブル:
    自身がこれまでに実行・蓄積したクエリに関するメタ情報.
    （クエリのユニークID，ピアグループID，クエリの内容，状態など）

16

位置透過性の実現

- P2Pネットワークの構成と探索
  - 構成や探索を行う機構自体は，既存の実装を使用
  - 問合せや生存情報は，アドバタイズメントとして ネットワーク上に告知
  - メタデータ管理によるキャッシュ
- アプリケーションに対するゲートウェイ機構
  - ノードやデータの探索および問合せを代理
  - 問合せは，ローカル・外部を区別せずに記述可能 （ノードIDや位置情報から対象を指定）

17

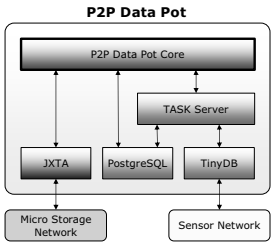
実装

- 現在実装を行っている
  - ハードウェア:
    - 無線LANを搭載したPC複数台 （最終的にはSH4搭載のLinux Boxへ移植予定）
    - Crossbow Mote-Kit2400J （センサーキット）
  - ソフトウェア: 既存のソフトウェアを利用
    - OS: Linux (Vine 4.1, 2.6系kernel, IPv6サポート)
    - RDBMS: PostgreSQL
    - センサネットワーク: TinyOS + TinyDB
    - P2Pネットワーク: JXTA

18

ソフトウェアの構成

- 環境情報の収集 (センサネットワーク, ストレージ)
  - TASK Server (TinyDB)
  - PostgreSQL
- ネットワーク基盤の構成 (P2Pネットワーク)
  - JXTA
  - IPv6によるアドレス構成
- マイクロストレージネットワークの構成 (ゲートウェイ，検索・配信処理)
  - P2P Data Pot Core (実装)



19

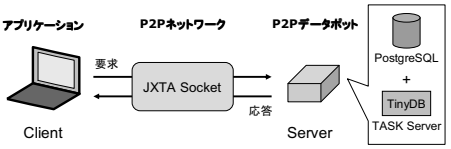
無線LANアドホックネットワーク

- アドホックネットワークの構成
  - 無線LAN (IEEE802.11b) アドホックモード
  - IPv6のリンクローカルアドレスを使用
    - アドレス設定不要で相互通信が可能
  - JXTA + IPv6
    - IPv6 リンクローカル マルチキャスト アドレスを使用し，JXTA アドバタイズメントの送受信が可能

20

ソフトウェアの実装

- 現在までに次の手順で実装を行った
  1. アドホックモードの無線LAN上でIPv6を利用し リンクローカルアドレスの構成
  2. IPv6とJXTAを利用したP2Pネットワークの構成
  3. 特定のノードに対しSQLクエリを発行し，結果を返す (JDBC経由でPostgreSQLを使用)
  4. 特定のノードに対しセンシング要求を発行し，結果を返す (TinyDBライブラリを使用)



21

今後のソフトウェア実装予定

- 以下の手順で実装を予定
  5. ノード探索の具体的な条件と探索手法の検討
  6. 対象ノードの探索機構の実装
  7. 探索によって発見したノード情報のキャッシング
  8. メタデータの管理機構の実装
  9. 複数のデータポットを対象にしたクエリの発行
  10. ゲートウェイ機構の実装とクエリ・インタフェースの定義

22

おわりに

- センサネットワークの課題
- 移動ストレージ
- P2Pデータポットについて
- 実装の現状と今後の予定

- 今後: 実装を継続
  - ある程度動作可能になり次第，平行して実験も予定
  - 障害対策の検討

23

予測に基づく動的負荷分散機能を持つクラスタファイルシステム

藤岡　聡[†]

立命館大学大学院　理工学研究科[†]

概要

近年，大規模分散環境として複数の計算機をネットワークで接続して構成されるクラスタシステムや広域に分散した資源を扱うグリッドシステムが注目を集めている[1].

これらクラスタ・グリッド環境で大量のデータの生成と参照を行うアプリケーションには，大容量で高速なファイルアクセス性能が求められる．クラスタ・グリッド環境ではネットワーク越しにファイルアクセスが発生するが，NFSやCoda[2] など従来の分散ファイルシステムでは，単一のファイルサーバにファイルアクセスが集中しネットワークバンド幅が不足するため，十分なファイルアクセス性能を得ることができない．一般にクラスタ・グリッド環境におけるファイル共有には，Lustre[3] やPVFS[4]などに代表されるファイルサーバそのもののクラスタリングであるクラスタファイルシステムが用いられる．クラスタファイルシステムは，透過的なファイルアクセスを維持しながらファイルもしくはその断片を複数のファイルサーバ上へ分散配置する．それによりファイルアクセスの集中を回避し，ネットワークバンド幅を効率的に利用できる．その結果，高いファイルアクセス性能を発揮する分散ファイルシステムが実現できる.

しかしながら，現在のクラスタファイルシステムの多くは，初回の書き込みに対してのみ負荷分散を行っており，それ以降はシステムの負荷状況が変動しても対処しない．その結果，特定のファイルサーバへのファイルアクセスの集中が発生する場合にファイルアクセス性能が低下する．この状況を回避し安定したファイルアクセス性能を維持するためには，ユーザ自身による動的なチューニングを行う必要がある．このチューニングには，ファイルシステム上の負荷状況の把握とそのファイルシステムに精通していることが求められるため，多くのユーザにとっては困難である．この問題の解決には，ファイルシステムによる負荷状況の把握と動的負荷分散技術が不可欠である.

そこで本発表では，大量のデータの生成と参照を行うアプリケーションを対象に，クラスタファイルシステムのファイルアクセスの効率化を目的とした，予測に基づく動的負荷分散機能を提案する．本手法では，負荷予測可能なモニタリングソフトウェアNWS[5]を用いて負荷状況を事前に検出する．その際，負荷状況であるファイルサーバにおいて，原因となるファイルを他のファイルサーバへ複製または移動する．これにより，ファイルアクセスの効率化を実現する．また本発表では，動的負荷分散時に用いるアルゴリズムについても検討する.

参考文献

[1] I. Foster and C. Kesselman, “The GRID Blueprint for a New Computing Infrastructure”, Morgan Kaufmann Publishers, 1998.

[2] Coda File System, http://www.coda.cs.cmu.edu/.

[3] Lustre, http://www.lustre.org/.

[4] P. H. Carns and W. B. Ligon III and R. B. Ross and R. Thakur, “PVFS: A Parallel File System For Linux Clusters”, In Proceedings of the 4th Annual Linux Showcase and Conference, pages 317-327, Atlanta, GA, October 2000.

[5] R. Wolski and N. T. Spring and J. Hayes, “The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing”, Journal of Future Generation Computing Systems, Vol. 15, No. 5-6, pp. 757-768, October 1999.

Prediction-Based Dynamic Load Balancing in a Cluster File System

† Satoshi Fujioka: Graduate School of Science and Engineering, Ritsumeikan University

予測に基づく動的負荷分散機能を持つ クラスタファイルシステム

立命館大学大学院理工学研究科
國枝・桑原研究室
藤岡　聡
rs038027@se.ritsumei.ac.jp

発表内容

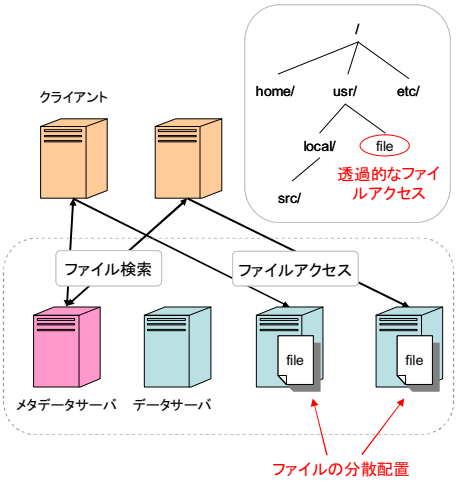
- 研究背景
- 研究概要
- 予測に基づく動的負荷分散
- 動的負荷分散アルゴリズム
- まとめと今後の課題

研究背景

- クラスタ・グリッド環境におけるファイル共有
  - 要求
    - 大容量で高速なファイルアクセス性能が必要
      ※ 動画像処理やゲノム解析などの大規模計算など
  - 現状
    - 従来の分散ファイルシステムではネットワークバンド幅が不足
      - 複数のノードから単一のファイルサーバへファイルアクセスが集中
      ※ NFS, Codaなど
    - クラスタファイルシステムの利用
      - ファイルサーバのクラスタリングによるファイルアクセスの分散
      ※ Lustre, PVFSなど

クラスタファイルシステムにおける負荷分散

- 手法
  - 複数台のファイルサーバに
    ファイルを分散配置
  → ネットワークバンド幅を効率的に利用
- システム構成
  - クライアント
    - ファイルの読み書き
  - データサーバ
    - ファイルの格納
  - メタデータサーバ
    - ディレクトリサービスの提供



従来の負荷分散における問題点

- 各データサーバの負荷状況は変動
  - データサーバへのファイルアクセスの集中が発生
  - 初回書き込み時の負荷分散では不十分
- 結果的にユーザ自身によるチューニングが必要
  - ユーザは負荷状況を把握し，負荷分散を行う
  - しかし，多くのユーザにとっては技術的側面から困難

→ ファイルシステムによる負荷状況の把握と動的負荷分散が不可欠

関連研究

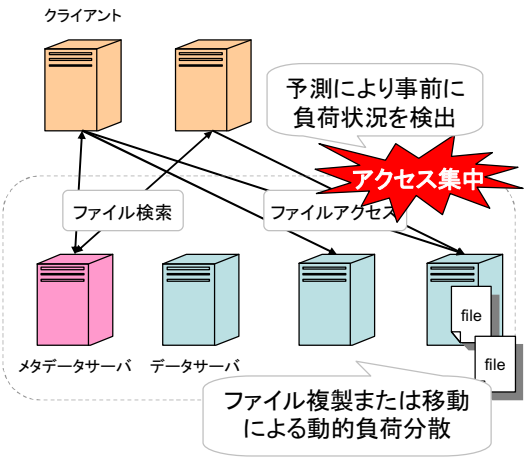
- アプリケーション特性に基づく複製ファイルの配置
  - アプリケーションのファイルアクセスパターンをユーザが予め把握
  - その情報に基づき最適な複製ファイルの配置を行う
  - →予めアクセスパターンが把握できないアプリケーションが存在
  - →クラスタ・グリッド環境が取り扱うアプリケーションは多岐にわたる
- 負荷状況に応じた複製ファイルの配置
  - システムモニタリング情報に基づき動的に複製ファイルの配置を行う
  - →ファイルシステムによる汎用的な動的負荷分散が可能
  - →動的負荷分散は，可能な限り負荷状況が発生する前に行いたい

研究概要

- 目的
  - クラスタファイルシステムにおけるファイルアクセスの効率化
- 手段
  - クラスタファイルシステムにおける動的負荷分散技術の強化
- 手法
  - ファイルアクセスの集中を事前に予測し，ファイル複製または移動によりこれを回避する

予測に基づく動的負荷分散

- データサーバへのファイルアクセスの集中を事前に検出
  - 各データサーバの負荷予測情報
  - メタデータサーバが持つファイルアクセス情報
- ファイル複製または移動による動的負荷分散
  - 同一ファイルへのアクセス
    - ファイル複製の利用
  - 異なるファイルへのアクセス
    - ファイル移動の利用



システムモニタリング要求

- 各種リソースのモニタリングが可能であること
  - スペック, CPU利用率, ネットワーク負荷など
- 高精度な負荷状況の予測が可能であること
  - 正確に負荷状況の予測を行いたい
- 低負荷であること
  - モニタリングコストをできるだけ抑えたい
- 拡張性を持つこと
  - 台数が増加した場合でも低負荷を維持したい

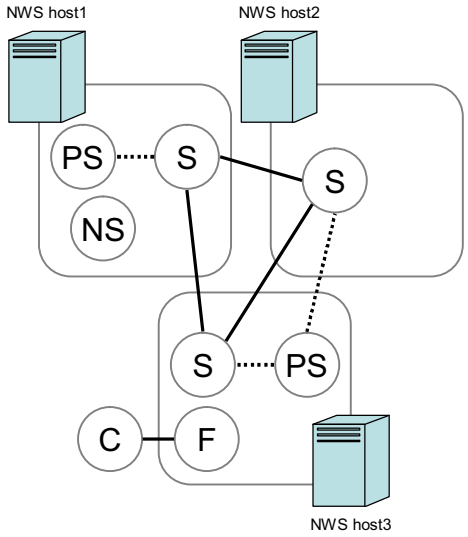
NWS

- NWS (Network Weather Service)
  - @UC Santa Barbara
  - クラスタ・グリッド向けのモニタリングソフトウェア
  - 特徴
    - 各種リソースのモニタリングとその予測が可能
      - 測定値の統計により予測値を生成
      - 予測精度は測定頻度に依存
    - 対象のグループ化, 階層化による低負荷なモニタリング
      - 重複する測定を削減
        - » 同一箇所を経由するネットワーク測定など

  → システムモニタリング要求を満たす

システム構成

- NS: Name Server
  - 各NWSホストへディレクトリサービスを提供する
- S: Sensor
  - 指定された資源の測定を行い, 結果をPersistent Stateに格納する
- PS: Persistent State
  - 各NWSホストの測定結果を格納する
- F: Forecaster
  - リソースの予測値を生成し, クライアントに提供する
- C: Client
  - 予測値をForecasterに要求する



予測機構

- 予測機構は以下のカテゴリに分類
  - mean-based methods
    - 測定値履歴の算術平均
  - median-based methods
    - 測定値履歴の中央値
  - autoregressive methods
    - 時刻 t+1 の予測値は, 時刻 t の測定値に対して自己回帰モデルを適応

予測値の生成

- ForecasterはSensorからの測定値を得たのち, 全ての予測機構で予測を行い, 予測値を生成
  - 測定時刻 t におけるそれぞれの予測機構 f での式

$$prediction_f(t) = METHOD_f(value(t), history_f(t))$$

  - 変数の意味

$\overbrace{value(t)}$: 時刻 t での測定値

$prediction_f(t)$: 予測機構 f よって生成された時刻(t+1)の予測値

$\overbrace{history_f(t)}$: 時刻 t までの測定値と予測機構 f による予測値その誤差

$METHOD_f$: 予測機構 f

予測機構の選択

- Forecasterは,予測機構の選択に以下の指標を用いる

$$MSE_f(t) = \frac{1}{t+1}\sum_{i=0}^{t}(err_f(i))^2$$: 予測誤差の二乗の平均

$$MAE_f(t) = \frac{1}{t+1}\sum_{i=0}^{t}\left|\frac{err_f(i)}{value(i)}\right|$$: 予測誤差の比率の平均

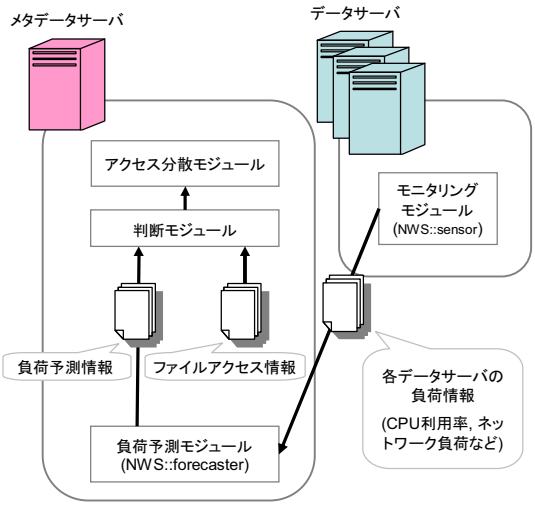
※ $err_f(t) = value(t) - prediction_f(t-1)$: 予測誤差

- 予測機構の選択は, 以下の条件により行う
  - 時刻 t における予測機構 f の $MSE_f(t)$ が最小となるものを選択
  - 時刻 t における予測機構 f の $MAE_f(t)$ が最小となるものを選択

システム内部構造

[データサーバ側]
- モニタリングモジュール
  - 定期的に各種リソースの負荷状況をモニタリングする (NWS::sensor機能)

[メタデータサーバ側]
- 負荷予測モジュール
  - モニタリングモジュールから得られた情報から負荷予測を行う (NWS::forecaster機能)
- 判断モジュール
  - 負荷予測情報とファイルアクセス情報からファイルの複製または移動の判断を行う
  - どのファイルをどのデータサーバへ複製または移動を行うか決定する
- アクセス分散モジュール
  - 各データサーバへファイルの複製または移動を行う



動的負荷分散アルゴリズム(1/3)

- 方針
  - データサーバの負荷を均一化することで安定したファイルアクセス性能を得る
- 用いるパラメータ

$V_{max\_file\_trans}$: 最大ファイル転送速度
$L_{prediction}$: CPU利用率の予測
$Connection_{prediction}$: コネクション数の予測

データサーバ における負荷状況の予測を以下の評価式で表す

$$Performance_{prediction} = \frac{V_{max\_file\_trans} * L_{prediction}}{Connection_{prediction} + 1}$$

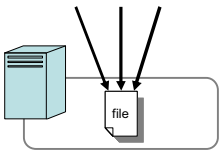
平均ファイル転送速度の予測

動的負荷分散アルゴリズム(2/3)

- 動的負荷分散アルゴリズムが呼ばれる頻度
  - NWSによる負荷状況の予測頻度に準ずる
- 動的負荷分散アルゴリズム
  - 全データサーバの中で負荷予測値が最大となるデータサーバを選択
  - そのデータサーバの負荷予測値が基準値を超える場合,
    - 基準値 = システムで許容可能な負荷予測値 + α
  - 以下の条件に分けて負荷分散を行う
    - 同一ファイルへのアクセスが集中する場合
    - 異なるファイルへのアクセスが集中する場合
  - これを基準値を超えるデータサーバが無くなるまで繰り返す

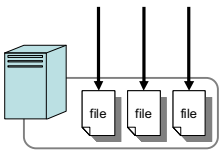
動的負荷分散アルゴリズム(3/3)

- 同一ファイルへのアクセスが集中する場合
  - データサーバ上で最もアクセス数の多いファイルを選択し,
  - 負荷予測値が最小であるデータサーバに対して,
  - 複製を行う



同一ファイルへの
アクセス集中

- 異なるファイルへのアクセスが集中する場合
  - データサーバ上でアクセス数が一定数を超えるファイルのひとつを選択し,
  - 負荷予測値が最小であるデータサーバに対して,
  - 移動を行う
  - 一定数を超えるファイルが無くなるまで繰り返す



異なるファイルへの
アクセス集中

検討事項

- 予測頻度
  - 予測によるコストが動的負荷分散による利得を上回らないよう設定が必要
    - 固定値
    - 負荷状況に応じた変動値
- 動的負荷分散アルゴリズムの停止性
  - 永遠に動的負荷分散を行わないよう対処
    - 負荷状況の判断に用いる基準値に過去に行った動的負荷分散の頻度を考慮
- 予測失敗時のペナルティ
  - ファイルアクセスが安定していると予測した場合
    - 実際にファイルアクセスの集中が発生
    - しかし, 次回予測時の動的負荷分散で対処可能
  - ファイルアクセスが集中していると予測した場合
    - 動的負荷分散を行うため, 他のノードに対して負荷が余分に発生
  - 負荷状況の判断に用いる基準値に過去の予測精度を考慮

まとめと今後の課題

- クラスタファイルシステムにおける動的負荷分散
  - 予測に基づきファイルアクセスの集中を事前に検出
  - アクセス集中時のファイル複製または移動
- 動的負荷分散アルゴリズム
- 今後の課題
  - 分散環境シミュレータSimGrid上での本手法のモデル化
  - 既存のクラスタファイルシステム上にプロトタイプを実装
  - 動的負荷分散アルゴリズムの評価と改良

オンラインストレージを用いた分散仮想ディスクの開発

野尻祐亮 †

† 東京農工大学 大学院 博士前期課程 工学府

1 はじめに

個人データや OS のデスクトップ環境を，自宅や外出先などで、場所を問わず利用したいという要求がある．本研究ではそれに応えるため，高い機能性や利便性を持ち，インターネット経由でアクセス可能な仮想ディスクシステムを開発する．

概念図を図 1 に示す。本システムでは，ユーザの視点から，通信に用いるプロトコルの種類やハードウェアの物理的な構成を隠蔽し，単なるストレージデバイスのように見せかける．ユーザにとっては，あたかもインターネット上に存在するストレージデバイスを直接読書きしているかのような利用形態を提供する．

2 個人データの持運びにおける現状の問題点

インターネット経由で個人データを持運ぶ方式はいくつか存在するものの、必ずしも使いやすいものではない。iSCSI (Internet Small Computer System Interface) は、個人ユーザが導入するにはコストが高い．一般に存在するオンラインのストレージサービスは，ユーザインタフェースが統一されていなかったり、透過性の面で不利であったりする。インターネット経由でブート可能な Linux ディストリビューションである HTTP-FUSE-KNOPPIX [1] では、システムファイル取得のため仮想的なブロックデバイスを HTTP-FUSE-cloop ドライバを用いて実現する。ただし、この仕組みはデータの読込みに特化しているため、ユーザデータの保存には応用できない。

一方、個人データを USB メモリなどの物理メディアで持ち運ぶ場合は，紛失・盗難などの危険がある．

3 設計

本開発では、ファイルシステムのドライバではなく、ブロックデバイスのドライバを作成する。ブロックデバイスとすることでユーザは、通常の物理的なディスクデバイスとの区別なく、透過的に扱うことができるようになる。そして、ext3 のようなファイルシステム

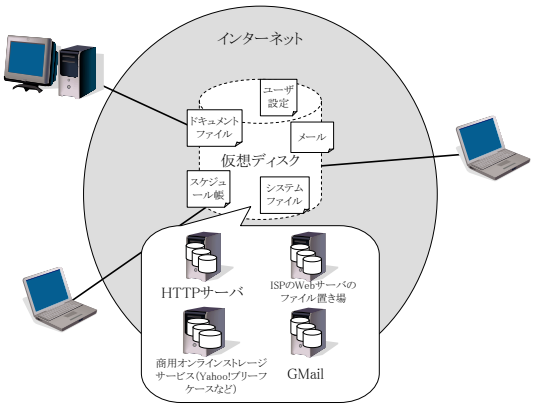


図 1: 概念図

や、ソフトウェア RAID といった、ストレージデバイスのための既存の枠組みをそのまま適用することができ、パフォーマンスや信頼性の向上を行える．

クライアントマシン上で動作させるドライバの構成を図 2 に示す。ドライバは、カーネルモード部分とユーザモード部分の 2 つの部分からなる。実際のネットワーク入出力処理などをユーザモード行うことで、プログラミングが容易となる・各種プロトコルごとにモジュール化できる・拡張が容易となるという利点がある。

カーネルモード・ユーザモードの各部分について述べる。

カーネルモジュール vdisk.ko カーネルモジュール vdisk.ko は、カーネル側に向けたインタフェースとしてブロックデバイス /dev/vdisk$n$（n はマイナー番号)、ユーザモード側に向けたインタフェースとしてキャラクタデバイス /dev/vdiskio を持つ。カーネルモジュールの役割は、ブロックデバイスを通してカーネルから受け取ったブロック I/O リクエストを、独自定義の I/O パケットに変換し、キャラクタデバイスを通してユーザモード側に転送することである。

デーモン vdiskd-http-ftp, vdiskd-local ユーザモードで実行するデーモンプログラムは、キャラクタデバイス /dev/vdiskio を通じて受け取った I/O パケットを解釈し、実際のネットワーク入出力処理を行う。デーモンは、各種外部ストレージご

Development of Distribuited Virtual Disk Using Online Storage
Yusuke NOJIRI†
†Graduate School of Engineering, Tokyo University of Agriculture and Technology

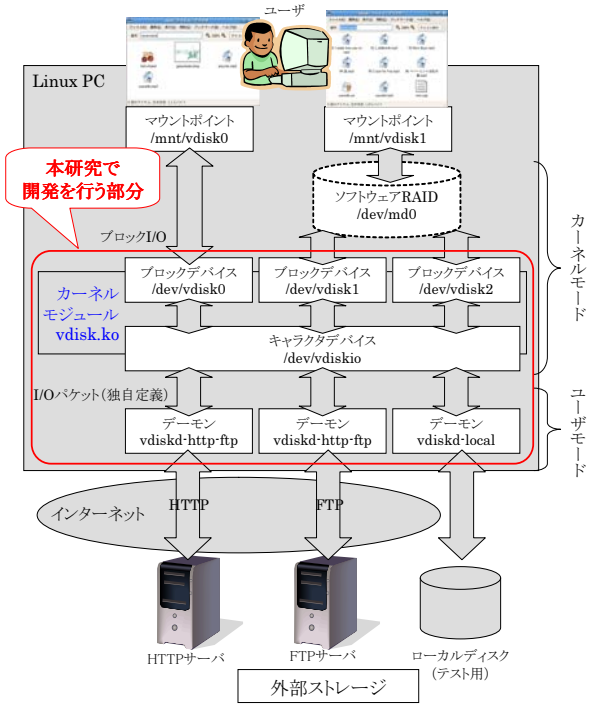


図 2: 仮想ディスクシステムの設計

とに特化したプログラムとする。デーモン vdiskd-http-ftp は、HTTP または FTP によりブロックファイルの入出力を行う。デーモン vdiskd-local は、ローカルディスクに対してブロックファイルの入出力を行うテスト用デーモンである。

4 評価

本システムの性能を評価するため、5 通りの条件で、スループットを計測するベンチマークテストを行った。クライアントマシンの CPU は Intel Pentium 4 2.80GHz、メモリは 512MB、OS は KNOPPIX 5.0.1 である。クライアント・サーバ間のネットワークは 1000BASE-T である。

ベンチマークのの結果を図 3 に示す。ローカルディスクに対する入出力で、本ドライバを介する場合と介さない場合では、スループットが読込み時で約 36%、書出し時で約 59%低下している。この値は本ドライバのオーバヘッドとみなせる。RAID-0 （ストライピング）のディスクアレイ化した場合では、むしろソフトウェア RAID のオーバヘッドが災いしたためか、多くの場合でスループットが低下している。ネットワークを介した場合のローカル環境に対するスループットの低下は小さく、最悪でも約 21%（「ローカル・RAID-0」/「HTTP・RAID-0」の読込み時）にとどまっており、妥当な結果である。

LAN 上でのベンチマークテスト行った結果，開発し

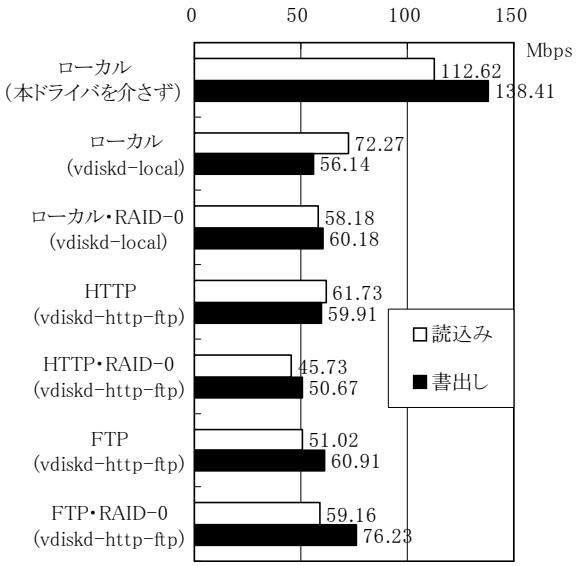


図 3: 仮想ディスクデバイスのスループット

たドライバのオーバヘッドは，スループットにして約 36%の低下（読込み時）であり，実用に堪える程度であるといえる．

5 おわりに

ディスクインタフェースをブロックデバイスドライバとしたことで，高い透過性を提供できた．この高い透過性を応用し，ホームディレクトリや OS そのものを本仮想ディスクシステムに載せる応用も考えることができる．

ベンチマークテストの結果から，開発したドライバのオーバヘッドは実用に堪える程度であるといえる．データの入出力においてユーザ空間を経由するため，多少のパフォーマンス低下は避けられないが，ユーザ空間を経由する設計にしたことによりもたらされる利点を考慮すれば，妥当な結果である．インターネットを介した場合，レイテンシの影響によりいいパフォーマンス得られないことも想定されるが，先読みの実装やブロックサイズの調整である程度改善できると考えている．

セキュリティに関しては，本システムの現状ではプロトコルの認証機構や暗号化（例えば SSL）に頼っている．本システムで独自にデータを暗号化する仕組みを持つなどして，さらにセキュリティを高めることは今後の課題である．

参考文献

[1] 産業技術総合研究所: HTTP-FUSE-KNOPPIX. http://unit.aist.go.jp/itri/knoppix/http-fuse/

オンラインストレージを用いた分散仮想ディスクの開発

Summer Joint Symposium 2007 for Advanced System Software
野尻 祐亮
東京農工大学
2007年9月12日

背景

- 「いつでもどこでも」個人データにアクセスしたい
  - ドキュメントファイル・メール・アドレス帳など
- どこでも同じデスクトップ環境を使いたい
  - 個人設定やソフトウェア構成など

ネットワークや各種物理メディアを用いて、どこからでもアクセスできるようにしたい

個人データの管理は面倒

- USB メモリなどの物理メディアは扱いたくない
  - 持ち運びが面倒、忘れると使えない
  - 紛失・盗難・故障のリスク

- オンラインのストレージサービスは不便
  - Web ブラウザなどを通しての操作が煩雑
  - 操作が統一されていない
    - ファイル一覧表示やメタ情報形式が異なる
    - アップロード・ダウンロード・コピー・削除などを実行するための操作が異なる

従来方式の問題点 (1)

- NFS (Network File System)
  - インターネット経由で利用できない
  - NFS サーバの準備が必要

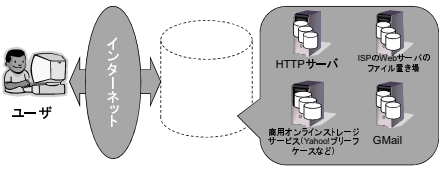
- iSCSI
  - 小規模システム向けではなく、個人ユーザが導入するにはコストが高い

従来方式の問題点 (2)

- FTP, SSH, HTTP (WebDAV), 各種オンラインストレージサービス
  - ファイルシステムやストレージドライバとしての利用は想定外
  - ユーザインタフェースの不統一
- HTTP-FUSE
  - 読込みに特化していて、個人データの保存には利用できない

概要

- **どこからでもアクセス可能な仮想ディスクデバイス**
  - インターネット上の物理ストレージを仮想化する
  - ユーザに物理的な構成を意識させない



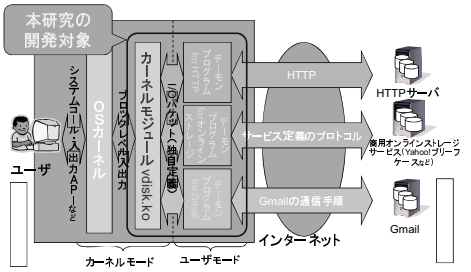
目標

- インターネット経由で利用可能
  - ユビキタスに利用できる
- 高い透過性
  - 物理的なディスクデバイスであるかのように見せかける
  - ソフトウェア RAID を用いて、性能・耐障害性向上
- 高い柔軟性
  - HTTP, FTP, 各種オンラインストレージなど、さまざまなプロトコルに対応可能

ファイルシステムレベル対ブロックデバイスレベル

- どちらの設計にすべきか？
  - ファイルシステムレベル: FAT や ext3 などと同様、ファイル単位でデータを管理
  - ブロックデバイスレベル: 物理的なストレージデバイスと同様、ブロック単位でデータを管理
- 高い透過性
  - 既存のファイルシステムやソフトウェア RAID が利用可
- 一貫性制御が不要
  - 一貫性制御はファイルシステムに任せる
  - 仮想ディスクへのアクセスはクライアント1台に限る

システム全体図



設計

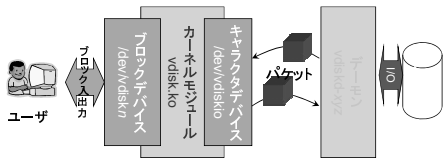
- デバイスドライバにより仮想ディスクを実現
- ドライバはカーネルモジュールとユーザモードのデーモンプログラムからなる
  - 外部ストレージ依存入出力処理をユーザモード部分が吸収する汎用的な設計
  - 外部ストレージの種類ごとに特化したデーモンプログラムを作成
    - 例えば HTTP に対応したデーモンプログラム、Gmail に対応したデーモンプログラム、…
  - ユーザモードなのでデーモンプログラムの実装が容易
    - ネットワーク入出力ライブラリなど既存のライブラリが利用できる
    - デバッグが容易

カーネルモジュール vdisk.ko (1)

- ブロックデバイスとしてのインタフェースをユーザに提供
  - ユーザにとって通常のディスクデバイス同様の操作性を実現
  - ソフトウェア RAID など既存のソフトと組み合わせることを可能に
- カーネル側からのブロック I/O をユーザモードのデーモンプログラムに転送

カーネルモジュール vdisk.ko (2)

- 2つの入出力インタフェースを持つ
  - ブロックデバイス（前述）
  - キャラクタデバイス
    - デーモンとパケット（後述）をやり取り



デーモンプログラム vdiskd-*xyz*

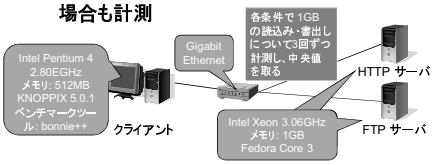
- プロトコルの差を吸収、プロトコル依存の入出力処理を行う
  - 例えば接続・切断、エンコード・デコード、送信・受信、セッション管理
- 外部ストレージとはブロック(256KB程度)単位で通信
- ブロック単位でキャッシュ

実装

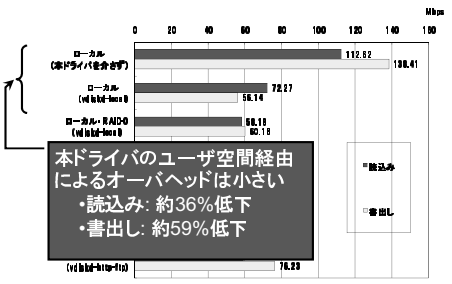
- Linux 2.6.17 デバイスドライバとして実装
- 実装言語
  - カーネルモジュール: C 計1142行
  - デーモンプログラム: C++ 計1403行
- 2種の外部ストレージ用デーモンを実装
  - vdiskd-http-ftp : HTTP/FTP用
  - vdiskd-local : ローカル用(テスト用)
- MD (Multi Disk) ドライバを用い RAID-0 (ストライピング) ディスクアレイ構築も
  - ディスク入出力を並列化してパフォーマンス向上を狙う

評価

- 仮想ディスクのスループットを計測
  - 外部ストレージはローカルディスク・HTTP サーバ・FTP サーバ(LAN内)とする
  - RAID-0 (ストライピング) のディスクアレイとした場合も計測



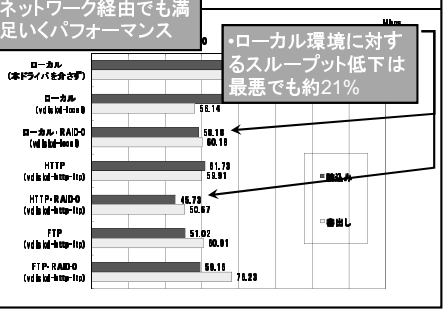
評価結果



評価結果

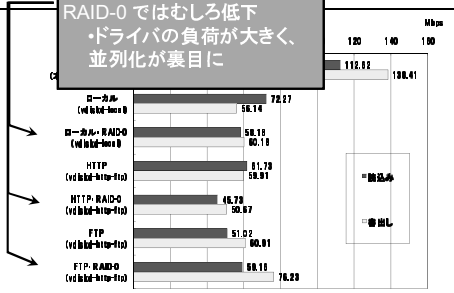
ネットワーク経由でも満足いくパフォーマンス

•ローカル環境に対するスループット低下は最悪でも約21%



評価結果

RAID-0 ではむしろ低下
•ドライバの負荷が大きく、並列化が裏目に



まとめ

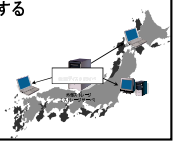
- インターネット経由で利用可能な仮想ディスクシステムを設計・実装
  - 高い透過性
  - 多様な外部ストレージへ容易に拡張可能な設計
  - オーバヘッドは約36%（読込み時スループット）、ネットワーク経由時で78%以上（ローカル環境比スループット）の高パフォーマンス

今後の課題

- セキュリティ強化
- その他の外部ストレージに対応
  - メールプロトコル (POP3/IMAP4/SMTP) 、SSH
  - GMail YahooBriefcase など
- 高負荷時の動作の改善
- 評価の強化

今後の展望

- ネットワーク断絶に対する対処
  - ローカルディスクに同時記録
- 完全ディスクレスシステムの実現に向けて
  - KNOPPIX へのドライバ組込み
    - ネットワーク経由でソフトウェア構成や個人データを保存できるようにする



センサネットワークにおける複数基地局を用いた自律的ネットワーク再構築手法

松井 雄一
立命館大学大学院理工学研究科

1　研究背景と目的

近年, 無線通信技術と半導体技術の発展により, 無線端末の小型化が進み, 無線ネットワークの適用分野が拡大している. これらを背景として, 小型で無線通信機能と複数のセンサ機能を持つ高性能なセンサノードが開発されている. 個々のセンサノードは，温度，湿度，加速度などを計測するセンサに加え，計算能力, メモリ, 無線送受信機を持ち, 多くの場合電池で動作する．このセンサノードを多数配置し, センサネットワークを構築することにより, 人や物の状態を正確に観測することが可能となる. センサノードが取得したセンサ情報は, 無線通信により直接, または間接的に基地局に集められ, 基地局上やインターネットを介してアプリケーションや利用者に提供される. センサネットワークは，環境モニタリング, 農業プラント，生物の生態調査への応用が考えられており, 利用規模の拡大やネットワーク管理の利便性を考慮した研究が行われている [2][3].

センサネットワークは，環境モニタリングを応用例として考えたとき，数十，数百という膨大な数のセンサノードが実世界上に広範囲に配置される．また, センサノードを管理する基地局もその数に比例して配置される. 環境内に配置されたセンサノードや基地局はメンテナンス作業が困難なため，センサネットワークによる長期間の観測のためには，手動による再調整を必要とせず，センサ情報の収集を継続できることが求められる． 特に, 基地局が故障して動作不可能な状況になった場合, 自身の機能だけでなく, 基地局が管理していたセンサノードの情報が取得できなくなるため, 基地局が使用不可能になった場合でも, 常にセンサノードの情報が取得できる新たな耐障害性機構が必要となる.

そこで, 本研究では, 基地局の故障の影響を受けないセンサ情報収集機構として，耐障害性の向上を目的とした自律的ネットワーク再構築手法を提案する.

2　次世代型斜面防災情報システム

2.1　想定環境

我々が想定している環境の 1 つである, 次世代型斜面防災情報システムの全体構成を図 1 に示す. 次世代型斜面防災情報システムとは, 土砂崩れの可能性がある場所にセンサノードを配置し, 現場の環境情報を取得することで, 土砂崩れの発生を予期したり, 今後生かされる土砂崩れに関するデータを取得する防災システムである. 各センサノードは, 定期的に環境情報を取得し, 取得した情報を基地局に送る役割を持つ. センサノードが取得する情報には, 温度, 湿度, 土壌水分量などがあり, これらの情報を定期的に取得する.

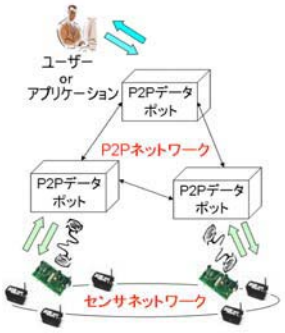


図 1　システムの全体構成図

2.2　ネットワーク構成

本システムはセンサネットワークと P2P ネットワークの 2 種類のネットワークで構成されている. センサネットワークには, センサノードと基地局が存在し, 基地局が, 各センサノードへの要求メッセージを無線送信することで, 要求に応じた環境情報を取得する. センサノードは, 基地局と直接通信できない場合でもマルチホップ通信を行うことで間接的にデータを届けることが可能である.

P2P ネットワークは, 複数の P2P データポット [1] と呼ばれるマイクロストレージサーバで構成されており, 複数の P2P データポットが相互に無線通信を行うことで自律的に構築される. P2P データポットは, クライアントからのクエリを基地局へ転送し, その結果を基地局より受信する. 基地局と P2P データポットは, RS-232C によってシリアル接続される.

3　既存技術

3.1　XMesh

XMesh は, 無線ネットワーク用に開発されたマルチホップネットワークプロトコルである. Xmesh を利用することにより, センサノードによる自律的ネットワークの構築が実現可能となる. また, 消費電力についても考慮されており, 常に各チップの電源がオンになった短時間集中型の観測を目的とした HP モード, および各チップに Sleep 時間を設けて長時間観測を目的とした LP モードという 2 つの電力モードが状況に応じて選択できるように設計されている.

3.2　考察

XMesh で複数の基地局が存在するセンサネットワークを実現したところ, ネットワークが不安定であり, 頻繁にネットワーク構成が変更するといった結果が得られた. これは, XMesh にとって複数の基地局が存在するセ

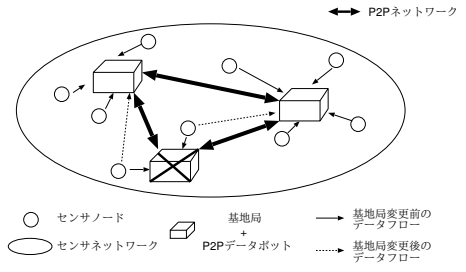


図 2　自律的ネットワーク再構築手法

ンサネットワーク環境が想定外であり, 各センサノードが複数の基地局を同一の基地局とみなしているために起こる問題であると考えられる. しかし, XMesh は, 自律的ネットワークの構築, 新機ノードの追加, マルチホップ通信など, センサネットワークを構築するための非常に多くの機能を持っているため, 利用価値は非常に高いと言える. そこで, XMesh を基に改良していくことにより, 自立的センサネットワーク再構築手法を実現する.

4　提案手法

4.1　概要

提案手法では, 基地局の故障検知, ネットワーク全体への故障検知通知, センサノードの自律的ネットワーク再構築という 3 つの手順を踏むことにより, 基地局の故障の影響を受けないセンサ情報収集機構を実現する. 図 2 に自律的ネットワーク再構築手法について示す. 本提案手法の前提条件として, 使用するセンサネットワーク環境内に複数の基地局が存在することと, ある基地局に属するセンサノードが他の基地局に属する少なくとも一つ以上のセンサノードと直接通信できる状態にあることを仮定する.

4.2　故障検知機能

基地局の故障の影響を受けない情報収集機構を実現するためには, 基地局が故障したことを検出する必要がある. 現状では, ネットワークコネクションのタイムアウトを設定することで故障検知の判断を行っている. しかし, タイムアウトにより基地局の故障検知を行う場合, タイムアウトになるまで故障検知することが出来ない, 誤った故障検知によりネットワークの再構築が頻繁に行われるといった問題が挙げられる. また, タイムアウトの設定を変更することは, センサネットワーク全体に影響を与えるため, タイムアウトのみでは, 故障検知の手法としては不十分である.

そこでタイムアウトによる故障検知に加えて, 基地局の状態を正確に判断する機能を実現することで問題を解決する. 基地局の故障検知の判断は, 隣接するセンサノードが行う. 基地局の状態には, 正常に通信できている状態, 故障により通信できない状態, 一時的に通信できない状態などの状態が存在し, これを正確に判断することで問題を解決できると考える. 現在, 故障検知機能についての詳細な手順は考察中である.

4.3　故障検知メッセージ

基地局の故障が検出されたとき, その基地局に所属する全てのセンサノードに通知する必要がある. その際, ネットワークに送信するメッセージを故障検知メッセージと呼ぶ. 故障検知メッセージは, 基地局の故障を検知したセンサノードによって生成され, 自身の子ノードに送信される. 故障検知メッセージを受信したセンサノードは, さらに自身の子ノードへとメッセージを転送していく. これを繰り返すことによって, 故障した基地局に属するセンサノードにだけ故障検知メッセージを転送することができる.

4.4　部分的ネットワーク再構築

故障検知メッセージにより, 故障した基地局に属するセンサノードによる部分的ネットワークの再構築が可能となる. 電力資源に制限があるセンサーネットワークにおいて, 部分的ネットワーク再構築は, センサネットワーク全体のネットワークを再構築するのに比べ, 無線送受信回数を抑えることができ, その結果消費電力を抑えることになるため, 非常に有効な手段である.

センサノードは, 自身の親ノードを選択するのに隣接ノードテーブル情報を利用するため, 過去の情報に依存したネットワークが構築されてしまう可能性がある. これを防ぐため, まず, センサノードは自身の隣接ノードテーブルの情報をリセットし, 新たに隣接ノードの情報を格納する. 新たに隣接ノードテーブルに格納された隣接ノードの中から, ホップ数・受信成功率・送信成功率の情報を基に, 最適な親ノードを選択する.

5　おわりに

本稿では, 複数センサネットワーク環境における耐障害性の向上を目的とした自律的ネットワーク再構築手法について述べた. 本手法は, 故障検知機能, 故障検知メッセージ, 部分的ネットワーク再構築の３つの機能を実装することで基地局の故障の影響を受けないセンサ情報収集機構を実現する. 現在の進捗としては, 故障検知メッセージの実装が完了し, 部分的ネットワークの再構築機能の考察を行っている. 今後は, これらの機能の実装と評価を完了させる.

参考文献

[1] 藤崎 友樹, 鈴来 和久, 横田 裕介, 大久保 英嗣 : “P2P データポット: センサーネットワーク向け分散型マイクロストレージアーキテクチャ” 電子情報通信学会第 18 回データ工学ワークショップ, DEWS2007 D1-1 (2007)

[2] S.R.Madden, M.J.Franklin, J.M.Hellerstein, W.Hong : “An Acquisitional Query Processing System for Sensor Networks,” ACM TODS Vol.30, Issue 1, pp.122-173, March 2005

[3] 牧村 和慶, 斉藤裕樹, 戸辺義人 : “センサネットワーク階層的領域管理システム,” 電子情報通信学会 センサネットワーク時限研究専門委員会 (第一回 SN 研究会), pp.69–76 (2004).

センサネットワークにおける複数基地局を用いた自律的ネットワーク再構築手法

立命館大学院理工学研究科
大久保・横田研究室
松井　雄一

発表内容

- はじめに
  - 複数基地局を用いたセンサネットワーク
- センサネットワーク環境における課題
- 想定環境
  - 次世代斜面防災システム
- 既存技術
  - Xmeshアプリケーション
- 提案手法
- おわりに

1

はじめに

- 無線通信技術や半導体の発展
- 小型化かつ高性能なセンサノード
  - 複数のセンシング機能
  - 無線通信機能

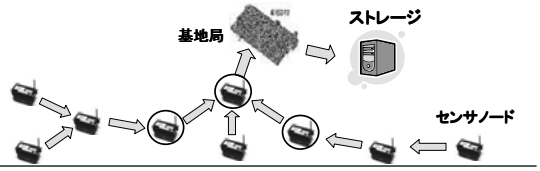
センサネットワーク技術の発展

- 実世界の環境情報を取得
  - （例）：温度，湿度，照度，加速度，位置情報
- センサネットワークを利用する規模の拡大
  - 数十、数百のセンサノードを使用した環境モニタリング

2

センサネットワーク

- センサノードが取得するデータは必ず基地局を経由してストレージに格納される
- センサノードの数が増大するにつれて・・・
  - 中継ノードの処理の増大　→　パケットロス
  - 中継による電力消費の増大　→　ネットワークの遮断

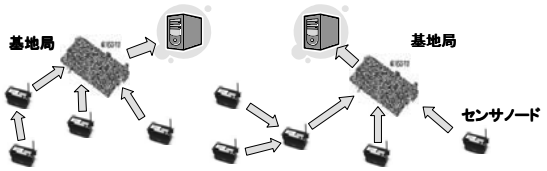


3

複数基地局を用いたセンサネットワーク

- 複数の基地局を設置
  - 中継ノードの負担が分散
  - ネットワーク全体的にホップ数の減少

  →長期運用、拡張性の面で優れている



4

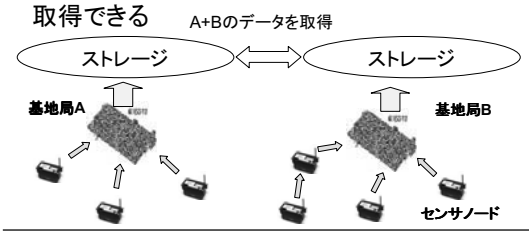
センサネットワーク環境における課題

- 大規模なセンサネットワーク環境において，消費電力，通信の低信頼性が問題
  - センサノードの資源に制限
  - ホップ数の増加による電力消費の増加とネットワークトラフィックの増大
- 実環境に配置した場合，センサネットワークの管理が困難
  - 機器(センサノードと基地局)の故障や位置の変化によるネットワークの遮断
  - 人が立ち入りにくく，広範囲な設置環境
    - 回収・修理がしにくい

5

基地局の故障による問題

- センサノードが取得するデータは必ず基地局を経由してストレージに格納される
- 通常状態では全てのセンサノードのデータが取得できる

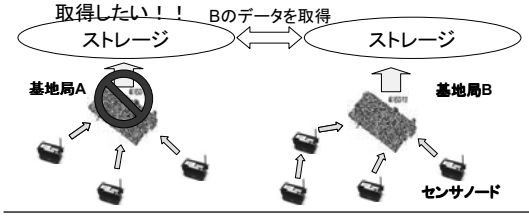


6

基地局の故障による問題

- 基地局Aが故障
  - 全てのセンサノードのデータが取得できなくなる
  - 基地局Bに属するセンサノードはデータを取得可能

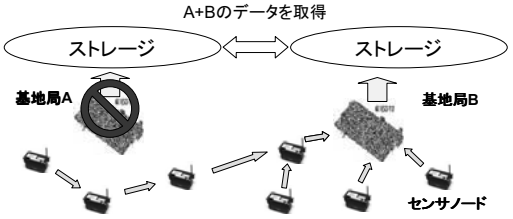
  → 基地局Aに属するセンサノードのデータを取得したい！！



7

基地局の故障による問題

- 基地局Aに属するセンサノードが、自立的に基地局Bのネットワークに参加
  - 基地局Bに属するセンサノードを中継し，全てのセンサノードのデータを取得することが可能



8

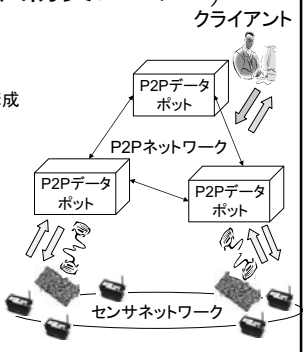
研究目的

- 基地局の故障の影響を受けないセンサネットワークシステムの構築
  - 屋外環境におけるシステムの長期運用
  - トポロジーの変化に対する迅速な対応
  - メンテナンスフリーによるコスト削減

9

想定環境(次世代斜面防災システム)

- センサネットワーク
  - 多数センサノード+多数基地局
  - 自律的にセンサネットワークを構成
  - センサが環境情報を取得
  - 基地局はセンサの情報を収集
- P2Pネットワーク
  - 複数P2Pデータポットで構成
    - ゲートウェイ機能を持ったマイクロストレージサーバ
  - センサの取得したデータを蓄積
  - 相互にP2Pネットワークを構成
  - 外部からの要求に対して応答



10

既存技術(Xmeshアプリケーション)

- センサネットワーク構築する上で多くの機能を有したアプリケーション
  - マルチホップ通信
  - 自律的ネットワーク構築
  - 新機ノード追加，離脱ノードのためのトポロジー更新

基地局が複数存在する環境は想定していない
→複数の基地局を同一とみなし，ネットワークが安定せず総ホップ数が増加する
しかし、利用できる機能が多々存在する
→Xmeshアプリケーションを改良し，研究目的を実現する

11

提案手法のシステム構成

- 基地局の故障検知
  - 基地局に隣接するセンサノードが基地局の故障を判断
- 故障検知msg
  - 基地局が故障したことをネットワーク全体に知らせるためのメッセージ
- 部分的ネットワーク再構築
  - 故障検知msgを受信したセンサノードが，新たな親ノードを選択しネットワークに参加

12

基地局の故障検知

- 故障検知が正確であり，検知する時間が早ほど理想的
- 現状は，タイムアウトによる故障検知
  - データの遅延が発生したり，ネットワークが不安定になるといった問題が挙げられる

  →タイムアウトに加え、正確に基地局の状態を判断する機能が必要
- 基地局の状態判断
  - 正常状態
  - 故障により全く通信できない状態
  - 通信混雑時や遮蔽物による一時的に通信できない状態

→正確に判断できれば，冗長なパケットの減少やデータの遅延を防ぐことができる

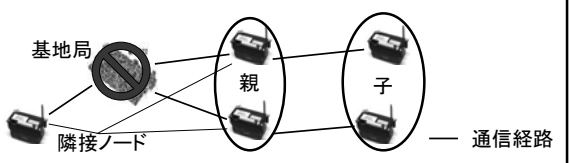
基地局の故障検知については現在考察中

13

故障検知msg

- 基地局に隣接するセンサノードが，故障を検知したことをグループ全体に知らせる
- 故障検知メッセージは、自身の子ノードにだけ転送する
  - 基地局の故障で影響を受けるセンサノード

  →部分的ネットワークの再構築



14

部分的ネットワーク再構築

- 最近の近隣ノードの状況を取得
  - 通信状況に関する累積情報を保持
  - 各ノードが格納している隣接ノードの情報をリセット
- 親ノードの選出
  - 通信の信頼度が重要
    - 過去の累積通信情報を考慮する必要がある
    - 通信の信頼度＝送信成功率＋受信成功率
  - 基地局までのホップ数の考慮
    - 少ないホップ数である親を選択することで電力削減
    - 総ホップ数の減少によりネットワークトラフィックの減少

15

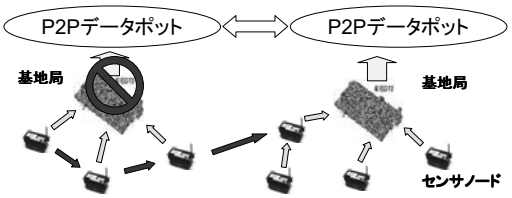
おわりに

- まとめ
  - 研究背景
  - 想定環境におけるセンサネットワークの問題点
  - 提案手法
    - 自律的ネットワーク再構築による耐障害性の向上
- 今後の課題
  - 故障検知機能の考察
  - 親ノード選出のための具体的な手法の考察
  - 故障検知機能と部分的ネットワーク再構築機能の実装
  - 性能評価
    - 既存技術(Xmeshアプリケーション)及び, 既存研究との比較

16

部分的ネットワーク再構築

- 故障検知msgを受け取ったセンサノードは, 自身の隣接ノードテーブルをリセットする
- 一定時間経過後,ホップ数・受信成功率・送信成功率により新たな親ノードを選択する



17

センサネットワーク

- 固定に配置する定点センシング
  - 単純な定点観測
  - 拡張性・柔軟性の欠如
- 移動体によるセンシング
  - センサが人間とともに移動(パーソナルセンシング)
    - センサを稠密に配置する必要がない
    - 安価, 精度よく広範囲の環境情報を取得可能

⇩

移動センシングプラットフォームの構成

18

多数のセンサノードが配置された状況で位置測定を行う場合における誤差伝播の評価

寺嶋 邦浩 †

† 立命館大学大学院理工学研究科

1 はじめに

現在, 我々は, センサネットワークを利用した移動センシングプラットフォーム技術に関する研究を行っている. 移動体によるセンシング技術は, 広範囲な空間において, 高解像度の位置情報とともに環境情報を収集可能とする技術である. 従来は, センサノードを固定・稠密に配置し, その場の環境情報を取得するパッシブな方法であった. 一方, 移動体によるセンシング技術は, 人間とともに移動するセンサノードが広範囲の環境情報をセンシングするアクティブな方法である. したがって, センサノードを稠密に配置することなく, 安価かつ高精度に広範囲の環境情報を取得できる.

環境情報を取得する際には, 環境情報とともに得られる位置情報の精度が重要である. 位置測定では, 位置が既知であるセンサノードが測位対象と通信を行い, 得られる距離情報から対象の位置を決定する. 現実には, さまざまな影響により, 得られる位置情報は誤差を含む. 測位誤差の値を正確に把握することは不可能であるため, 高精度な位置測定を行うことは, 非常に難しい問題となっている. しかし, 測位誤差の値を正確に把握することは困難であるが, 位置測定に用いるパラメータから, 位置測定に対する信頼度を決定し, 測定結果の精度を事前に予測することは可能であると考えられる. そこで, 本稿では, 多数のセンサノードが配置された状況において, 測位精度が高いと予測されるものから測位を行う手法について述べる. 本手法により, ネットワーク全体での測位誤差を減少させることが可能であるとともに, ネットワーク全体で利用するセンサノード数を削減可能となる.

2 位置測定の概要

位置測位では, ビーコンノードと未知ノードが通信を行うことにより, 未知ノードの位置を決定する. ここで, ビーコンノードは, 位置が既知であるセンサノードを指し, 未知ノードは測位対象を指す. 位置測位の手順は, 距離の推定と位置の推定の 2 つのステップに分けることができる. 位置測位の様子を図 1 に示す. 距離の推定では, ビーコンノードと未知ノードが何らかの方法で通信を行い, その結果を用いてビーコンノードと未知ノード間の距離を推定する. 位置の推定では, 距離の推定で得られた距離情報を基地端末に集め, 集めた距離情報とビーコンノードの位置情報から幾何学的に位置関係が決定されるため, 計算により未知ノードの位置を推定する.

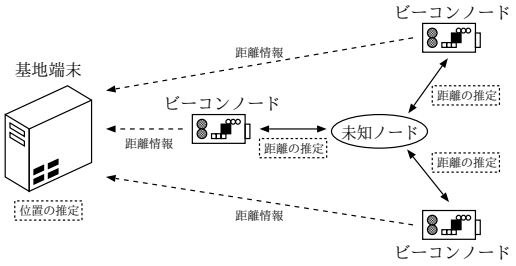


図 1 位置測定の概要

3 測位誤差の評価手法

3.1 評価手法の想定環境

本手法では, 多数のセンサノードが配置された状況を想定している. 多数のセンサノードの位置測定を行うために多数のビーコンノードを配置する場合, センサノードが高価なものであることや配置する労力を考えるとコストパフォーマンスが低い. しかし, ビーコンノードが少ない場合, 空間全体の未知ノードをカバーし切れない. そこで, 位置測定を終えた未知ノードを擬似的なビーコンノードとして次の位置測定に利用することで, 空間全体をカバーする. このとき, 位置測定を終えた未知ノードの位置情報に誤差が含まれていると, 位置測定を進めるごとに未知ノードの位置情報の誤差が蓄積・増大していく問題がある.

多数のセンサノードが配置された状況で位置測定を行う場合, 誤差が蓄積・増大する問題を解決することができれば, ビーコンノードの配置するためのコストを削減するとともに, 信頼度の高い位置測定を実現することが可能である. 本手法は, 誤差の蓄積・増大を減少させることで, ネットワーク全体として信頼度の高い位置測定を実現することを目的としている.

3.2 評価手法の概要

距離の推定と位置の推定を行うことにより, 未知ノードの位置を決定することができる. しかし, 現実には, センサノード内部でのゲート遅延や配線遅延, 通信信号の伝播遅延から距離情報は誤差を含む可能性が高く, 誤差を含む距離情報から決定される未知ノードの位置情報に対する信頼度は変化する. 距離情報の信頼度を高めることが可能であれば, 測位精度に対する信頼度は向上するが, 通信信号の伝播遅延は位置測定を行う環境によってさまざまに変化するため, 距離情報の信頼度を高めることは困難である. また, センサノードの位置関係が未知

ノードの位置情報に影響を与える可能性があり, 位置関係によっては, 大きな誤差が発生しやすい状況となる.

このように, 単体での位置測定における測位誤差を減少させることは, 非常に困難である. そこで, 本手法では, 単体での位置測定における測位値に対する信頼度を数値化して評価し, 信頼度の高いものから位置測定を行うことで, 誤差の蓄積・増大を減少させる. すなわち, 単体での測位誤差を減少させるのではなく, ネットワーク全体としての測位誤差を減少させることに主眼をおく.

3.3 評価手法に用いるパラメータ

位置測定では, 距離の推定のステップで得られた距離情報をもとに, 位置の推定のステップにおいてセンサノードの位置関係から未知ノードの位置を決定する. よって, 測位値に影響を与える要因は, センサノード間の距離とセンサノードの位置関係である. センサノード間の距離とセンサノードの位置関係から測位に対する信頼度を数値化することで, 測位の順序を決定することが可能であると考えられる. 本手法では, センサノード間の距離とセンサノードの位置関係を評価パラメータに用いる.

3.3.1 センサノード間の距離

センサノード間の距離が離れるほど, 通信信号の減衰の影響により, 受信機が通信信号を認識するタイミングがずれ, 距離情報に大きな誤差が発生する可能性が高くなる. 逆説的に考えれば, 得られた距離情報が大きな値の場合, 得られた距離情報には大きな誤差が含まれている可能性が高いと言える. そこで, センサノード間の距離と得られる距離情報のばらつきの関係について調査を行った. 調査に用いたセンサノードは Cricket [1] である.

センサノードの距離が $5m$ まで, $50cm$ ごとの距離情報のばらつきを図 2 に示す. ばらつきは, それぞれ 2000 個の距離情報の標準偏差で表した. 各散布データから, 図 2 に示す回帰曲線が得られ, 回帰曲線の決定係数 R^2 は, $R^2 = 0.8592$ であった. このことから, センサノード間の距離が離れるほど, 距離情報のばらつきは指数関数的に増大していくことがわかる. よって, 得られた距離情報と回帰曲線から得られる標準偏差の値を, 測位値に対する信頼度を決定するためのパラメータとして利用可能であると考えられる.

3.3.2 センサノードの位置関係

センサノードの位置関係により, 同じ距離情報を用いて測位しても, 測位値に誤差が発生しやすい状況になる可能性がある. そのため, センサノードの位置関係を数値化することにより, 測位値に対する信頼度を決定するためのパラメータとして利用可能であると考えられる.

位置測定手法 multi-lateration [2] の式を距離情報について整理することで, 式 (1) が得られる.

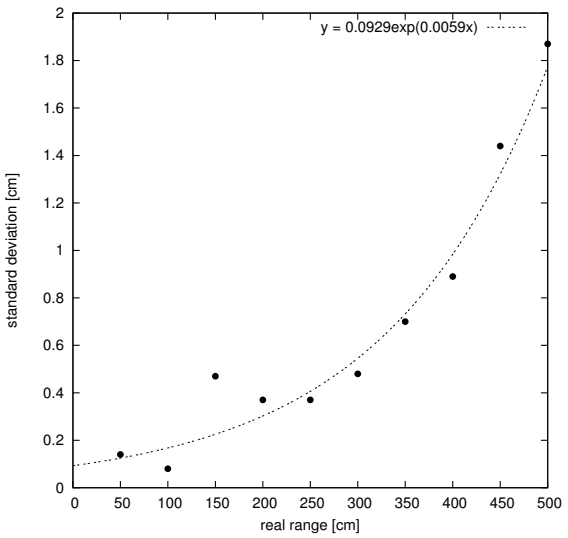


図 2　距離ごとの測定値の標準偏差

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} z_{xi}(d_i^2 - d_k^2) + t_x \\ \sum_{i=1}^{N} z_{yi}(d_i^2 - d_k^2) + t_y \end{bmatrix} \quad (1)$$

ここで, d_i, d_k は距離情報であり, z_{xi}, z_{yi}, t_x, t_y はセンサノードの位置関係により決定される定数である. また, i, k は $1 \leq i \leq N, 1 \leq k \leq N, k \neq i$ を満たす整数であり, N はセンサノードの総数である. 式 (1) から, 全てのセンサノード間の距離情報が等しい場合, 得られる位置情報はセンサノードの位置関係に固有なものであることがわかる. これにより, センサノードの位置関係は, t_x, t_y を用いて数値化可能であると言える.

4 おわりに

本稿では, 多数のセンサノードが配置された状況において, 測位に対する信頼度を数値化し, 信頼度の高いものから測位を行う手法を述べた. 本手法を用いることにより, ネットワーク全体としての測位精度を向上可能である. 今後の予定として, 信頼度決定に用いるパラメータの重み付けを決定し, 実装と評価を行う.

参考文献

[1] Priyantha, N. B., Chakraborty, A. and Balakrishnan, H., “The Cricket Location-Support system,” In Proceedings of the 6th Annual international Conference on Mobile Computing and Networking (MobiCom ’00), ACM Press, pp.32-43, 2000.

[2] Savvides, A., Han, C., and Strivastava, M. B, “Dynamic Fine-Grained Localization in Ad-Hoc Networks of Sensors,” In Proceedings of the 7th Annual international Conference on Mobile Computing and Networking (MobiCom ’01), ACM Press, pp.166-179, 2001.

多数のセンサノードが配置された状況で位置測定を行う場合における誤差伝播の評価

立命館大学大学院理工学研究科
大久保・横田研究室
寺嶋邦浩

発表内容

- はじめに
- 位置測定の概要
- 提案手法
  - 概要
  - 想定環境
  - パラメータ
    - センサノード間の距離
    - センサノードの位置関係
- おわりに

はじめに

- 現在, 大久保横田研究室が行っている研究
  - センサネットワークを利用した環境情報収集技術
    - 移動センシングプラットフォーム技術
    - 次世代型斜面防災システム

    ✓広範囲な空間において, 高解像度の位置情報とともに環境情報を収集可能とする技術である.
- 得られる環境情報に位置情報は必須であり, 高解像度の位置情報が求められている.

  ✓高解像度の位置情報をどのように得るか？

はじめに

- 得られる位置情報は, 様々な影響から誤差を含む.
  - センサノード内部の遅延誤差
  - 測定距離の長さ
  - センサノードの位置関係

    ✓測位誤差の値を正確に把握することは困難である.
  - 測位誤差軽減のアルゴリズム
    - × 測位対象が単体の位置測定
    - ○ 測位対象が複数の位置測定

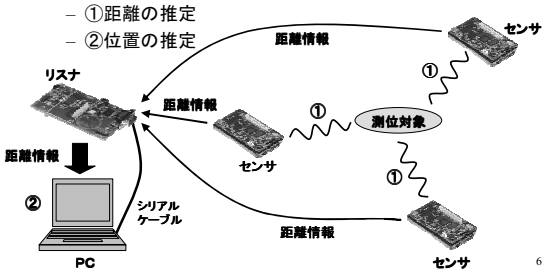
はじめに

- 信頼度パラメータの設定
- ネットワークにおける測位順序決定アルゴリズム
  - ➢信頼度が高いものから測位を行うことで, ネットワーク全体としての測位精度を高める手法を提案する.
- 語句の定義
  - ビーコンノード･･･位置が既知のセンサノード
  - 未知ノード･･･位置が未知のセンサノード

5

位置測定

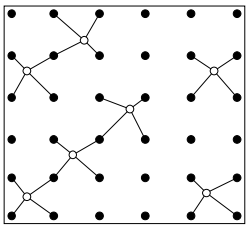
- 位置測位の2つのステップ
  - ①距離の推定
  - ②位置の推定



6

提案手法の想定環境

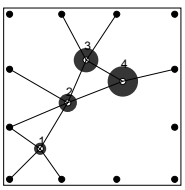
- 多数のセンサノードが配置された空間を想定する.
  - 多数の未知ノードを多数のビーコンノードで測位する.
    - センサノードの価格と配置する労力を考えるとコストがかかる.



7

提案手法の想定環境

- 多数のセンサノードが配置された空間を想定する.
  - 多数の未知ノードを少数のビーコンノードで測位する.
    - 測位を終えた未知ノードを擬似的なビーコンノードとする.
    - 測位を進めるごとに, 誤差が蓄積・増大していく.



※誤差の蓄積・増大を抑えることができれば, ビーコンノード数の削減および高信頼度の位置測定が実現可能である.

8

提案手法の概要

- ネットワーク全体としての測位誤差を減少させることを主眼とする.
  - 単体での位置測定の誤差を減少させることは困難である.
    - 距離の推定において, センサノード内部での遅延誤差や通信信号の伝播誤差により, 正確な距離情報を取得することが困難である.
    - センサノードの位置関係が測位値に影響を与える.

✓ 位置測定に利用するパラメータから, 測定結果に対する信頼度を事前に予測することは可能である.

➢ 測定結果に対する信頼度を数値化し, 信頼度の高いものから測位を行うことで, 精度の高い位置測定が実現可能である.

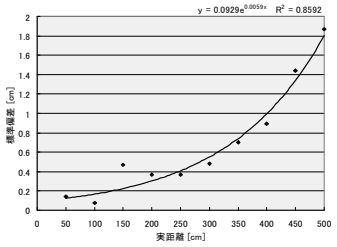
9

提案手法に用いるパラメータ

- センサノード間の距離
  - センサノード間の距離が離れるほど, 受信機が信号を認識するタイミングがずれ, 大きな誤差が発生しやすくなる.
  - 得られた距離情報が大きな値の場合, 得られた距離情報には大きな誤差が含まれている可能性が高い.
- センサノードの位置関係
  - センサノードの位置関係によっては, 測位値に誤差が発生しやすい状況になる.
  - GPSのPDOPと呼ばれる指標で示されている.
  - センサノードの位置関係を数値化することにより, 評価パラメータとして用いる.

10

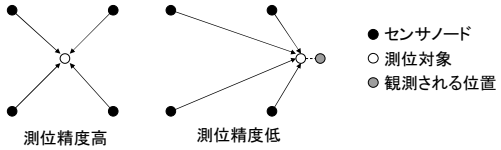
センサノード間の距離



11

センサノードの位置関係

- 測位対象とセンサノードの位置関係には密接な関係がある.
  - GPSのPDOPと呼ばれる指標で示されている.



- センサノードの重心と測位対象の距離の関係
  - ➢ 測位精度に影響する.

12

センサノードの位置関係

- 位置測定手法multi-laterationの式を変形すると，次式が得られる．

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} z_{xi}(d_i^2 - d_k^2) + t_x \\ \sum_{i=1}^{N} z_{yi}(d_i^2 - d_k^2) + t_y \end{bmatrix}$$

  - x, y は，求める位置情報
  - Nは，ビーコンノードの総数
  - d_i, d_k は，距離情報
  - z_{xi}, z_{yi}, t_x, t_y は，センサノードの位置関係に依存する値
    - ✓ 計算に利用する全ビーコンノードの座標値から決定される．
  - $1 \leqq i \leqq N$, $1 \leqq k \leqq N$, $i \neq k$

おわりに

- 本発表のまとめ
  - 多数のセンサノードが配置された状況において，測位に対する信頼度の高いものから測位を行う手法を述べた．
  - 提案手法の有用性は，次に示す通りである．
    - 初期配置するセンサノード数の削減
    - ネットワーク全体での測位誤差の減少
- 今後の予定
  - 信頼度決定に用いるパラメータの重み付けを行う．
  - 提案手法の実装と評価を行う．
  - 最終目標として，提案手法における測位順序決定アルゴリズムの完成を目処にする．

センサネットワークを利用した屋内移動体測位システムの概要

高木 敬介 †
† 立命館大学大学院理工学研究科

1 はじめに

現在, 我々は, 移動体センシングプラットフォームの研究を行っている. 移動体センシングプラットフォームとは, センシング機能を持つ無線通信端末を人や物を始めとする移動体に携帯させ, 位置情報や取得した周囲の環境情報を利用してサービスを提供するための基盤技術である.

移動体センシングプラットフォームを利用したシステムは, 位置に基づくルーティングや, 環境モニタリングといったセンサネットワークを利用したシステムに応用可能である. また, ナビゲーションシステムやトラッキングシステムといった, 従来ではモバイルアドホックネットワークや GPS といった高度な技術で実現されていたシステムをセンサネットワークで実現可能にする.

そこで, 移動体センシングプラットフォームを実現するための要素技術である測距エンジン, 測位エンジン, 描画エンジンを開発している. 測距エンジンは, 空間に配置された固定端末と移動端末間の距離を測定するための技術である. 測距は, 電波強度から距離を測定したり, 超音波やを使う特殊なハードウェアを利用することにより測距を実現する. 測位エンジンは, 測距エンジンから取得した距離情報と固定端末の位置情報に基づき, 幾何学的な演算処理によって移動端末の位置を推定する. 描画エンジンは, 測位エンジンから取得した位置情報に基づき, 移動体の速度や位置をグラフィカルな空間上にマッピングする機能を持つ.

2 システムの構成要素

我々が研究している屋内移動体測位システムは, 屋内に固定端末が複数設置されており, 固定端末の座標が, 固定端末自身, あるいは, ネットワークで接続された計算機システム上に, あらかじめ入力されていることを前提としている.

本章では, その屋内移動体測位システムに必要不可欠となる, 技術的構成要素について述べる.

2.1 測距エンジン

測距エンジンは, 固定端末と移動端末間の距離を測定する測距機能を持つ. 測距機能を実現するためには, 以下の方法が考えられる.

- TDoA (Time Difference of Arrival)
  伝搬速度の異なる 2 つの信号の到達時刻から端末間の距離を計測する. 2 つの信号送出のためのハードウェアが搭載されている必要があるので, 端末にかかるコストが高いが, 超音波などを用いることで, センチメートル精度の測距が行える.
- ToA (Time of Arrival)
  信号の伝搬時間から端末間の距離を計測する. 無線信号を利用する場合端末に高精度な時刻同期が必要となる.
- RSSI (Receive Strenght Signal Indication)
  無線信号の電波強度から端末間の距離を計測する. 電波強度地図の作成が困難である.

また, 信号の方向により, 移動端末が固定端末との距離を取得する Mobile Passive 型, 固定端末が移動端末との距離を取得する Mobile Active 型の測距エンジンがある. Mobile Passive 型の測距エンジンを持つシステムでは, 移動端末が複数ある場合, それぞれの移動端末が干渉しないため, スケーラビリティにおけるメリットがある. 一方, Mobile Active 型の測距エンジンを持つシステムでは, 複数の固定端末は移動端末との距離情報を同時に得ることができるため, 測位精度が向上する.

本研究では, 測位エンジンに MIT が開発したセンサノードである Cricket を利用している. Cricket は, 超音波と電波を利用した TDoA 測距を行う測距エンジンである. Cricket は, 測距信号を送信する Listener モードと信号を受信する Beacon モードで動作する端末を組み合わせることにより Mobile Passive 型, または, Mobile Active 型の測距エンジンとして利用できる.

2.2 測位エンジン

測位エンジンは, 測距エンジンから取得した距離情報と固定端末の位置情報に基づき, 幾何学的に 2 次元, もしくは, 3 次元座標で移動端末の位置を導出する機能を持つ.

利用する情報により, 位置推定には次の方式がある.

- bi-lateration
  それぞれ, 2 つの距離情報と固定端末の位置情報から移動端末の位置を推定する. 測距性能に誤差が極めて少ない場合には, 測位性能が高い.
- multi-lateration
  それぞれ, 3 つ以上の距離情報と固定端末の位置情報から移動端末の位置を推定する. 情報を選定することで, 測位精度を向上させることが可能である. 組合せが膨大であるため, 演算にかかる時間と計算機にかかる処理性能にコストが費される.

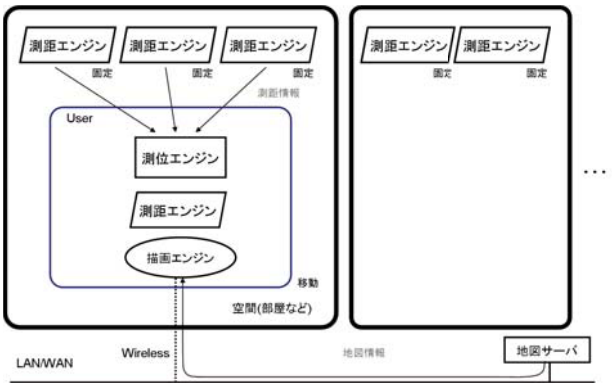


図 1　ナビゲーションシステム構成例

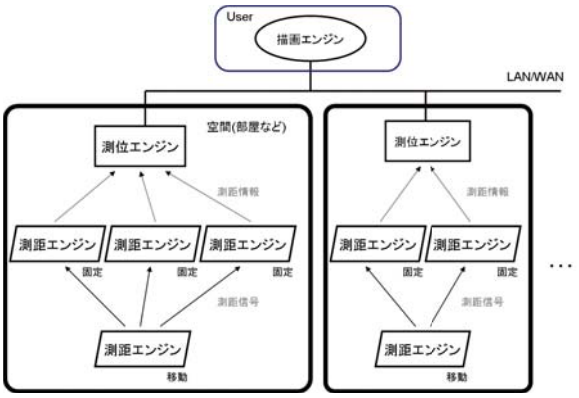


図 2　トラッキングシステム構成例

- tri-lateration
  信号の受信方角を付加的に利用する方式である．したがって, 測距エンジンには, 信号の受信号方角を検知するための特殊なデバイスが必要となる．

また, 測位エンジンには, 測距エンジンにより入力される情報のキャッシュ・フィルタリング機能, 複数の測位方式の選択機能, 入力する測距エンジンの選択機能を持つ．

2.3　描画エンジン

描画エンジンは, 測位エンジンから入力される位置情報に基づき, 移動端末の情報をグラフィカルに表示する．次に, 描画エンジンに搭載される機能を示す．

- 描画
  移動端末や移動端末が存在する地図情報をグラフィカルに表示する．移動端末の移動した軌跡を描画する．
- 移動端末の速度推定
  入力される位置情報に時刻情報を付加し, 移動端末の移動速度を推定する．移動速度を推定することにより, 移動端末の位置の予測や, 入力される位置情報のフィルタリングを行う．
- 地図情報の管理
  移動端末が存在する空間に関する情報である地図情報を管理する．地図情報には, システムで一意に決定される空間 ID, 空間のサイズ・形状, 空間に配置されている机や家電製品等の物体情報, 物理的に接続されている空間のリンク情報などがある．
- マップマッチング
  地図情報に記述されている情報に基づき, 移動可能・不可能な範囲を決定し, 測位情報を補正する．

現在, 測距・測位エンジンは, Microsoft が開発した言語である C# version 2.0 を用いて実装を行っており, 実行環境として .NET Framework 2.0 を必要とする．

3　システムの構成例

本章では, 2 章で述べたシステムの構成要素を組み合わせることにより実現可能な応用例として, ナビゲーションシステムとトラッキングシステムについて述べる．

3.1　ナビゲーションシステム

ナビゲーションシステムのシステム構成を図 1 に示す．ナビゲーションシステムでは, 固定端末がトリガとなり, 移動端末が距離情報を保持する Mobile Passive 型の測距システムを採用することで, スケーラビリティに優れたシステムを構成することができる．移動端末は, 現在地を表示するためのグラフィカルなインタフェースや無線通信インタフェースなどの高価な計算機資源が搭載されていることが期待できる．したがって, 地図情報は, 地図サーバとしてリモートホストにあらかじめ格納することができる．

3.2　トラッキングシステム

トラッキングシステムのシステム構成を図 2 に示す．トラッキングシステムでは, 移動端末がトリガとなり, 固定端末が距離情報を保持する Mobile Active 型の測距エンジンを採用し, 空間ごとに配置された測位エンジンと遠隔地に存在する描画エンジンの組合せで動作する．

本システムでは, 次の手順でトラッキングを実現する．(1) 移動端末の位置を特定するために, 描画エンジンは測位エンジンに対して追跡クエリを発行する．(2) 測位エンジンは, ネットワーク全体から追跡対象である移動端末を探索する．(3) 追跡クエリを受信した移動端末は, 測位エンジンに対して定期的に測位トリガを発行する．(4) 測位エンジンは, 移動端末の位置情報を描画エンジンに通知する．(5) 描画エンジンは, 測位エンジンから得た測位結果を用いて移動端末の軌跡を描画することによりトラッキングを実現する．

4　おわりに

本稿では, 屋内移動体測位システムの構成要素とシステムの実現例について述べた．本稿で述べた手法を実現することにより, ナビゲーションシステムやトラッキングシステムなど, 高度な機能を持つ応用システムを実現可能である．今後は, 各構成要素の実装を進め, 移動体測位システムの評価を実施する

センサネットワークを利用した
屋内における移動体測位システムの概要

JSASS 2007 (2007/09/12)

立命館大学大学院 理工学研究科 博士課程前期課程
情報理工学専攻 計算機科学コース
大久保・横田研究室 M1 高木敬介

1

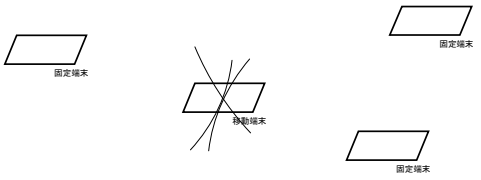
はじめに

- センサネットワークを利用した
  移動体センシングプラットフォームの研究
- センサネットワークによる測位技術を利用した
  システム開発
- 応用例
  - 屋内ナビゲーションシステム
  - 屋内トラッキングシステム

2

基本原理

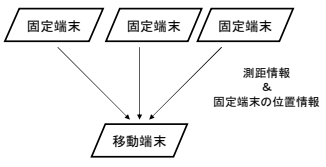
- 前提
  - 移動体の移動範囲内には、予め位置が既知であるセンサノードが複数配置されている(一つの空間に数個)
- 固定端末の座標と固定端末-移動端末間の距離から幾何学的に移動端末の座標を導出する



3

応用例1:屋内ナビゲーションシステム

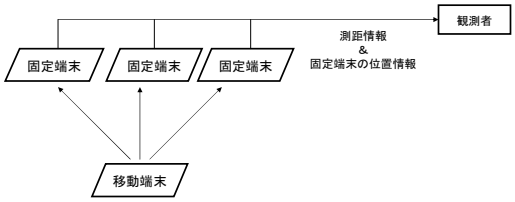
- 屋内におけるGPSの代替
- 地下街、キャンパス等のナビゲーション
- 位置が既知である固定端末と移動体間の距離から
  移動体の位置を導出する



4

応用例2:屋内トラッキングシステム

- 屋内の移動体の速度・位置を遠隔地から把握
- 移動体には、主に、
  ロボットの監視や物品管理を想定している



5

デモ

- ナビゲーション型の測位アプリケーション
- 固定端末を1m×1mの正方形状に配置し、
  その領域付近での移動端末の位置を導出する

6

Jini を用いたセンサネットワークシステムの動的な構築手法

植田 裕規 †

† 立命館大学大学院理工学研究科

1 研究背景

近年，無線デバイスの普及やコンピュータの小型化，低コスト化が進んでいる．これらの技術，および周囲の環境情報を取得するセンシング技術を利用したセンサノードが開発されている．現在，センサノード間の無線通信による広範囲なセンシングを実現するセンサネットワークシステムに関する様々な研究が進められている．

既存のセンサネットワークシステムにおいて，複数のセンサネットワークを管理することは困難である．すなわち，各センサネットワークの状態，機能，場所といった情報管理の問題から，複数のセンサネットワークの動的な構成変化に対応させることは困難である．また，既存のセンサノードとセンサノード用オペレーティングシステムには様々なものがあり，世界中で研究，開発されている．したがって，センサネットワークには異なる種類のものが多数あり，これらを一元管理することは困難である．このような問題を解決するために，Jini を用いたゲートウェイのセンサネットワークシステムへの適用を提案する [?].

本研究は，センサネットワークに Jini を用いたゲートウェイを提供することで，インターネットを経由し外部から複数のセンサネットワークの情報を取得する仕組みを実現し，外部からのリモートアクセスや複数のセンサネットワークの動的な管理を容易にすることを目的としている．これにより，複数のセンサネットワークから構成される柔軟性の高いシステムを構築することが可能となる．

2 次世代型斜面防災情報システムへの適用

現在，我々は，次世代型斜面防災システムのためのセンサネットワークシステムの開発を進めている．このシステムは，斜面に複数の種類のセンサを設置し，斜面の状態をリアルタイムに観測し，事前に土砂崩れの危険性を察知しようとするものである (スライド資料 3 枚目を参照)．このシステムに対し，スライド資料 6 枚目に示すように Jini を用いたゲートウェイを適用することにより，次のことが可能となる．

Lookup Service によるセンサネットワークの管理
: システム管理者は，各センサネットワークの機能をサービスとして Lookup Service に登録することで，稼動中のシステムに対して動的にセンサネットワークを追加・削除することが可能であり，新たな観測地の追加・削除という要求を満たす．ユーザは，これらの追加・削除に対して意識することなくセンサネットワークシステムシステムを利用することができる．

センサネットワークの情報を取得
: システム管理者は，サービスとして登録した各センサネットワークの情報を予め定義された形式に従って記述することで，Lookup Sevice に登録することが可能であり，ユーザに対してセンサネットワークの情報を動的に提供することができる．

インターネットを介したリモートアクセス
: ユーザは，ゲートウェイとなる Lookup Service を経由することで，インターネットを介して各センサネットワークに対してリモートアクセスが可能になる．これにより，外部からセンシングデータを取得することが容易になる．

3 デモンストレーション

本発表では，提案手法に基づいてプロトタイプシステムの実装を行いデモンストレーションを行った．本システムは，2 台のノート PC をスイッチングハブで接続し，片方の PC をセンサネットワークの機能を利用するクライアントとして用い，もう一方の PC をセンサネットワークが接続されたサービスプロバイダと Jini の機構を持つゲートウェイとして用いた．このような構成において，クライアントにより LAN を介したセンサネットワークへのリモートアクセスを行い，センシングデータを取得可能であることを示した．なお，デモンストレーションシステムの構成は，スライド資料の 7 枚目と 8 枚目に詳細を示す．

デモンストレーションは，次に示す手順で行った．

1. ゲートウェイとなる PC 上で RMID，クラスサーバ，Lookup サーバを起動させる．これにより，Jini を利用できる環境を構築する．
2. ゲートウェイとは異なる PC 上でクライアントを起動させ，センサネットワークがゲートウェイに対して登録されていないことを確認する (スライド資料 9 枚目の左参照)．
3. ゲートウェイを起動させた PC 上でセンサネットワークの機能を提供するサービスプロバイダを起動させ，ゲートウェイに対してサービスとして登録する．この際に，クライアントにセンサネットワークの情報が出力されることを確認し，動的にセンサネットワークを追加できること示す (スライド資料 9 枚目の右参照)．
4. クライアントからゲートウェイに登録されたセンサネットワークの機能を呼び出し，リアルタイムにセンシングデータを取得する様子を確認する (スライド資料 10, 11 枚目を参照)．

4 おわりに

本稿では，インターネットを介したリモートアクセスや複数のセンサネットワークを管理するために，Jini の機構を用いたゲートウェイを適用することで動的にセンサネットワークシステムを構築する手法について述べ，次世代型斜面防災情報システムを用いた適用例について述べた．また，本システムのデモンストレーションを行った．

今後の課題として，システムとしての汎用性を向上させるために，ゲートウェイに登録するセンサネットワークの情報の厳密な定義やセンサネットワークの機能を利用するためのインタフェースの形式化が挙げられる．

参考文献

[1] 植田 裕規，鈴来 和久，横田 裕介，大久保 英嗣: センサネットワークにおける Jini による柔軟なゲートウェイサービスの構築，第 69 回情報処理学会全国大会予稿集，第 3 分冊，6V-5，2007.

Jiniを用いたセンサネットワークシステムの動的な構築手法

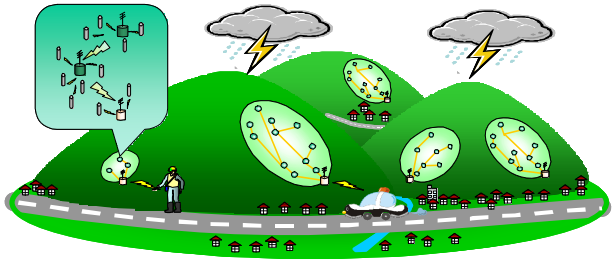
立命館大学
理工学研究科　情報理工学専攻
基本ソフトウェア研究室
M1　植田 裕規

目次

- 次世代型斜面防災情報システム
- 研究背景
  - 既存センサネットワークシステムの問題点
  - Jiniを適用する利点
- 提案手法の全体像・適用例
  - 次世代型斜面防災システムへの適用
- 本日のデモプログラム
  - デモプログラムの流れ
  - デモシステムの構成

2

次世代型斜面防災情報システム



：ワイヤレスセンサーとインテリジェントロガーからなる計測ポイント

：ワイヤレスセンサー

：中継局（インテリジェントロガーによるデータの収録および分析・センサー設定の遠隔操作）

：基地局（インテリジェントロガーによるデータ受信・データ保存・発信・センサー設定の遠隔操作）

3

研究背景

- 既存センサネットワークシステムの問題点
  - センサネットワークごとにセンサネットワークの機能や設置場所が異なるため，これらの情報の管理が問題
    - 動的な構成の変化に対応することが困難
  - センサネットワークごとに所持する機能が異なるため，クエリの仕様が異なる
    - 異種センサネットワークの混在が困難

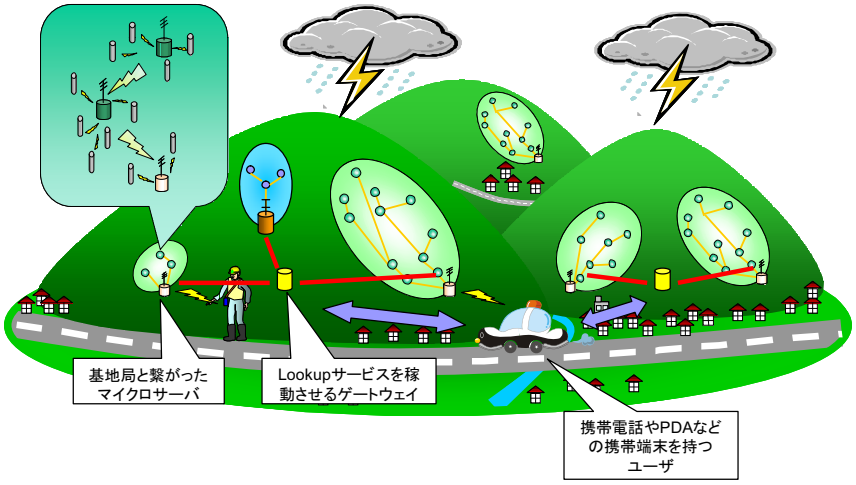
4

Jiniを利用する利点

- Lookup serviceによるセンサネットワークの管理
  - 各センサネットワークをサービスとして動的に管理する
- JiniのAttribute（属性）の利用
  - センサネットワークの情報やクエリの仕様を表すオブジェクトをAttributeとして登録できる
- インターネットを介したリモートアクセス
  - Lookup serviceを通して，外部からセンサネットワークを遠隔操作できる
- 開発コストの削減，組込み機器への搭載 etc.

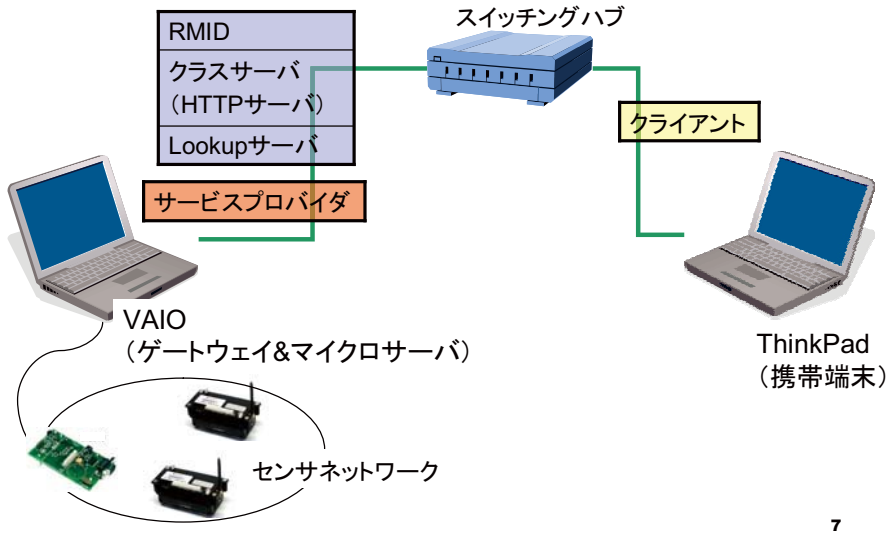
5

次世代型斜面防災情報システムへの適用



6

本日のデモプログラム



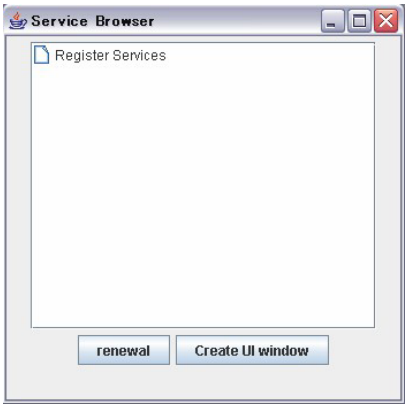
7

デモシステムの構成

- ノートPC2台（VAIO-TX90S, ThinkPad-T60)
- OS：Windows XP home, pro SP2
- 基地局：MIB510
- センサノード：MPR2400J+MTS310
- TinyOS v1.13 + TinyDB v1.1.14.5
- Java JDK1.5.12
- Jini v1.2.1

8

デモ：センサネットワークの情報取得

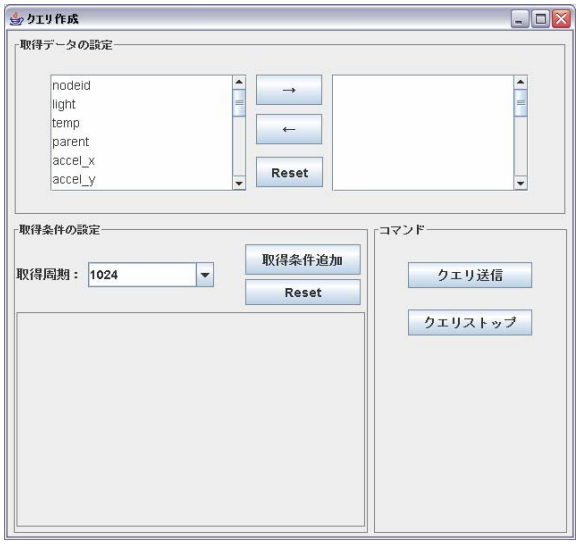


センサネットワーク登録前

センサネットワーク登録後

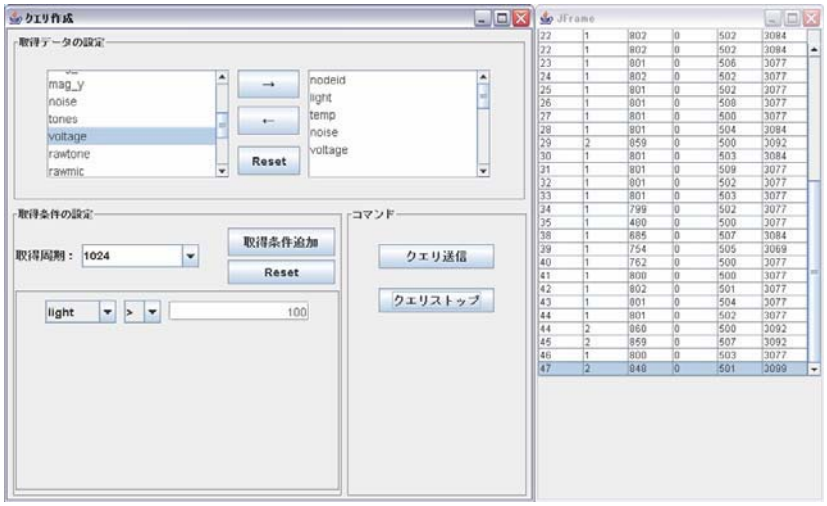
9

デモ：クエリ作成のウィンドウ取得



10

デモ：センシングデータの取得



11

おわりに

- 研究背景と概要
  - 次世代型斜面防災情報システム
  - 既存センサネットワークシステムの問題点
    - 動的な構成の変化への対応が困難
    - 異種センサネットワークの混在が困難
  - Jiniを用いたゲートウェイを適用する利点
    - 複数のセンサネットワークを動的に管理
    - リモートアクセスが容易に実現可能
- 本日のデモについて
  - デモプログラムの概要
    - 2台のPCを利用したセンサネットワークへのリモートアクセス
  - デモシステムの構成

12

データを蓄積したセンサノードに対する時制を用いた問い合わせのデモンストレーション

冨森 英生 †
† 立命館大学大学院理工学研究科

センサネットワークを用いて気温など環境情報を取得する場合, 変化するユーザの要求に対応するために問合せ処理システムを用いる. センサノードは電池で動作するため, 消費電力の大きい無線通信を削減する必要がある. 通信を削減するには, ノードがデータを選択し, ユーザが必要とするデータのみ送信する機能が重要になる.

既存の問合せ処理システムでは, ノードは過去に取得したセンサデータをノード上に保存しない. 例として, ノードが取得した過去の平均気温と現在の気温を比較する目的でセンサネットワークを用いる場合, 各ノードは周期的に取得した気温データをすべてホストに送信し, ホスト上で過去の気温の平均を求める. ユーザの要求を満たすために必要なデータは過去の平均気温という集約したデータであるが, 集約するために必要なデータを全てノードからホストに送信する必要があるため, 集約演算が通信の削減に寄与しないという課題がある.

この課題を解決するため, ノードのメモリ上に過去に取得したセンサデータを格納し, ノード上でデータを集約しホストに送信する手法 [1] を提案する. 本手法では, ユーザは 2 種類の問合せを発行する.

1. ユーザは各ノードのメモリ上にテーブルを作成するデータ蓄積クエリを発行する. クエリで指定された格納するセンサデータの種類と首取得周期, テーブルの大きさに従い, ノードはセンサデータを取得し作成したテーブルに格納する.
2. データを蓄積したノードに向けてデータ回収クエリを発行する. クエリには集約対象となるデータの範囲が時刻や期間により指定され, ノードはテーブルに格納されたデータのうち指定された範囲のデータを集約し演算結果のみホストに送信する.

本手法では, ノードが取得したセンサデータはノードのテーブルに格納され, 集約演算の結果のみホストに送信する. ただし, センサノード向けアプリケーションは, 搭載するメモリ容量が少ないことを考慮して作成する必要がある. このため, ノードが保持するデータは必要なデータのみに制限される. 過去のデータを用いて送信するデータの選択をノード上で行うことにより, 無線通信の削減による消費電力の抑制が可能になる. また, センサノードに格納されるデータには, 取得時刻のタイムスタンプを付与するものとする. 処理対象となるデータの指定を時刻や期間を用いて容易に記述可能にするには, 問合せ言語で時制を扱う必要がある.

提案手法の実現手法として, 既存の問合せ処理システムである TinyDB [2] を拡張する. 拡張する必要のある機能には, 次のものが挙げられる.

- ノード上でのテーブルの作成, データの格納.
- テーブルに格納したデータの集約演算.
- 問合せ言語 (TinySQL) の拡張.

上記のうち, テーブルの作成とデータの格納機能の拡張は完了している. 現在, 集約演算を行うための機能を実装している. 本デモンストレーションでは, シミュレータ上で次の問合せを行いノードが動作する様子を示した.

1. ホストからデータ蓄積クエリを発行し, ノード上に作成した気温データを格納する (スライド 11).
2. テーブルに格納したデータに対しデータ回収クエリを発行し, ノードで集約演算を行い結果のみホストに送信する (スライド 12).

問合せは, TinySQL に SQL/Temporal などを参考に時制拡張を加えた言語を用いて記述する. ただし, 現状では問合せ言語の拡張は完了していないため, 問合せ言語をコンパイルして得られるデータを事前に用意し, ノードに送信することで問合せを実現した.

今後は, TinySQL に対する時制拡張を行い, 拡張した言語による問合せ記述を可能とする.

参考文献

[1] 冨森英生, 鈴来和久, 横田裕介, 大久保英嗣：時間指定可能なクエリ処理システム, 情報処理学会第 69 回全国大会, 第 3 分冊, 6V-8 (2007).

[2] Madden, S., Franklin, M., Hellerstein, J. and Hong, W.: TinyDB: an acquisitional query processing system for sensor networks, *ACM Transactions on Database Systems (TODS)*, Vol. 30, No. 1, pp. 122–173 (2005).

データを蓄積したセンサノードに対する時制を用いた問い合わせのデモンストレーション

立命館大学大学院 理工学研究科
基本ソフトウェア研究室
M1 冨森英生

1

背景

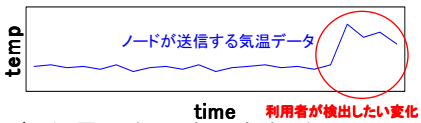
- センサデータの取得を問合せ処理で行う
  - ユーザはデータの種類，取得周期を指定
  - ノードはセンサデータを取得し，ホストに送信する
- ノードの電力消費を抑制する必要がある
  - 電力消費の大きい無線通信を削減

ノード上で必要なデータのみ選択・生成し送信する機能の重要性

2

既存技術の課題

- 集約演算を用いたデータ量の削減
  - 過去1時間の平均気温と比較して大きな変化があった場合のみ気温データを取得



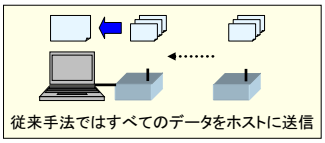
- 従来手法
  - 過去のセンサデータ(履歴)をノード上に保存しない
  - すべてのデータをホストに送信し，ホスト上で集約演算を行う必要あり

集約演算の利用が通信回数の削減に寄与しない

3

履歴データと集約演算を用いた通信回数の削減

- 履歴データを用いた集約演算・比較演算のノード上での実行
  - 過去に取得したデータとの比較
  - 平均値などのデータ集約
- 演算結果のみの送信による通信回数の削減
- 問合せ言語において履歴データと時制を扱う必要



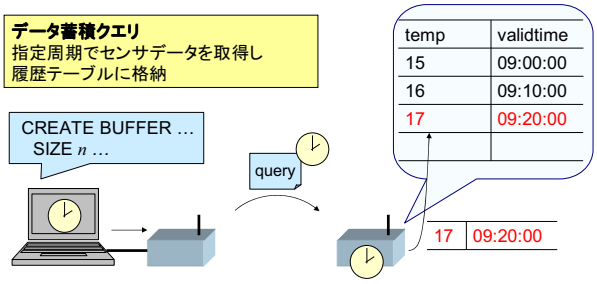
4

提案手法

- データを蓄積したセンサノードに対する時制を用いたクエリ処理システムの実現
- 過去に取得したデータに対する問合せを行う
  - 過去のデータとの比較
  - ノード上でのデータ集約
- 時刻，期間を用いた問合せ記述
  - センサデータに関連付けられたデータの取得時刻を容易に処理可能
- TinyDBを基に拡張を進めている
  - 問合せ言語はSQL/Temporalなどを参考して拡張

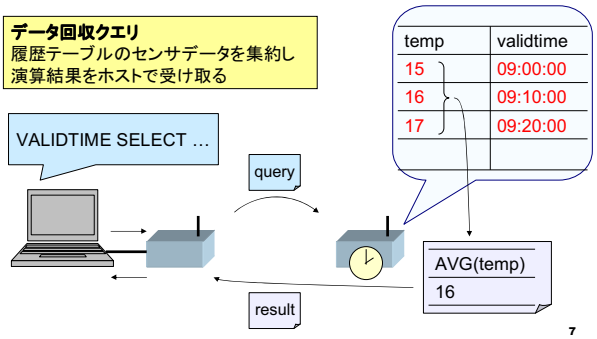
5

履歴テーブルの作成



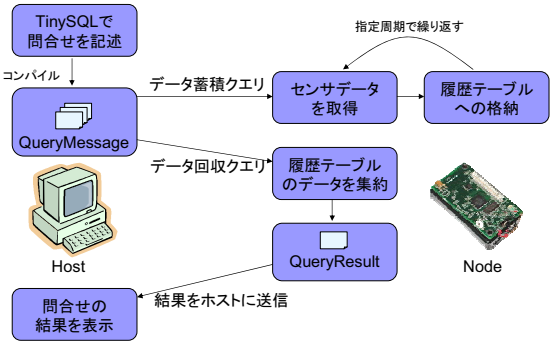
6

時制問合せ



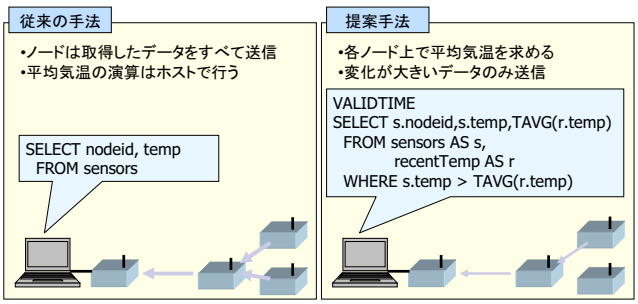
7

問合せ処理



8

適用例



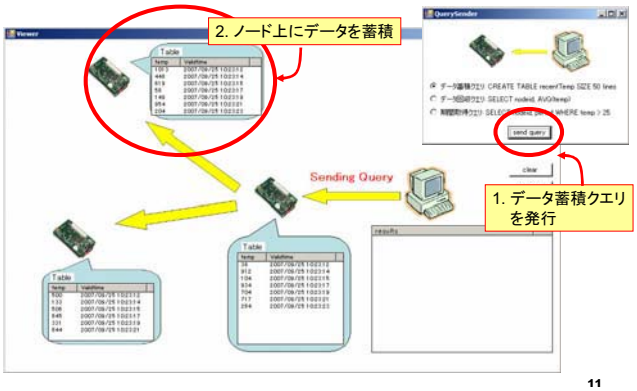
9

デモンストレーション

- クエリの発行, 処理結果の取得
  - **データ蓄積クエリ**: ノードは気温データを取得し各々の履歴テーブルに格納
  - **データ回収クエリ**: 各ノードが蓄積した気温データの平均値を求めホストで取得
  - **期間取得クエリ**: 気温 > x を満たす期間を各ノードが求めホストで取得
- シミュレータを利用

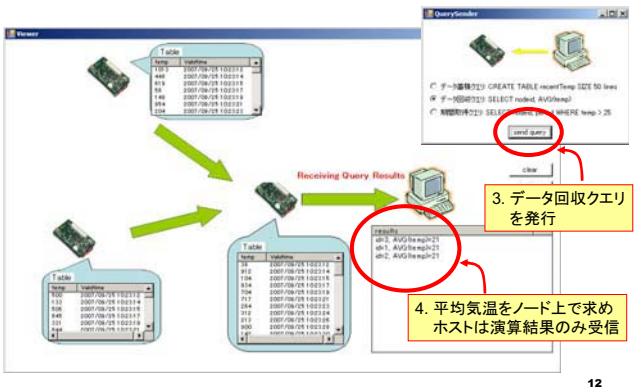
10

デモ: データ蓄積クエリ



11

デモ: データ回収クエリ



12

Tender における OS 動作の可視化のための解析と表示手法

木下 彰† 田端 利宏‡ 谷口 秀夫‡

†岡山大学工学部 ‡岡山大学大学院自然科学研究科

1. はじめに

サービスの多様化や要求の高度化により，ソフトウェアの規模は大きくなり，その構造を把握するのは難しい．このため，処理の実際の流れを理解するのが難しく，不具合の発生時，不具合箇所の特定が難しい．特に，オペレーティングシステム(OS)のように処理の流れが複雑なソフトウェアでは，これらは非常に大きな問題である．

そこで，***Tender*** における OS 動作の学習とプログラム開発支援を主目的とした OS の動作の可視化機構に関して，情報収集機構の概要と，表示部について述べる．

2. *Tender* の可視化機能[1]

Tender では，操作する対象を資源として分離，独立化しており，独立化した資源処理をプログラム部品と呼ぶ．プログラム部品の呼出は，必ず資源インタフェース制御(RIC)を経由して行う．このとき，各プログラム部品の呼出前と呼出後に必要な情報を記録することで，RIC において可視化情報の収集が可能になる．収集した情報は，動作記録表と呼ぶデータ構造に記録する．

また，流れ識別子を用いることで，割込やタイムスライスといった処理開始契機と処理を関連付けて記録する．このようにすることで，異なる契機による処理を区別して記録できる．

3. 可視化表示部

表示部は，処理契機を考慮した動作の流れを理解しやすい形で提供することを目標とする．可視化を行うにあたり，最初に ***Tender*** 上で動作記録を行い，次に記録した情報をバイナリファイルに書き出す．このファイルを可視化アプリケーション実行時に読み込んで解析し，SVG 画像を XML 形式の文書として出力する．以降，収集情報の解析手法と表示手法について述べる．

表示部では，流れ識別子の情報を基にして，処理契機の移り変わりを処理の流れとして表す．このようにすることで，処理契機に違いによる処理の流れを明確化できる．しかし，これだけでは OS 処理とプロセスの動きとを区別できず，実際の動作の流れを把握できない．そこで，各処理の呼出，復帰時刻を考慮して収集情報を解析することで，OS 処理とプロセスの動きを区別して表示する．

この手法を用いた例として，プロセススケジューリングの動作を可視化した様子を図 1 に示す．可視化図の表示を行うにあたり，用途に合わせて時間軸 1 目盛りあたりの時間，可視化図の表示範囲を指定できる．

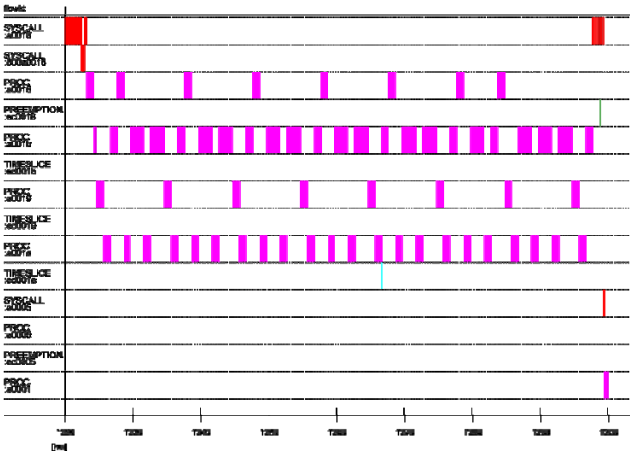


図 1 動作の可視化例

Tender では，プロセッサの割り当て単位として資源「演算」を導入しており，演算を持つプロセスだけがスケジューリング対象となる．図 1 の例では，3 つのプロセスを生成し，各プロセスにそれぞれ 10%，30%，50%の性能を割り当てて走行させている．以上より，図 1 において，各プロセスに割り当てた性能に基づいた処理の様子を表せていることがわかる．

4. おわりに

Tender における収集情報の解析と表示手法について述べた．今後は，プロセススケジューリング以外の OS 機能の可視化手法について，検討する．

参考文献

[1] 木下彰，河原太介，田端利宏，谷口秀夫：***Tender*** における OS 動作の可視化のための情報収集と表示の方式，情報処理学会研究報告，2007－OS－105，Vol.2007，No.36，pp.31－38，2007.

Method of Analysis and Display for Visualization of Operating System Behavior in ***Tender***

Akira KINOSHITA†, Toshihiro TABATA‡ and Hideo TANIGUCHI‡

† Faculty of Engineering, Okayama University

‡ Graduate School of Natural Science and Technology, Okayama University

*Tender*における
OS動作の可視化のための解析と表示手法

木下彰†　田端利宏‡　谷口秀夫‡

†岡山大学工学部
‡岡山大学大学院自然科学研究科

目次

No.2

はじめに

＜研究背景＞

ソフトウェアの動作状況を見ることができない

⇩

(問題1) 処理の実際の流れを把握することは難しい

(問題2) 不具合発生箇所の特定が難しい

OSのように処理の流れが複雑なソフトウェアでは大きな問題

⇩ 動作状況の可視化で対処

(1)状態情報：資源(プロセッサ，メモリetc)の現在の利用状況

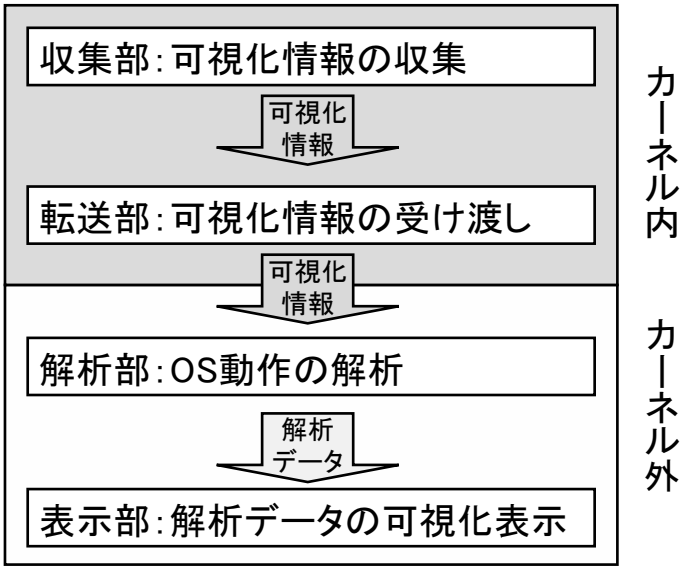
(2)動作情報：プログラムの時系列の実行状態

⇨ OSの処理の実際の流れを理解するためには
特に割込を考慮した動作情報の把握が重要

Tender に割込を考慮したOS動作の可視化のための
収集情報の解析と表示手法を実現

No.3

可視化機能の基本構造



No.4

可視化の目的

(1) OS動作の学習支援 (表示部)

(a)処理の流れやシステム状態の情報をよりわかりやすく提供

⇨ OS内部の処理を考慮した処理全体の流れを表示

└{ 割込処理，例外処理，プロセス切り替え

(b)学習目的により表示方法を変化する必要性

(2) OSプログラムの開発支援 (解析部)

動作試験や不具合発生時の動作確認

⇨ 不具合発生箇所や処理遅延箇所の特定，動作確認

(1) 処理の呼出関係の把握

(2) 各処理に要する時間の把握

No.5

OSプログラム構造

OSの動作情報を把握するためには

(対処1) OSの機能ごとにプログラムを分割

⇨ 資源(OSが操作する対象)を分離し，独立化して管理

独立化された資源処理をプログラム部品と呼ぶ

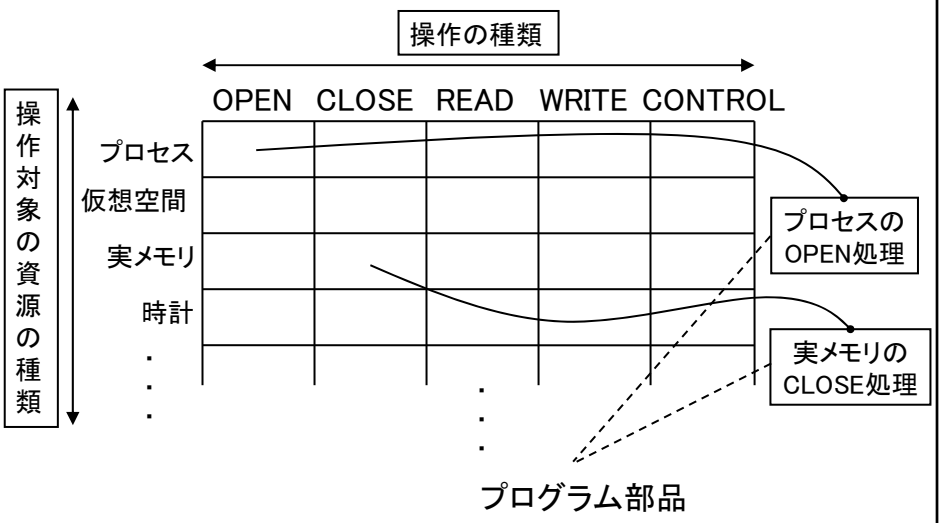
(対処2) プログラム部品の呼出を把握

⇨ 資源インタフェース制御(RIC)によるプログラム呼出把握

“操作対象の資源の種類”と”操作の種類”から構成される
プログラムポインタ表によりプログラム部品を管理

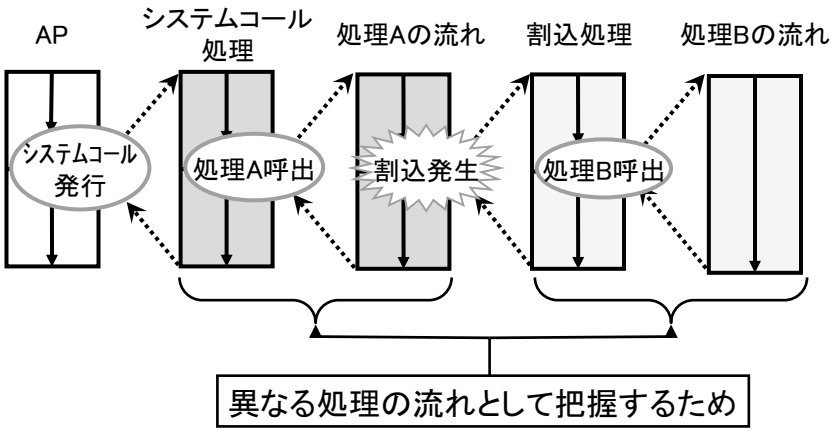
No.6

プログラムポインタ表



No.7

割込を考慮した処理の流れの把握法



異なる処理の流れとして把握するため

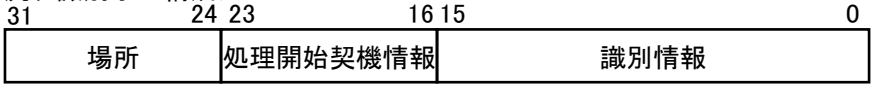
(1) 処理開始契機を分類
(2) 処理と開始契機を関連付けて記録

No.8

処理開始契機の分類

流れ識別子(flowid)を用いて処理と開始契機を関連付けて記録

＜流れ識別子の構成＞



＜処理開始契機情報と識別情報＞

処理開始契機情報	識別情報
システムコール発行	処理を要求したプロセスのプロセス識別子の通番
割込	割込ベクタ番号
例外	例外ベクタ番号
タイムスライス	発生時のプロセスのプロセス識別子の通番
プリエンプション	発生時のプロセスのプロセス識別子の通番

⇒ (1)異なる開始契機を持つ処理を区別し，割込を考慮した処理の流れを明確化可能
(2)処理の流れとOSの機能との関連を容易に把握可能

No.9

収集する可視化情報

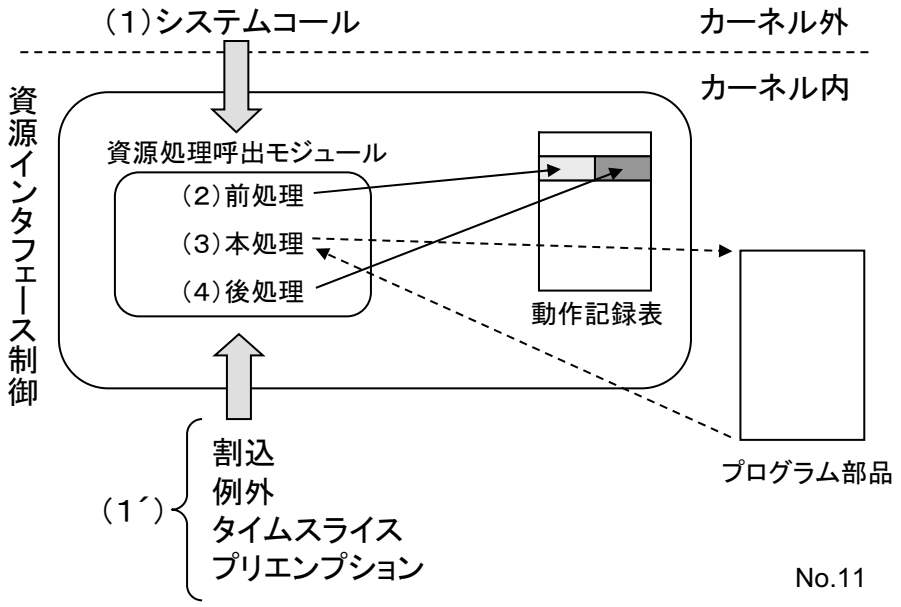
可視化に必要な情報	項目名	内容
(1)処理の要求先を示す情報	dest_rid	操作対象の資源識別子
(2)処理内容に関する情報	operate	操作の種類
(3)処理結果情報	ret_val	戻り値
(4)処理開始契機情報	flowid	流れ識別子
(5)処理開始時刻	call_k_time	呼出時のカーネル時刻
(6)処理終了時刻	ret_k_time	復帰時のカーネル時刻

※処理の要求元を示す情報は
解析部において記録済みの項目を検索

(A)上記の項目を1エントリとして動作記録表に記録
(B)エントリ番号を処理番号と呼ぶ

No.10

情報収集機構の構造



No.11

表示部の基本構造

割込を考慮した動作の流れを理解しやすい形で提供する

<収集情報の問題点>

(1) 可視化する内容が複雑になる

(2) 情報量が多くなり，処理全体の流れを把握しにくい

必要な情報のみを抜き出して理解しやすい形で表示を行う

flowidの情報を基にして
処理開始契機を考慮した処理の流れを表示

以下のようなことが視覚的に把握できる

(1) 各処理の開始契機，処理時間，処理順序

(2) 割込，例外処理の発生時刻，処理量

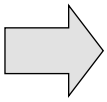
No.12

可視化のための収集情報の解析

flowidを基にした動作の可視化

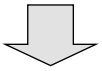
(1) flowidが同じ処理は同一処理である

(2) flowidの変化により処理の移り変わりを判断し，処理の流れを示す



<問題点>

flowid変化時の時刻のみを取っているため
OS処理とプロセスの動きを区別できない



(対処) 同flowid内における資源処理の呼出，復帰時刻を考慮することでOS処理とプロセスの動きを分割して表示

No.13

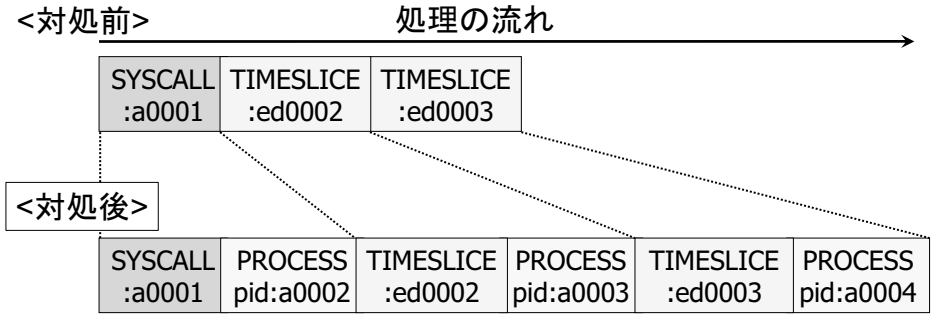
動作記録表

id	src_id	dest_rid	rsc_kind	operate	ret_val	flowid	call_k_time	ret_k_time
607	-1	80000	EXEC	CONTROL	8001a	ed001a	88f4fe452	88f5aac62
608	607	60001	COUNTER	READ	0	ed001a	88f4fe845	88f4fe982
609	607	a001b	PROC	CONTROL	0	ed001a	88f4feb5e	88f4ff301
610	609	d001a	VMEM	WRITE	1a	ed001a	88f4fecf5	88f4ff208
611	610	b0122	VREGION	CONTROL	1547000	ed001a	88f4feec2	88f4ff00c
612	607	60001	COUNTER	READ	0	ed001a	88f4ff498	88f4ff5de
613	-1	80000	EXEC	CONTROL	8001b	ed001b	88f5a97e3	88f6005fa
614	613	60001	COUNTER	READ	0	ed001b	88f5a9b9a	88f5a9cce
615	613	a001a	PROC	CONTROL	0	ed001b	88f5a9ea8	88f5aa64b
616	615	d0019	VMEM	WRITE	19	ed001b	88f5aa04e	88f5aa552

No.14

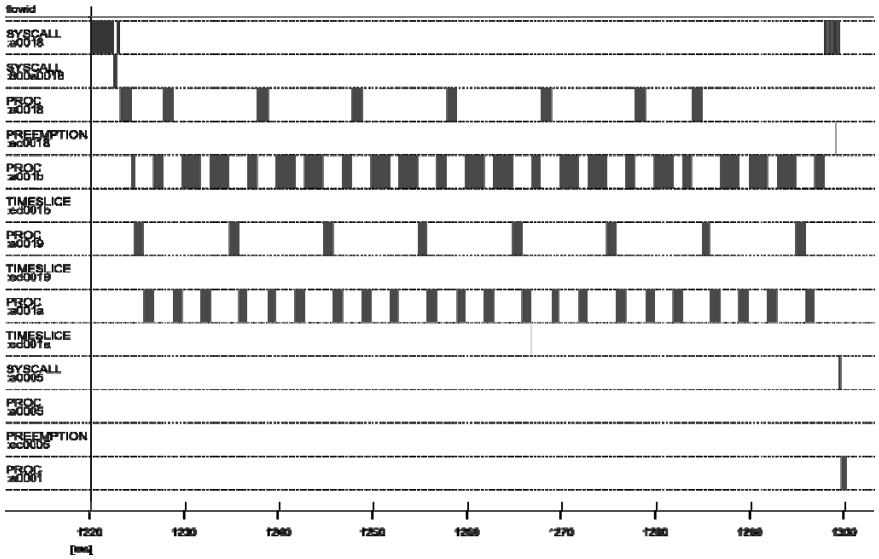
表示方法

処理開始契機が 割込，例外 でないとき
識別情報は直前に走行していたプロセスのプロセス識別子の通番
⇨ OS処理のflowidの通番は直前に走行しているプロセスの識別子の通番である
⇨ OS処理とプロセスの動きを分割して表示



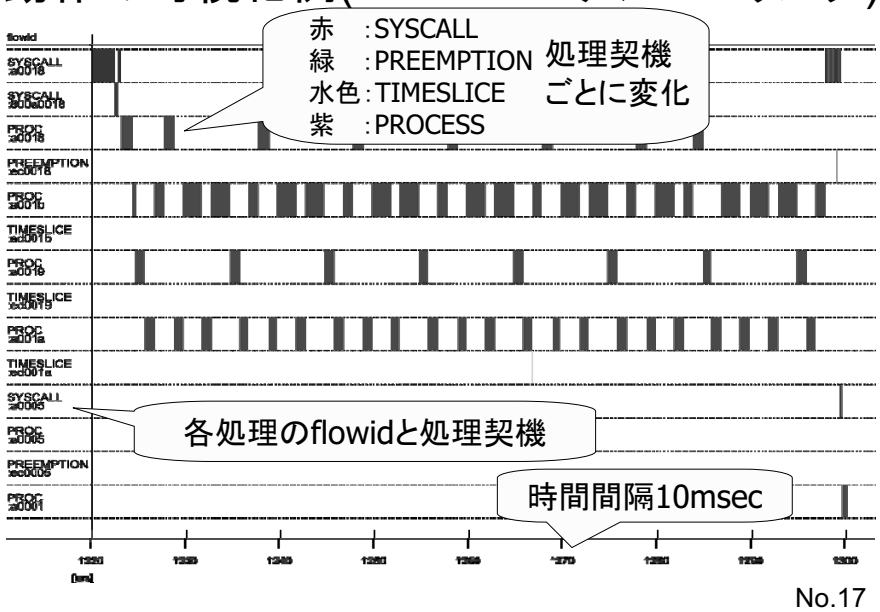
No.15

動作の可視化例(プロセススケジューリング)



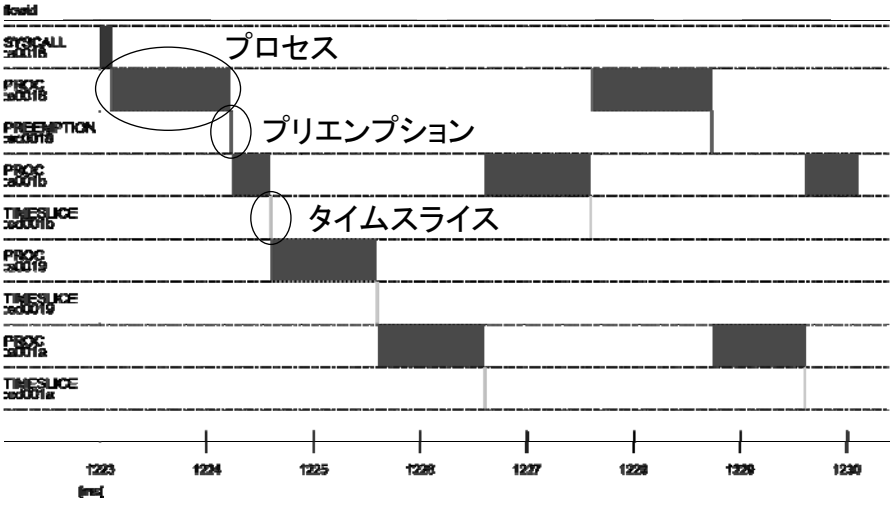
No.16

動作の可視化例(プロセススケジューリング)



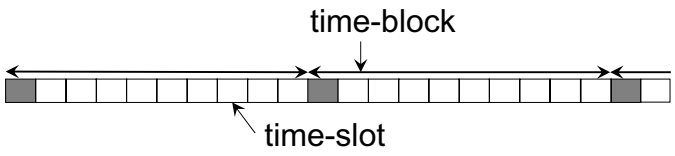
No.17

動作の可視化例(プロセススケジューリング)



No.18

実行速度調整機能



(1)資源「演算」によりプロセスにプロセッサを割り当てる
(2)タイムブロックとタイムスロットを用いて実現されている

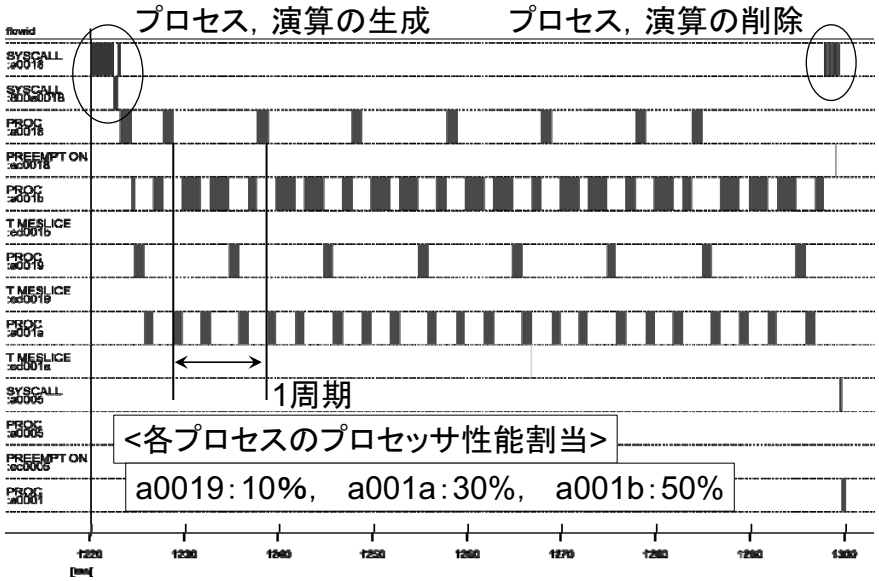
タイムブロック: プロセッサの処理時間を一定時間(1秒)ごとに区切ったもの

タイムスロット: タイムブロックをある一定の時間(1ミリ秒)でさらに区切ったもの

タイムブロック中のタイムスロットを指定された性能(%)の割合分だけ演算に割当てる

No.19

動作の可視化例(プロセススケジューリング)



No.20

おわりに

■流れ識別子を基にして表示部を実現
(1) 処理の流れを表示する機構を実現した
(2) OS処理とプロセスとを区別して表示する
(3) 動作の可視化例を示し，OS処理を含めた動作の可視化が行えることを示した

＜今後の課題＞
(1) 学習目的，および目的に合わせた表示方法の検討
(2) 転送部，解析部を含めた可視化機能全体の構築
(3) 資源呼出を基本とした詳細な動作の可視化方法の検討

No.21

仮想計算機モニタを利用したゲスト OS の入出力要求監視手法

金城 聖 †
† 立命館大学大学院理工学研究科

1 はじめに

OS の動作状況観測手法は，OS やアプリケーションの開発やデバッグ，性能評価などに使用されてきた．従来の OS 動作状況観測手法は，コンパイラと連動して OS やプログラムにデバッグ用スタブを埋め込んだり，アプリケーションから専用のシステムコールを利用することで実現している．このような制御は，観測・評価対象である OS が実行している．そのため，性能評価アプリケーションやデバッガなどが，観測によって生じる OS の動作に与える影響を無視することができず，詳細な観測や性能評価，デバッグを行うことが困難である．このような問題を解決する上で提案されたのが，計算機上にあるハードウェアの機能を利用した観測手法である．ハードウェアによる観測では，ソフトウェアが介在しないため，OS の動作に影響を与えずに観測することが可能である．しかし，CPU の処理速度と比較して，ハードウェアが使用する通信路の速度や仕様によって性能が低下するため，詳細な観測を行うのは困難である．

また，近年の仮想化技術の普及によって，VMware や Virtual PC などの安価な仮想計算機環境が登場した．また，Xen[1] に代表されるオープンソースの仮想計算機モニタ (以下，VMM と呼ぶ) の登場により，仮想計算機環境の実現コストが大幅に低下した．また，CPU のマルチコア化や大容量メモリの低価格化によって計算機資源が潤沢になり，OS の動作環境として仮想計算機モニタを仮想計算機環境が注目されるようになった．

このような背景から，VMM を利用した OS の動作状況観測手法 [2] の研究を行っている．既存の観測手法には，観測対象の内部に観測機構を実装することで観測を行う手法や，ハードウェアの機能を利用して観測を行う手法がある．このような手法は，観測対象に影響を与えたり，観測範囲を制限してしまうなどの問題がある．そこで観測機構を OS とハードウェアの間に設けるために VMM を利用する．VMM は，ハードウェアを仮想化して，OS に提供するソフトウェアである．そのため，VMM に観測機構を実装することで，OS の動作に影響を与えることなく，OS の動作の観測を行うことが可能となる．

以下，本稿では，VMM を利用したゲスト OS の I/O 要求監視手法について述べる．2 章で VMM を利用した動作状況観測の概要ついて述べ，3 章で Xen の I/O 仮想化手法について述べる．4 章で I/O 要求監視手法と今後の課題について述べ，5 章で本稿をまとめる．

2 VMM を利用した動作観測

従来の動作観測は，観測対象 OS の内部に観測機構を実装したり，専用のハードウェアを利用したりして，実現されていたが，本手法では，VMM を利用して実現する．VMM は，計算機資源を仮想化し，OS における計算機資源の必要する処理を仮想化された計算機資源を使用することで，複数の OS を同時に動作させるためのソフトウェアである．この特徴を利用し，ゲスト OS の動作を監視する．ゲスト OS の計算機資源の利用要求の処理は，VMM を行うため，VMM 内部に観測機構を実装することで，ゲスト OS の動作の観測が可能となる．

3 Xen における I/O 仮想化

本稿では，I/O 要求の監視手法について述べる．I/O 要求監視手法の検討をする上で，まず，既存の VMM における I/O 仮想化の仕組みについて調査を行った．以下では，既存の VMM である Xen の I/O 仮想化について述べる．

Xen における I/O の仮想化は，

- 共有メモリ領域
  ゲストドメイン，特権ドメイン間の I/O 要求と応答のやりとりを行うためのメモリ空間．
- Event Channel
  I/O 要求と応答の共有メモリへの書き込みを通知するための通信路．
- Device Model
  特権ドメイン内部にあるゲストドメインからの I/O 要求の処理を行うデバイスエミュレーション機構．

以上の 3 つの機構によって実現される．システム構成図を図 1 に示す．

Xen における I/O の仮想化は，以下の手順によって実現される．

1. ゲストドメイン上の OS が I/O 要求を発行すると，制御が Xen に移行し，Xen がゲストドメインに提供している仮想 I/O デバイスが要求を確認する．
2. Xen が仮想 I/O デバイスに対して発行された I/O 要求を共有メモリに書き込む．
3. 要求を共有メモリに書き込んだ後，Event Channel を経由して，特権ドメインに要求の書き込み終了を通知する．

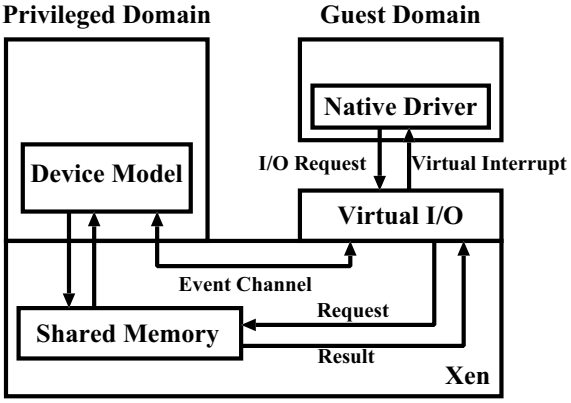


図 1　**共有メモリを利用した I/O の仮想化**

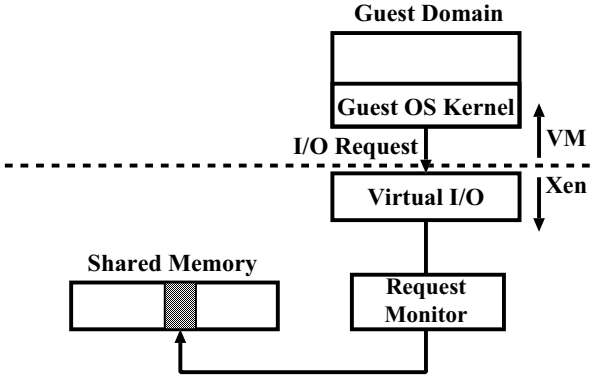


図 2　**共有メモリを利用した I/O の仮想化**

4. 通知を確認後，特権ドメインは共有メモリ上の要求を取り出し，Device Model によって処理を行う．
5. 特権ドメインは，処理結果を共有メモリに書き込む．
6. 書き込み後，Event Channel を経由して，ゲストドメインに結果の書き込み終了を通知する．
7. ゲストドメインは，通知を確認後，結果を共有メモリから取り出し，ゲスト OS に対して割込みを発生させ，処理を完了する．

上で述べたように，I/O デバイスの仮想化は要求と応答の通信によって実現される．

4　I/O 要求監視手法

4.1　提案手法

提案手法では，仮想 I/O デバイスが共有メモリ領域に要求を書き込む段階で監視する．ソフトウェア構成を図 2 に示す．

ゲスト OS の I/O 要求監視は，以下の手順によって実現する．

1. 仮想 I/O デバイスから I/O 要求が発行される．
2. 要求監視機構が発行された I/O 要求を記録し, その後共有メモリ領域に書き込む

I/O 要求監視機構に必要な機能としては，

- 要求の発行元を判別する機能
  ゲストドメインが複数動作している場合，どのドメインが要求を発行しているのかを区別し，ドメイン毎の監視を行う
- I/O 要求の解析機能
  どのデバイスへの要求なのかを判別することで，デバイス毎の監視を行う

が挙げられる．

4.2　今後の調査項目

現在，Xen のソースコードの解析による I/O 仮想化の調査を行っている．I/O 要求はゲストドメインとゲストドメインに対して提供されている仮想 CPU と関連がある．I/O 要求は，ゲストドメインの情報と仮想 CPU の情報から構成されているため，I/O 要求，ドメイン，仮想 CPU に関する構造体の調査と，Xen 内部で利用されている I/O 関連の操作の調査を行う必要がある．

5　おわりに

本稿では，VMM を利用した I/O 要求監視手法の提案と既存の VMM の I/O 仮想化の仕組みについて調査を行った．今後の課題としては，上記で述べた調査の完了と明確な監視モデルの検討が挙げられる．

参考文献

[1] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, Andrew Warfield : Xen and the Art of Virtualization, ACM Symposium on Operating Systems Principles, (2003).

[2] 金城 聖, 鈴来 和久, 毛利 公一 : “仮想計算機モニタを用いた OS 動作状況観測手法”, 第 69 回全国大会講演論文集, Vol.1，pp.117–118，情報処理学会, 2007.

仮想計算機モニタを利用したゲストOSの入出力要求監視手法

立命館大学 大学院 理工学研究科
毛利研究室
M1 金城 聖

発表内容

- はじめに
- 仮想計算機モニタを利用したOSの動作状況観測
- 仮想計算機モニタの調査
  - Xenの概要
  - XenのI/O仮想化
- I/O要求監視手法
- 進捗と調査項目
- おわりに

はじめに

- 研究背景
  - リアルタイムVMの研究
    - 仮想計算機モニタにおけるリアルタイム性保証
      - リアルタイム性の評価時において動作観測
  - システム開発効率の向上
    - オペレーティングシステムやアプリケーションの開発，デバッグ
    - システムの試験運用時における不具合の抽出
    - ハードウェアに不具合が発生した場合の動作のシミュレーション
- 仮想計算機モニタを用いたOSの動作観測手法

仮想計算機モニタを利用した動作観測

- 仮想計算機モニタを利用する利点
  - OSの動作環境構築が容易
    - 実計算機上での動作と同様の動作が可能
  - OSとハードウェア間に存在する計算機抽象化層の存在
    - 観測機構をOSの下の層に設置することが可能
    - OSの内部からではなく，外部からの観測することで，OSの動作に与える影響を最小限に
- 観測対象
  - ゲストOSのI/O，メモリアクセス，CPUの利用状況など
- ゲストOSのI/O要求の監視
  - 仮想計算機モニタXenをもとに実現

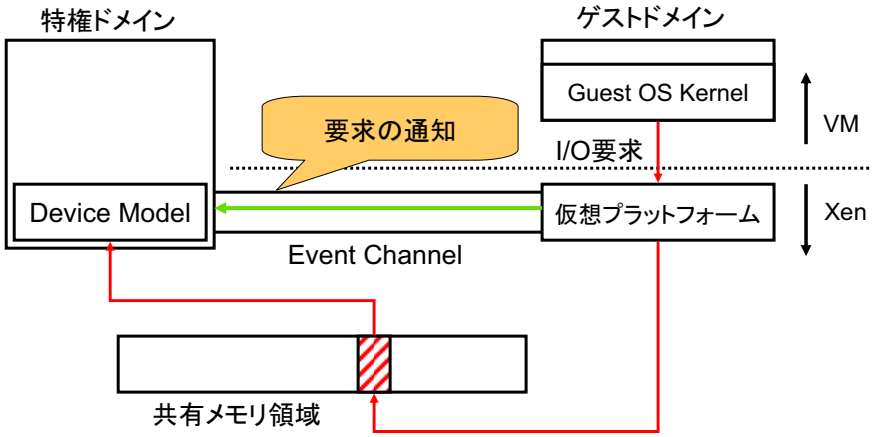
仮想計算機モニタの調査

- Xen 仮想計算機モニタ
  - オープンソースの仮想計算機モニタ
  - 計算機の完全な仮想化
  - 仮想計算機をドメインという単位で管理
  - 特権ドメインとゲストドメインで構成される
    - 特権ドメイン：ハイパーバイザコールを介してゲストドメインの生成・削除，起動・終了を行う管理用ドメイン
    - ゲストドメイン：ゲストOSが動作する仮想計算機．ゲストOSからは完全な計算機として認識
  - CPUの仮想化支援機構に対応

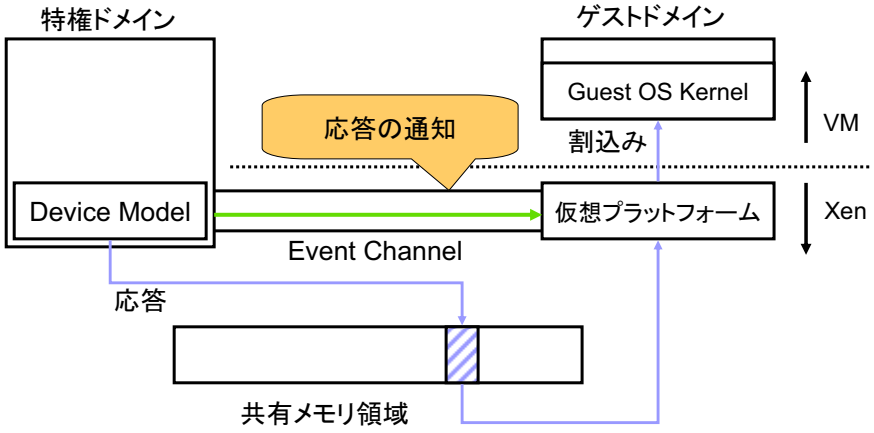
XenにおけるI/O仮想化(1/3)

- 共有メモリ領域
  - ゲストドメイン，特権ドメイン間のI/O要求・応答のやりとりを行うための空間
- Event Channel
  - 特権ドメイン，ゲストドメインに対して要求・応答の通知するための通信路
- Device Model
  - Qemu由来のデバイスエミュレーション機構
  - ゲストOSからのI/O要求の処理を行う

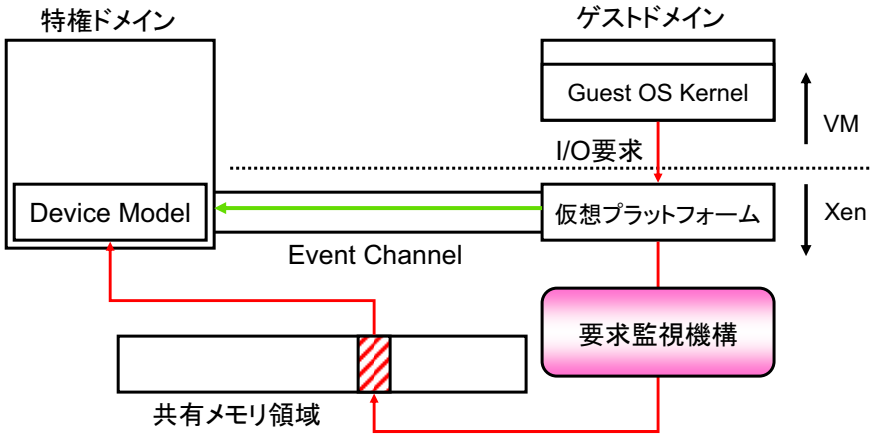
XenにおけるI/O仮想化(2/3)



XenにおけるI/O仮想化(3/3)



I/O要求監視手法



要求監視機構に必要な機能

- どのドメインからの要求かを判別する機能
  - 複数ドメインが存在する場合における，ドメイン別の監視
- 要求の詳細な解析
  - 例：NICを経由したパケットの送信要求なのか，ディスクへの書き込み要求なのか
- 監視結果の保存・閲覧機能

進捗状況と調査項目

- 進捗状況
  - Xenのソースコードの解析
  - I/O要求の生成，I/O要求の特権ドメインへの転送について調べた
  - I/O要求は，ドメインの情報とドメインが持つ仮想CPUのI/O関連の操作に関する情報から生成される
- 調査項目
  - Xen内部で利用されているI/O関連の操作の実体の調査
  - ドメイン，仮想CPU，I/O要求のデータ構造の調査
  - I/O要求に関するパラメータの初期化・設定を行っている関数の調査
    - ゲストドメインの起動の流れを追っていく

おわりに

- まとめ
  - OSの動作観測の有用性
  - 仮想計算機モニタを利用した動作状況観測手法
    - I/O要求監視手法
    - 仮想計算機モニタのI/O仮想化の調査
  - 進捗と今後の調査項目
- 今後の課題
  - ドメイン，仮想CPU，I/O要求のデータ構造の解析
  - 解析結果を基に，監視機構実現するためのソフトウェア設計．実装

仮想計算機上で動作するOSへの動的なプロセッサコア割当て手法

荒木 裕靖 †
† 立命館大学大学院理工学研究科

1 はじめに

近年，仮想化技術を搭載したマルチコアプロセッサが登場し，普及している．このようなプロセッサでは，OSに対してプロセッサコアを動的に割付けることが可能となっている．複数のOSが1台の計算機上で動作するような仮想計算機環境において，これらのコアをOSの負担に応じて，適切にOSへ配分することによって，より効率的で適応的な資源割り当てが実現可能となる．コア資源を動的に配分することによって，必要の無いと思われるコアを停止させ消費電力を抑えることが可能となり，また，動作していないコアを別のOSに割り当てることで，ひとつの計算機上でまったく異なる動作をするシステムを構築することも可能になる．このようなマルチコア資源利用のためには，動的なコア数の変動に適応可能なOSが必要となってくる．

既存のOSでは起動時にコア数を検出するが，その後，コア数が変動する場合を考慮していない．このため，OSは割り当てられたコア資源を負荷の高低にかかわらず独占する．そこで，OSの起動後でも動的にコア資源の変動を可能にすることによって，コア資源の独占を回避することができ，より効率的なコア資源の利用が可能となる．

以上の様な計算機資源の利用は，仮想計算機モニタとOSの協調によって実現される．仮想計算機モニタは仮想計算機上で起動しているOSの動作状況を監視し，状況に応じて資源を適切に管理する．各々のOSは，仮想計算機モニタから提供された資源を適応的に利用する．本研究では特に，資源を適応的に利用可能なOSに着目し，プロセッサコア数の変動に適応可能なマルチコア対応カーネルの構築を行う．このカーネルを利用することで，OSは起動しているアプリケーションの特性によってコア資源を動的に変動させることが可能となる．

現在，上記のカーネル構築に向け，Lavender[1]のマルチコア対応化に取り組んでいる．マルチコア環境でのLavenderの起動のために，既存のOSであるLinuxのSMPでの起動を参考にした．現在は，マルチプロセッサでの起動を行うためAPICの調査を行い，Lavenderへと実装を行っている．

2 Lavenderでの APの起動

動的なコア資源の変動に対応し，かつ，それを適応的に利用可能なカーネルを構築するためLavenderのマルチコア対応化を進めている．

SMPにおいてプロセッサは2つのクラスのプロセッサに分類される．OS起動時に動的に決定され，OSの起動に利用されるプロセッサをブートストラッププロセッサ (Bootstrap Processor，以下BSPと記す) と呼び，それ以外のプロセッサをアプリケーションプロセッサ (Application Processor，以下APと記す) と呼ぶ．マルチコアにおいても，各々のコアを上記のように分類する．

まず，マルチコア対応化のため，自身で起動することが無いAPを起動させる必要がある．以下では，APの起動方法について述べる．

2.1 APの起動手順

APの起動は，BSPによって行われる．BSPは，起動対象のAPに対し自身のLocalAPICを利用したプロセッサ間割込み (InterProcessor Interrupt，以下IPIと記す) によってスタートアップ信号を送信しAPを起動する．以下に，起動手順を記す．

1. APのLocalAPIC IDの取得
2. INIT信号のアサート，デアサート
3. STARTUP信号の送信
4. APの情報を管理，保持

このような流れでAPを起動し，OSで識別を行う．

2.2 LocalAPICの初期化

Local APICの初期化として，CPUID命令によってLocal APICのサポートの有無を確認する．次に，モデル固有レジスタ (Model Specific Register，以下MSRと記す) からの値を取得することで，Local APICが利用可能な状態かを確認し，無効になっていた場合MSRの11ビット目をセットすることでLocal APICを有効にする必要がある．これらの処理が終了すると，Local APICを利用することが可能となる．

しかし，この段階ではLocal APICが提供しているレジスタへとアクセスすることができないため，レジスタがマップされている物理アドレスを仮想アドレスへとマッピングする必要がある．Local APICが提供しているレジスタは物理アドレスの0xFEE0 0000以降にマップされている．これは先ほど利用したMSRの値を参照することで確認できる．

2.3 Lavender でのレジスタのマッピング

今回，物理アドレスの 0xFEE0 0000 へとアクセスするため，Lavender へコードを追加した．Lavender では，図 1 に示すように，仮想アドレスの 0xF800 0000 より低位のアドレスに使用していないアドレス空間があるため，この使用していない 0xF700 0000 から 4KB の領域を 0xFEE0 0000 へとマップするようにコードを追加している．この追加したコードによって，Lavebder 上で 0xF700 0000 へとアクセスすることで，物理アドレスの 0xFEE0 0000 へとアクセスすることが可能となる．これにより，Local APIC の利用が可能となり，AP 起動に用いる IPI の利用が可能となる．

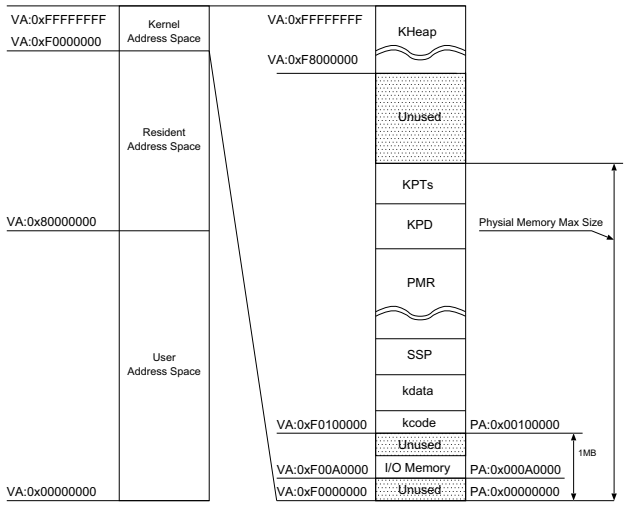


図 1　**Lavender のアドレスマップ**

2.4 IPI の送信

割込みコマンドレジスタ (Interrupt Command Register，以下 ICR と記す) のアドレスは LocalAPIC のベースアドレス (0xFEE0 0000) からオフセット 0x300 と 0x310 の位置にあり，図 2 に示すような領域を持っている．

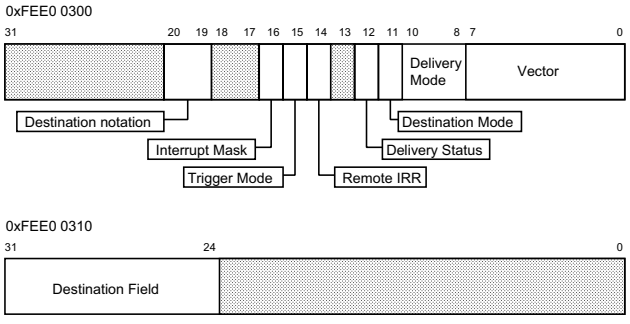


図 2　**割込みコマンドレジスタ**

以下に，AP 起動時に利用する項目を挙げる．

- Destination Field
  IPI を送信する対象のプロセッサの LocalAPIC ID を書込む
- Delivery Mode
  送信する IPI の種類を選択する
- Trigger Mode
  割込みをレベルか，エッジトリガかを選択する
- Remote IRR
  エッジトリガの割込みの際，アサート，デアサートを切替える
- Vector
  今回は STARTUP IPI 利用時に AP が処理を開始するアドレスを書込む

これら ICR の項目に対し適切な値を書込むことにより，INIT，STARTUP といった IPI を送信することができ，これらを受け取った AP は自身を起動する．最後に，BSP によって AP の情報を管理し，AP の起動処理を完了する．

3 おわりに

本稿では，本研究の背景について述べ，適応的なコア資源の利用ために仮想計算機モニタと OS との協調が必要であることを述べた．また，適応的なコア資源の利用が可能なカーネルの構築に向け Lavender を改良していることを述べ，AP の起動方法について述べた．

今後の課題として，現在起動しているコアの情報の管理が行えていないため，Lavender での AP の情報の管理，保持についての実装が挙げられる．また，プロセッサの変動に対応したスケジューリング手法の調査，提案，実装を行っていく．

参考文献

[1] 毛利公一，大久保英嗣:”マイクロカーネル Lavender の設計と開発”，電子情報通信学会論文集 (D-I)，Vol.J82-D-I，No.6，pp.730-739，(1999).

仮想計算機上で動作するOSへの動的なプロセッサコア割当て手法

立命館大学 大学院 理工学研究科
毛利研究室
荒木裕靖

1

目次

- はじめに
- コアの割当て手法
- Lavenderのマルチコア対応
  - APの起動手法
  - Local APICの初期化
  - APICレジスタのマッピング
- 今後の課題

2

はじめに(1/2)

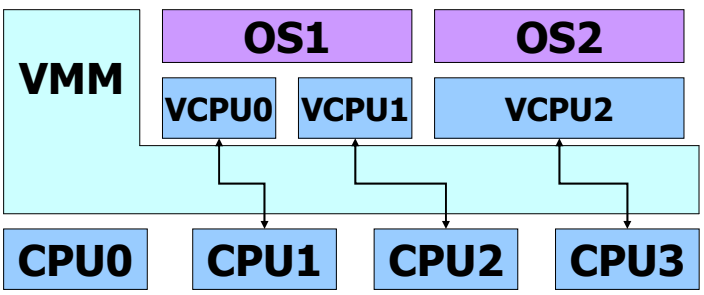
- 近年，マルチコアプロセッサが普及
  - OSに対してコアを動的に割り当てることが可能
  - 従来のOSでは起動後コア数の変動に未対応
- 仮想化技術との組み合わせ
  - 仮想計算機上でのマルチコア環境を実現

動的なコア数の変動に適応可能なOSが必要

3

はじめに(2/2)

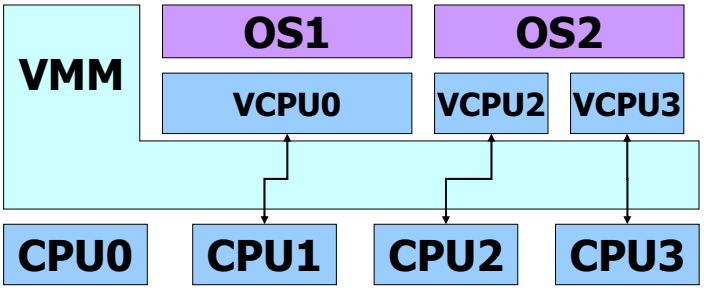
- コア資源の適応的利用



4

はじめに(2/2)

- コア資源の適応的利用

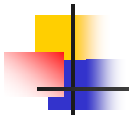


5

目的

- 仮想計算機モニタとOSが協調し適応的に資源を利用可能な環境
- 仮想計算機モニタはOSの動作を観測
- OSは提供された資源を適応的に利用
  - コアの動的な変化に対応するカーネルの構築

6



コア数の変動

- コア数の変動のタイミング
  - 特定のソフトウェアの起動，終了により変動
  - OS全体のCPU利用率により変動
- 変動の処理
  - プロセッサコアの起動，停止
  - スケジューラによる識別の変更

7

共有プロセッサプール(IBM)

- 処理装置と仮想プロセッサ
  - 処理能力を0.01処理装置として指定可能
  - 1.00処理能力で約1物理プロセッサに相当
- 物理プロセッサを複数のOSで共有可能
- 処理の負荷に応じて処理能力を変動可能
  - 再起動なしに指定範囲内での変動が可能

8

Lavenderのマルチコア対応化

- ターゲットとしてLavenderを利用
- 現在シングルプロセッサのみ対応
- コアの適応的割付けのためマルチコア対応化作業を進めている
  - プロセッサは2種類のクラスに分類される
  - Bootstrap processor(BSP)
  - Application processor(AP)

9

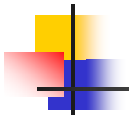
APの起動手順

- APの起動はLocal APICのプロセッサ間割込み(IPI)によって行われる

1. APのLocal APIC IDの取得
2. INIT IPIのアサート，デアサート
   - APのハードウェアチェック
3. START IPIの送信
   - 送信後000x y000hから処理を開始
4. BSPによって該当APの情報を管理，保持

- その他，様々なエラーチェック

10



Local APICの初期化

- プロセッサのベンダーを確認
- Local APICのサポートの有無を確認
- Local APICを有効にする
  - RDMSR命令によってモデル固有レジスタ(MSR)を読込み，11ビット目をセットする
- Local APICレジスタを仮想アドレス空間へマッピング

11

Lavenderでの Local APICレジスタのマップ

- 物理アドレスFFE0 0000hをマップ
  - MSRを参照することで確認
- Lavenderで未使用のアドレスへとマップ
  - ページメモリの初期化ルーチン内に実装
  - ページディレクトリへ追加
  - 仮想アドレス空間F700 0000hをFFE0 0000hへとマップ

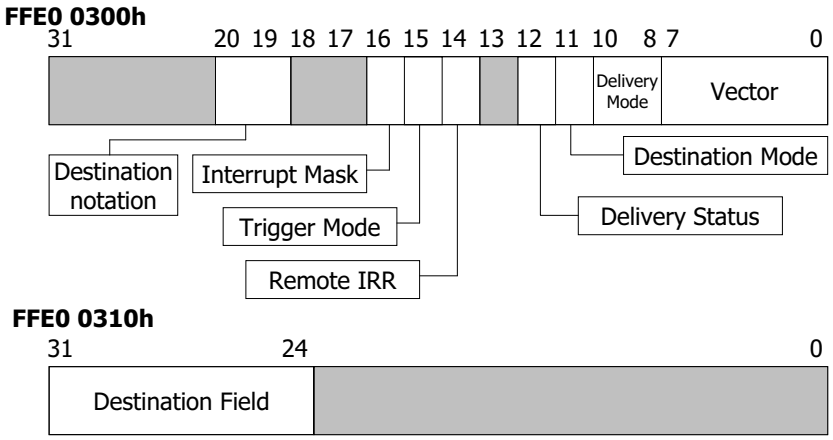
12

IPIの送信方法

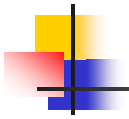
- BSPがLocal APICの割込みコマンドレジスタ(ICR)に書込む
- 書込むICRのアドレス
  - FEE0_0300hとFEE0_0310h
- 書き込む内容
  - 対象プロセッサのLocal APIC ID
  - 伝達モード…INIT，START UP，など
  - トリガモード…エッジ，レベル

13

割込みコマンドレジスタ



14



今後の課題

- LavenderにおけるAPの利用開始
  - AP用初期化コードの追加
  - スケジューラによる識別
- 動的なプロセッサの割当て手法の調査
  - IBM　共有プロセッサプール
- コア変動に対応したスケジューリング手法の調査，実装

15

おわりに

- 適応的なコア資源の利用
- 仮想計算機モニタとOSとの協調
- Lavenderのマルチコア対応
  - APの起動方法
- 今後の課題
  - APをLavenderで利用可能な状態にする
  - 既存技術の調査

16

確率的な分散制約最適化手法を用いたセンサ網の資源割り当て手法の提案*

貝嶋 浩次† 松井 俊浩‡ 松尾 啓志§
名古屋工業大学¶ 名古屋工業大学¶ 名古屋工業大学¶

1 はじめに

複数のセンサを用いて対象物（ターゲット）を多方向から観測する分散協調処理に基づく観測システムに関する研究が行われている．このシステムは自動監視や遠隔講義などのシステムに応用され，複数の対象物を観測する基礎となる技術である．

一方で，分散制約最適化問題の研究が行われている．この問題は分散協調システムにおける問題解決の基礎的な研究として位置付けられており，問題を解くための分散制約最適化手法には最適解の保証がある厳密解法と最適解の保証はないが解の探索時間が短い非厳密解法に大きく分けられる．

分散制約最適化手法を分散協調処理に基づく観測システムに適用した研究はこれまでに多く行われている．しかし，これらの研究では問題の規模が大きくなった時に解の探索時間が大幅に増加するため，実際的な規模への適用には課題がある．本研究では大規模な問題へ適用させるため非厳密解法の 1 つである確率的な分散制約最適化手法を適用する手法を提案する．

2 分散制約最適化問題

分散制約最適化問題はネットワーク環境を変数と制約に定式化した離散制約最適化問題である．分散制約最適化問題は n 個の変数 $x_1, x_2, ..., x_n$, 変数の値域 $D_1, D_2, ..., D_x$, 変数 x_i, x_j 間の制約 $c_{i,j} \in C_s$ 及び制約に対応した評価関数 $f_{i,j}$ からなる．変数と制約は複数のエージェントに分散して配置される．各エージェントは他のエージェントとメッセージ通信を行い評価関数の値を大域的に最適化する変数の値の組み合わせを求める．

3 センサ網の資源割り当て問題

分散協調処理に基づく観測システムではターゲットを複数のセンサを用いて多方向から観測する．各センサは有限の視野を持ち，視野内にいるターゲットに対して観測するセンサの集合を割り当てる．この時，各センサは視野，観測可能なターゲット数など有限な資源の中でセンサ間の割り当ての整合性を保つ必要がある．

本研究では図 1 のようなグリッド状にセンサを配置した問題を扱う．各センサの視野は隣接する四角形の領域で表される．例えば，センサ a の場合は視野内にターゲット 1 が存在するため，センサ a はターゲット 1 が視野内にいることを知る．

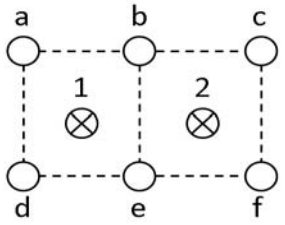


図 1: 配置例

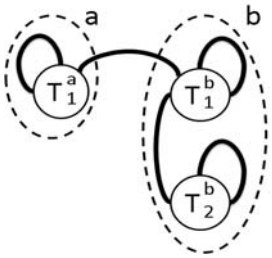


図 2: 制約網の表現

図 1 のセンサ a,b の部分を定式化すると図 2 のようになる．センサ i が知るターゲット j に関する変数 T_j^i は，そのターゲットを観測するセンサの割り当てを値として持つ．T_1^a が abe という値をとった場合，センサ a はターゲット 1 をセンサ a,b,e の 3 個のセンサで観測するということを知る．また，変数の値域は観測するセンサの割り当ての集合である．

定式化した制約網には，同じセンサが持つ変数間で結ばれる intra-agent 制約 (図 2 の T_1^b, T_2^b 間)，同じターゲットに関する変数間で結ばれる inter-agent 制 (図 2 の T_1^a, T_1^b 間) 約，各変数が独立して持つ ignore 制約の 3 種類の制約が存在する．intra-agent 制約はセンサが持つ観測資源の制限を示した制約で，1 つのセンサは 1 つのターゲットのみ観測できることを意味する．inter-agent 制約は意志決定の整合性を示した制約で，同じ値（割り当て）をとることを意味する．ignore 制約は観測制度の要求を示した制約で，ターゲットを k 個以上のセンサで観測することを意味する．intra-agent 制約または inter-agent 制約を違反する割り当ては実行不可能であるため，これらの制約を全て満足した実行可能解を求める必要がある．

4 確率的な分散制約最適化手法

DSTS(Distributed Stochastic Tabu Search)[1] はタブーリストを用いた確率的な分散制約最適化手法である．各ノードは受信したメッセージ通信により得る近傍ノードの値を基にコストの改善量の最大値 Δ_{max} とその時の値 `new_value` を求める．$\Delta_{max} \geq 0$ の場合は確率 p1，Δ_{max}<0 の場合は制約違反が存在する場合のみ確率 p2 で値を `new_value` に変更する．値を変更したノードは近傍へ変更後の値を送信し，一定期間（タブー期間）遷移を禁止する値をタブーリストへ書き込む．この処理を終了条件を満たすまで繰り返す．DSTS のパラメータは p1,p2, タブー期間で，これらのパラメータは問題に応じて適切に決定する必要がある．

*Sensor network resource allocation using distributed stochastic optimization method

†Koji Kaishima
‡Toshihiro Matsui
§Hiroshi Matsuo
¶Nagoya Institute of Technology

5　提案手法

5.1　実行可能解の探索

確率的な手法の DSTS は最適解の保証がないため実行可能解を発見できる手法が必要である．そこで次の改良を行う．まず，必ず満足する必要がある intra-agent 制約と inter-agent 制約を hard 制約とみなし，これらの制約の重みを ignore 制約の重みに比べ十分大きくする．また，DSTS の値変更条件において，Δ_{max}<0 の場合は hard 制約の違反が存在する場合のみ確率 p2 で値を変更するように設定する．これらの設定を行うことで hard 制約を満足した変数は値を変更しなくなるため割り当ては実行可能解に収束できる．タブーサーチを基にした DSTS と併用することで実行可能解を発見できる可能性は十分高くなると考えられる．

5.2　制約の重み付け

変数が持つ制約の個数は制約の種類によって異なるため，局所解の回避と探索効率を改善する重み付けが必要である．そこで，次の式のように変数 T_j^i が持つ intra-agent 制約の重み `W_intra(i,j)` を設定する．ただし，$\phi(i)$ はセンサ i が観測するように割り当てられたターゲット数，$Neignor(i,j)$ は変数 T_j^i と inter-agent 制約を持つセンサの集合，`W_intra(i,j)(k,j)` は変数 T_j^i が持つ T_j^k 間との inter-agent 制約の重みである．

$$W\_intra(i,j) = (\phi(i) - 1) \sum_{k \in Neighbor(i,j)} W\_inter(i,j)(k,j)$$

intra-agent 制約を満足するためには $\phi(i)$ が 1 以下になるよう変数の値を遷移させる必要がある．そこで，$\phi(i)$ の値が大きい程 `W_intra(i,j)` が大きくなるように重み付けを行う．この重み付けにより局所解を回避し高速に実行可能解を発見できると考えられる．

6　評価

提案手法の有効性をシミュレーションで評価した．図 1 のようにセンサをグリッド状に配置した問題でターゲット数を 10,15,20,25 と変化させた時の解の精度，サイクル数を測定した．解の精度は 3 個のセンサで観測されるターゲット数とした．また，サイクルは全てのエージェントが近傍メッセージに基づく一連の処理を行うまでを表した探索時間の単位である．解を発見するまでを 1 回の試行とし，1000 回の試行の平均を測定した．その他の設定は次の通りである．

- ターゲットは制約網が単一の連結成分からなるように配置する
- i サイクル目に近傍ノードから送信されたメッセージは (i+1) サイクル目に受信する
- 変数の値域は 3 個のセンサで観測する場合とそのターゲットを観測しない場合に制限する

表 1: DSTS のパラメータ

パラメータ A	p1=0.50,p2=0.10, タブー期間 1 サイクル
パラメータ B	p1=0.90,p2=0.20, タブー期間 1 サイクル

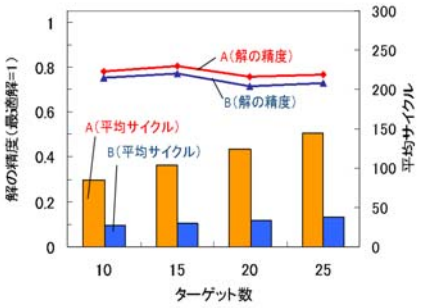


図 3: 実験結果

- パラメータは解の精度が最も高かったパラメータ A と，探索時間が最も小さかったパラメータ B を用いる．設定値は表 1 の通りである
- 値を変更したノードは変更前の値をタブーリストへ書き込む

実験結果を図 3 に示す．横軸はターゲット数，縦軸は最適解を 1 とした場合の解の精度とサイクル数の平均である．折れ線グラフが解の精度，棒グラフがサイクル数を表す．解の精度は，ターゲット数 25 の問題においてパラメータ A,B で最適解の約 4%の差がある．しかし，これは 3 個のセンサで観測されるターゲット数に 1 個未満の差しかないことを意味するため，パラメータ A,B による解の精度の差は小さいといえる．平均サイクルは，パラメータ B の方がパラメータ A に比べサイクル数の増加量が小さい．また，パラメータ B はターゲット数が 25 の問題に対して平均サイクルが 50 サイクル未満であり高速に実行可能解を発見できていることがわかる．以上の結果から探索時間を優先したパラメータ設定は有効であるといえるため，提案手法は大規模な問題に対して有効であるといえる．

7　まとめ

複数のセンサを用いた分散協調処理に基づく観測システムに確率的な分散制約最適化手法である DSTS を適用し、大規模な問題に対応する手法を提案した。シミュレーションによる結果から提案手法は大規模な問題に対して有効であることを確認した。今後の課題として、ターゲットが移動する場合などのさらなる実際的な問題への適用や分散センサ網のための協調プロトコルとの比較・検討が挙げられる。

参考文献

[1] 飯塚泰樹, 鈴木浩之. “Tabu search を用いたマルチエージェント型分散制約充足手法の提案”. Technical report, 電気情報通信学会研究報告, 2006.

確率的な分散制約最適化手法を用いたセンサ網の資源割り当て手法の提案

貝嶋浩次　松井俊浩　松尾啓志
名古屋工業大学

背景

- 分散協調処理に基づく観測システム
  - 複数のセンサを用いて対象物を多方向から観測

【応用例】
- 自動監視
- 遠隔講義

複数のセンサを複数の観測対象に割り当てる
資源割り当て問題を解く必要がある

背景

- 分散制約最適化問題
  - 分散協調システムにおける問題解決の基礎となる問題
  - ネットワーク環境を変数と制約に定式化

- 分散制約最適化問題の解法

手法	最適解の保証	探索時間
厳密解法	ある	長い
非厳密解法 （確率的な手法など）	ない	短い

背景

- 従来研究
  分散制約最適化手法をセンサ網の資源割り当て問題へ適用

- 問題点
  問題の規模が大きくなった時に探索時間が大幅に増加

探索時間の短い確率的な分散制約最適化手法を
観測システムに適用する手法を提案

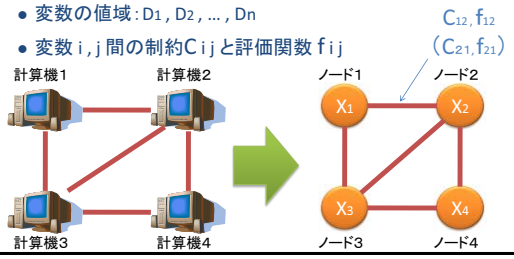
発表の流れ

1. 分散制約最適化問題への定式化
   - 分散制約最適化問題の説明
   - センサ網の資源割り当て問題の形式化
2. 適用する確率的な手法の説明
   - DSTS（Distributed Stochastic Tabu Search）
3. DSTSを適用する際の問題点
   - 問題を解決する手法の提案
4. シミュレーションによる評価

分散制約最適化問題

ネットワーク環境を変数と制約に定式化した問題

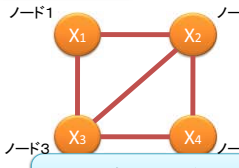
- 変数はノード(エージェント)，制約はリンクに相当
  - 変数：$X_1, X_2, \ldots, X_n$（ノード i は変数 Xi を持つ）
  - 変数の値域：$D_1, D_2, \ldots, D_n$
  - 変数 i , j 間の制約C i j と評価関数 f i j



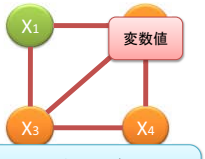
分散制約最適化問題

- 各ノードが知っている情報
  - 自ノードの変数値
  - 自ノードが関係する制約及び近傍ノード

制約網全体



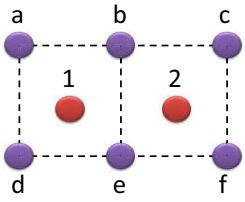
ノード1（変数X_1）が知る情報



全ての制約に対するコストの総和が
最小となる値の組み合わせが最適解となる

センサ網の資源割り当て問題

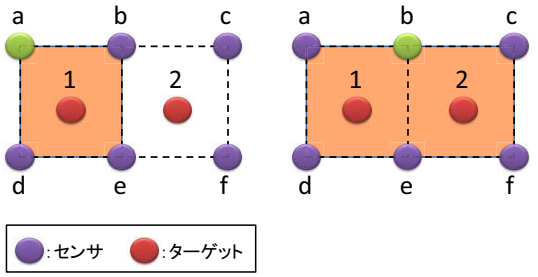
配置



：センサ　：ターゲット

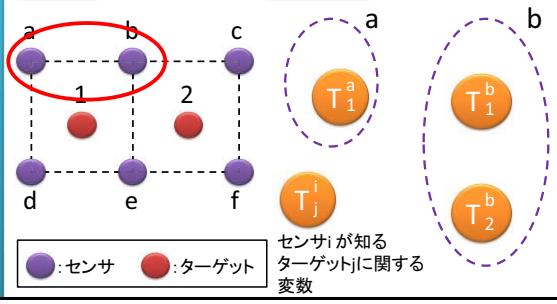
センサ網の資源割り当て問題

センサa の視野　センサb の視野



センサ網の資源割り当て問題

配置　形式化

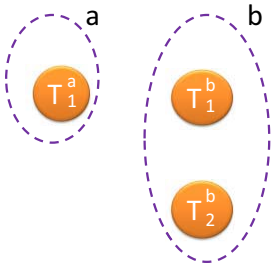


制約網の変数

変数はターゲットを観測する
センサの組み合わせを
値として持つ

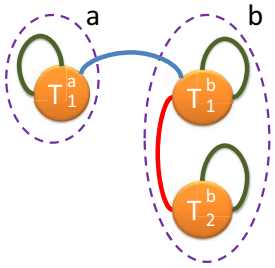
T^a_1 = a b eの場合
センサaはセンサa, b, eが
ターゲット1を観測すること知る

D^a_1 = {a, b, d, e,
ab, … , abde ,IGNORE}
観測するセンサなし



制約網の制約

- intra-agent 制約
  - 観測資源の制限
  - 1つのセンサは1つの
    ターゲットのみ観測できる
- inter-agent制約
  - 意思決定の整合性
  - 同じ値（割り当て）をとる
- ignore 制約
  - 観測精度の要求
  - 1つのターゲットをk個以上の
    センサで観測（今回はk＝3）



制約の個数と実行可能解

各変数が持つ制約の個数

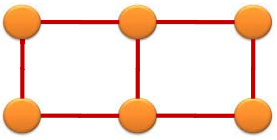
制約の種類	制約の個数	変数 T_j^i が持つ制約の重み
intra-agent制約	高々1個	W_intra(i,j)
inter-agent制約	3個	W_inter(i,j)(k,j)
ignore制約	1個	W_ignore(i,j)

intra-agent制約とinter-agent制約を
全て満足する割り当ては
実行可能である（実行可能解）

DSTS（Distributed Stochastic Tabu Search）[飯塚,2006]

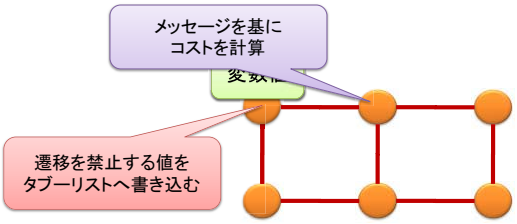
タブーリストを用いた確率的な分散制約最適化手法

- コストの改善量の最大値Δ_{max}とその時の値new_valueを求める
  - **$\Delta_{max} \geqq 0$の場合**
    確率p1で値をnew_valueに変更
  - **$\Delta_{max} < 0$の場合**
    制約違反が存在する場合のみ
    確率p2で値をnew_valueに変更



DSTS（Distributed Stochastic Tabu Search）[飯塚,2006]

- 値変更後の処理
  - 近傍ノードへ変更後の値を送信
  - 一定期間(タブー期間)遷移を禁止する値を
    タブーリストへ書き込み



DSTSを適用する際の問題点

- **実行可能解の探索**
  - 確率的な手法であるDSTSは最適解を求める保証がない
  - 求める解は実行可能解でなければならない
- **制約の重み付けの検討**
  - 変数が持つ制約の個数は制約ごとに異なる
  - 局所解の回避と探索効率を改善する必要がある

実行可能解の探索

問題点（1）

- DSTSは最適解を求める保証がない
- 求める解はintra-agent制約とinter-agent制約を満足した実行可能解でなければならない

改良点（1）

- $W\_intra(i,j)$, $W\_inter(i,j)(k,j)$ >> $W\_ignore(i,j)$
- DSTSの値の変更条件

  Δ_{max}＜0の場合：intra-agent制約かinter-agent制約の違反が存在する場合のみ確率p2で値を変更

intra-agent制約とinter-agent制約を満足すれば
値の変更は行われなくなるため
実行可能解を発見できる確率は十分高い

制約の重み付けの検討

問題点（2）

- 変数が持つ制約の個数は制約ごとに異なる
- intra-agent制約とinter-agent制約の重み付けを検討する必要がある

改良点（2）

- $W\_intra(i,j) = (\varphi(i) - 1) \sum_{k \in Neighbor(i,j)} W\_inter(i,j)(k,j)$

  intra-agent制約：$\varphi(i) \leqq 1$

$\varphi(i)$：センサiが追跡するように割り当てられたターゲット数

$Neighbor(i,j)$：変数Tとinter-agent制約を持つセンサの集合

局所解を回避し
高速に実行可能解を発見することができる

評価

提案手法の有効性を確認するためにシミュレーションによる評価を行う

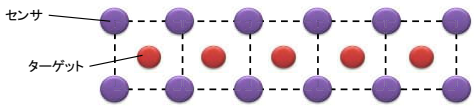
1. 実行可能解の探索
   - 提案手法で実行可能解が求められることを確認
2. 厳密解法との比較
   - DSTS（提案手法）とAdopt（厳密解法）の比較

   Adopt(Asynchronous Distributed Constraint Optimization) [P.J.Modi,2003]
3. 大規模問題への適用
   - パラメータを変化させた時の結果の比較

評価1　実行可能解の探索

- 提案手法で実行可能解を発見できることを確認
  - 制約違反数の遷移を各制約ごとに測定
- 実験環境
  - 一列のグリッドにセンサを配置（以降の評価も同様）
  - 問題の規模

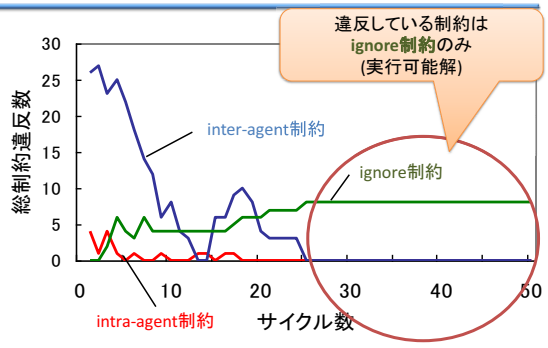
ターゲットの数	5
センサの数	12
変数の数	20



評価1　実行可能解の探索

- 「サイクル」を用いて評価
  全てのノードが以下の処理を行うまでを1サイクル とする
  - 近傍からメッセージの受信
  - メッセージの処理
  - 必要な場合はメッセージの送信(通信遅延は1サイクル)
- 変数の値域Dの大きさは5
  例：D＝{a b c, a b d, a c d, b c d, IGNORE}
  （a b c, a b d, a c d：3個のセンサで観測　／　b c d：観測しない　／　IGNORE：観測しない）
- DSTSのパラメータ
  - p1 = 0.90 , p2 = 0.20 , タブー期間　1サイクル
  - タブーリストへ書き込む値は変更前の値

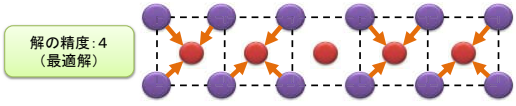
評価1　総制約違反数の遷移



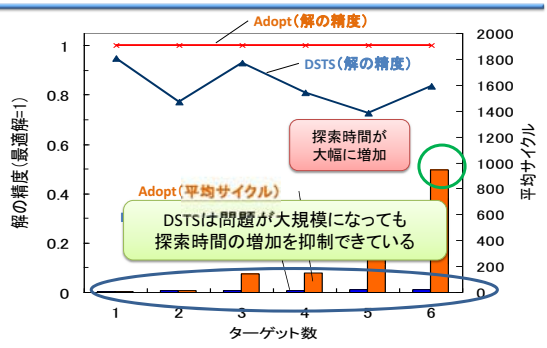
評価2　厳密解法との比較

- DSTS（提案手法）とAdopt（厳密解法）の比較
- 実行可能解を求めるまでを1回の試行とし
  1000回の試行の平均を測定
  - 解の精度：3個のセンサで観測されているターゲット数
  - サイクル数
- 問題の規模

ターゲットの数	1	2	3	4	5	6
センサの数	4	6	8	10	12	14
変数の数	4	8	12	16	20	24



評価2　解の精度と探索時間



評価3　大規模問題への適用

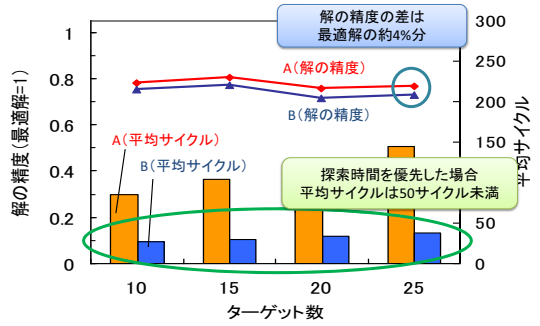
- 大規模な問題でパラメータを変化させた時の結果を比較
- 実行可能解を求めるまでを1回の試行とし
  1000回の試行の平均を測定
  - 解の精度：3個のセンサで観測されているターゲット数
  - サイクル数
- 問題の規模

ターゲットの数	10	15	20	25
センサの数	22	32	42	52
変数の数	40	60	80	100

- DSTSのパラメータ

パラメータ名	p1	p2	タブー期間	優先項目
A	0.50	0.10	1	解の精度
B	0.90	0.20	1	探索時間

評価3　解の精度と探索時間



今後の課題

- さらなる実際的な問題への適用
  - ターゲットの優先度が異なる環境
  - ターゲットが移動する環境
  - センサがグリッド状以外に配置された環境
- 分散センサ網のための協調プロトコルとの比較・検討

まとめ

- 確率的な分散制約最適化手法を用いた
  センサ網の資源割り当て手法を提案した
  - アルゴリズムはDSTSを使用
  - 実行可能解を発見できるように改良
  - 制約の重み付けの検討
- シミュレーションによる評価の結果から以下の点を
  確認した
  - 提案手法を適用した結果、実行可能解を発見できる
  - 問題が大規模になっても探索時間の増加を抑制できる
  - 探索時間を優先したパラメータ設定でも
    解の精度を大きく落とすことなく実行可能解を発見できる

分散制約最適化手法を用いたコアロケーションスケジューリングへの適用

Co-allocation scheduling using distributed constraint optimization technique

行方 裕紀* 松井 俊浩* 松尾 啓志*
Yuki Namekata Toshihiro Matsui Hiroshi Matsuo

1 はじめに

グリッドコンピューティング技術の進展により, 広域ネットワーク上にあるコンピュータ資源を用いた大規模科学技術計算が実現可能となった. このような広域・大規模グリッド計算環境においては分散した計算・通信資源を同時に確保する事（コアロケーション）[1] が必要である. 近年, このような環境における複数スケジューラによる分散スケジューリングの研究が行われている. 本研究では, 非同期分散環境における協調問題解決のモデルである分散制約最適化問題（Distributed Constraint Optimization,DCOP）の, コアロケーションのための分散スケジューリングへの適用を提案する. ここでは, 複数のローカルスケジューラが分散協調を行うモデルを想定する. DCOP の探索アルゴリズムとして, 制約網に対する擬似的な木（Pseudo tree）による変数順序付けを用いた解法が提案されている. 本研究では, このような擬似木を用いた動的計画法に基づくアルゴリズム [2] の適用を検討する. この手法は厳密解法であり, メッセージ数が変数の数に対して線形だが, ノード間で交換されるメッセージのサイズが木の幅に対して指数関数的に増加する. そこで, 擬似的な木の幅を制限することを考慮した問題の緩和手法を提案する.

2 コアロケーションスケジューリング問題の分散制約最適化問題としてのモデル化

2.1 コアロケーションスケジューリング問題

コアロケーションスケジューリング問題では, リソース集合 $R := \{R_1, ..., R_n\}$, ジョブ集合 $\epsilon := \{E^1, ..., E^k\}$, 時間領域 $\tau := \{1, ..., T\}$, の情報が与えられる. ジョブ E^k には, ジョブの必要とするリソース $A^k(\subset R)$, 時間枠 L^k, ジョブの価値 V^k, の３つの情報が, 関連付けられている. 同じリソースを同じ時間に使うことはできない. 時間領域 τ の中でスケジュールされたジョブの価値の和が最大になる解を探すことが目的である.

2.2 制約網の作成

制約網の作成には, イベントスケジューリング問題を制約最適化問題として形式化する EAV(Events as Variable)[3] を用いた. EAV では, ジョブごとに変数を作り, ジョブの開始時間を変数の領域とする. 制約辺は, 共通するリソースを必要とするノード間に作成される. また, ノード間には評価関数が設けられ, 大域利得を最大にする変数の割当てを考える.

2.3 Pseudo tree

Pseudo tree は, 制約網に含まれるノードの半順序関係を与える構造である. 各ノード (変数)[1]は木の深さによって順序付けされ, 制約辺は, 木の枝となる木辺（tree edge）とそれ以外の辺である後退辺（back edge）に分類される. 擬似木は, サブツリー間に制約辺がないため, 探索手法の効率化に役立つことが知られている.

2.4 DPOP アルゴリズム

DPOP アルゴリズムは Pseudo tree に基く, 最適解を求めることができるアルゴリズムである. DPOP アルゴリズムは, ２段階の処理からなる. 各段階ではそれぞれ,UTIL メッセージ・VALUE メッセージの伝搬による処理が行われる. UTIL メッセージの伝搬による処理は, 擬似木の葉ノードからはじまり木辺に沿って擬似木の根ノードへとボトムアップに広がる. UTIL メッセージは動的計画法に基づいておりノード間の UTIL メッセージサイズが後退辺の数により指数関数的に増加する. ここで, 一番サイズの大きい UTIL メッセージの関連ノード数を木の幅と呼ぶ.

3 提案手法

DPOP では, 木の幅が増加すると UTIL メッセージのサイズが指数関数的に増える. そこで, 本論文では, 問題の緩和により, 木の幅を削減する手法を提案する. 提案手法を以下に示す.

3.1 後退辺の削除

コアロケーションでは, 同じリソースを持つジョブが同時に, 同じ変数（タイムスロット）を取ることがない. この制約条件を利用して, 木の後退辺の削除を行う. 後退辺の削除は, 削除する後退辺でつながれたノードの変数を互いに取れない時間枠にすることで行われる. この方法により, ジョブが必要としている同じリソースが同じ時間にスケジューリングされることはなくなり, 矛盾無くスケジューリングが行われる. しかし, 問題の緩和により最適解が得られない場合がある. 本研究の変数の値域の制限における戦略では, 木の上位ノードを優先, また, 必要な時間領域が大きいタスクを優先して早い時間領域に変数を割り当てる 2 種類の戦略をそれぞれ用いる.

3.2 問題の分割

問題を分割し, 問題を縮小することで木の幅を小さくする方法を提案する. 本研究では初期の検討として問題の分割では, ジョブ数が半分になるような分割方法を用いる. 分割後に, 片方のグラフで解を求める. 次に, もう片方のグラフで分割した際に制約辺を失ったノードに対して変数値の調整を行う. 変数値の調整では, 解の求まったノードとの間に制約辺があったノードは変数値を解以外の時間枠にする. 問題の緩和により最適解が得られない場合がある.

*名古屋工業大学 Nagoya Institute of Technology

[1]本研究では便宜上, 各ノードは一つの変数のみを持ち, ノード＝変数とみなす

4　シミュレーションによる評価

提案手法で述べた, 制約辺の削除, 問題の分割の有効性を検討するため, シミュレーション実験を行なった. 実験では, 複数のローカルスケジューラが協調処理を行なうモデルを想定した.

4.1　実験方法

以下の環境で実験を行なった. 初期の検討として, 問題のサイズはジョブ数が 20,40,60,80,100 個とした. リソース (クラスタ) の数は 50 個とした. 各ジョブには必要とするリソースは 2 個, タイムスロット数は 1〜3 となる場合を想定した. 各問題は, 単一の連結成分からなる制約網が作成されるように生成した. また, 大規模な時間単位を想定し, ノードの持つ変数領域は $\tau := \{1, ..., 20\}$ とした. 提案手法と従来手法の解の精度と UTIL メッセージのサイズを, 問題のサイズを変化させ比較した. 後退辺の削除では, 木の幅が 4 になるまで後退辺の削除の操作を複数回を繰り返した. グラフの分割では, ジョブの数が半分になるように分割した. 評価の尺度として, 解の精度を定義した. 解は, ジョブを可能なかぎり値の小さい (順序が先の) 時間領域にスケジュールしたときのタイムスロットの最大長で表し, 解の精度はどれだけ最適解に近いかを表すため「最適解の評価値／提案手法の解の評価値」とした. ここで最適解とは, ジョブがスケジュールされたタイムスロットの最大長が最も小さい解である.

また, 後退辺の削除では 2 種類の戦略に対して, スケジューリングの経験則の使用・未使用を考慮し, その戦略に相当する Greedy 手法との比較を行なった.

戦略 1:タスクの順番はランダムな Greedy 手法 (木の上位ノードを優先する戦略に相当)

戦略 2:タスクに必要な時間領域が大きい順にソートを行なった Greedy 手法 (必要な時間領域が大きいタスクを優先に相当)

4.2　結果

問題のサイズと解精度, 削減した制約辺の数との関係を図 1 に示す. 図 1 より, 両手法とも制約辺の削減が増えるにつれて解の精度が低下している. これは, 制約辺の削減により問題が緩和し, 最適解が変化したためと考えられる. しかし, ジョブ数を変化させた場合でも解の精度の低下は比較的少ない. 探索範囲を限定し, メッセージのサイズの削減を行なったが, 両手法とも 0.95〜1 程度の解の精度がある.

表 1 より, 両手法によって UTIL メッセージのサイズが削減されていることが確認できる.

また, 後退辺の削除と Greedy 手法との比較の結果を図 2 に示す. 提案手法 (戦略 1) と Greedy 手法 (戦略 1), 提案手法 (戦略 2) と Greedy 手法 (戦略 2) との比較により, 同じ戦略を用いた場合, 提案手法が Greedy 手法を下回っていないことが確認できる. また, 提案手法 (戦略 1) と提案手法 (戦略 2) の比較により, 提案手法ではスケジューリングの経験則を用いたほうがよい結果となった. 提案手法での, 値域の分割の戦略が解の精度に影響を及ぼしていることがわかる.

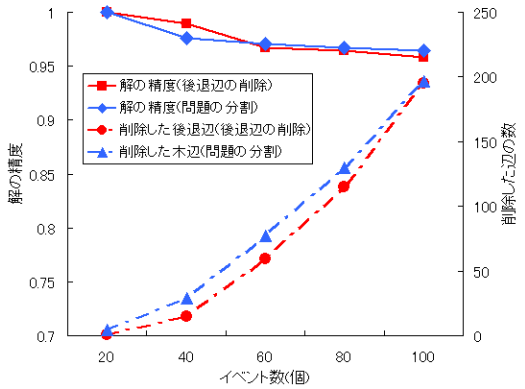


図 1: 解の精度と削減後退辺数の関係

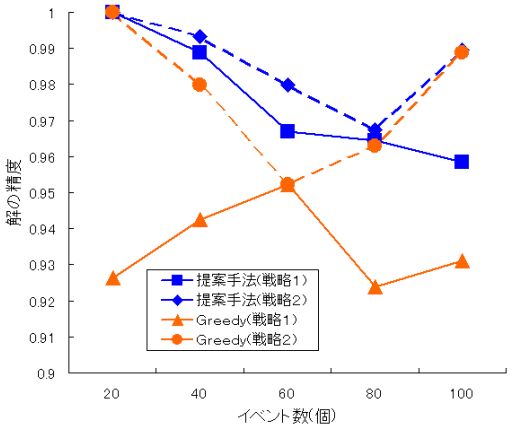


図 2: 提案手法と Greedy 手法との比較

ジョブ数 (個)	20	40	60	80	100
改善前 (MB)	2.5×10^{-1}	2.0×10^{6}	6.7×10^{16}	1.2×10^{33}	6.0×10^{53}
後退辺の削除	1.3×10^{-1}	6.1×10^{-1}	6.1×10^{-1}	6.1×10^{-1}	6.1×10^{-1}
問題の分割	9.0×10^{-3}	1.6×10^{2}	1.8×10^{-1}	1.1×10^{1}	4.9×10^{1}

表 1. 提案手法による UTIL メッセージサイズの削減

5　おわりに

本研究では分散制約最適化問題をグリッド計算環境のコアロケーションのための分散スケジューリングに適用した. 解探索は動的計画法に基づくアルゴリズム DPOP を用い, メッセージサイズの増加を抑制するために 2 つの問題の緩和手法を示した. 後退辺の削除による方法では,Pseudo tree の後退辺を削除することにより木の幅を小さくし, 解探索の範囲を限定した. また, 問題の分割による方法では, 問題を半分に分割することにより問題のサイズを小さくし, 解の組合せの範囲を限定した. これらの手法により, 問題を緩和することで UTIL メッセージのサイズを減少させた. シミュレーションを用いた初期の評価においては, 提案手法により, メッセージサイクル数, メッセージのサイズ, が削減され, 一方で, 解の精度の低下は比較的小さいことが示された. また, 後退辺の削除では値域の分割にスケジューリングの経験則が解に影響を及ぼすことが確認できた.

今後の課題として, 後退辺の削除, 問題の分割におけるヒューリスティクスの検討が挙げられる. 後退辺の削除では変数の割当て, 問題の分割では分割場所を変えることによって探索範囲を改善し, 効率のよい探索が期待できる. また, 実際的な問題への適用, 他のスケジューリング方法との比較も今後の課題である.

参考文献

[1] 竹房あつ子, 中田秀基, 工藤知宏, 田中良夫, 関口智嗣, ”計算資源とネットワーク資源を同時確保する予約ベースグリッドスケジューリング” SACSIS 2006, pp. 93-100, 2006.

[2] A. Pecu and B. Faltings, ”A Scalable Method for Multiagent Constraint Optimization”, Proc. 9th Int. Joint Conf. on Artifical Intelligence, pp. 266-271, 2005.

[3] R. T. Maheswaran, M. Tambe, E. Bowring, J. P. Pearce and P. Varakantham, ”Taking DCOP to the RealWorld:Efficient Complete Solutions for Distributed Multi-Event Scheduling”, Proc. 3rd Int. Joint Conf. on Autonomous Agents and Multiagent Syatems, pp.310-317, 2004.

分散制約最適化手法を用いた コアロケーションスケジューリングの適用

名古屋工業大学
○ 行方 裕紀
松井 俊浩
松尾 啓志

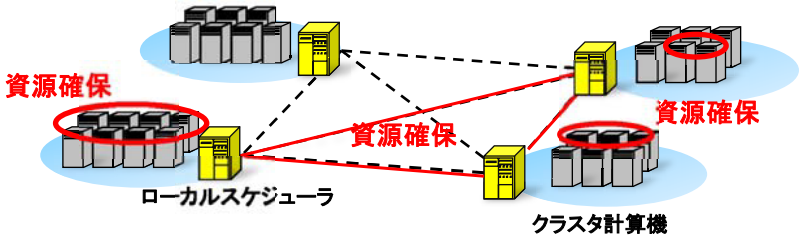
研究の背景 グリッド計算環境におけるスケジューリング

グリッド計算

広域のネットワーク上にあるコンピュータ資源を同時に用いた大規模計算

- コアロケーション
  複数のクラスタ群でタスクに必要な資源を同時に確保
- 分散協調スケジューリング ⇒管理面、対故障性などに有効
  複数のクラスタのローカルスケジューラによって行われるスケジューリング

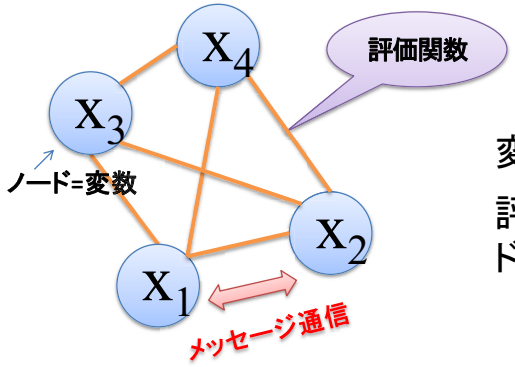
⇒非同期分散環境における協調問題解決のモデルが必要



研究の背景　分散制約最適化問題

分散制約最適化問題(Distributed Constraint Optimization, DCOP)

ノード(端末)間の分散協調処理のための基礎的な探索問題



変数：ノードの状態

評価関数：2項間のノードの状態に対する評価

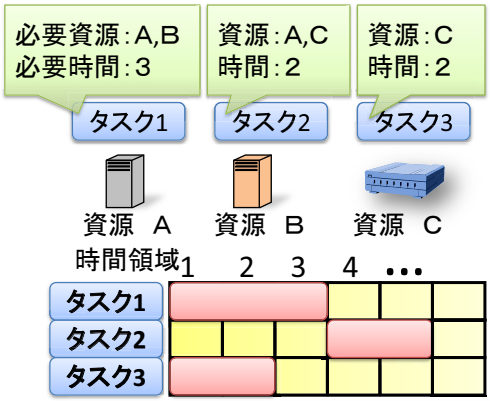
⇒各ノードはメッセージ通信により全体の利得が最良になる自変数の値を探索

本研究の概要

1. 問題の形式化
   - コアロケーションスケジューリングを分散制約最適化問題として形式化
2. 問題の緩和手法(提案手法)
   - 分散制約最適化手法を適用とその問題点
   - 問題の緩和による2つの手法を提案
     - 後退辺の削除(問題に含まれる制約数の削減)
     - 問題の分割(問題の規模の削減)
   - シミュレーションによる提案手法の評価

コアロケーションスケジューリング問題の形式化

タスクに必要な資源の同時確保を考慮したスケジューリング問題として扱う

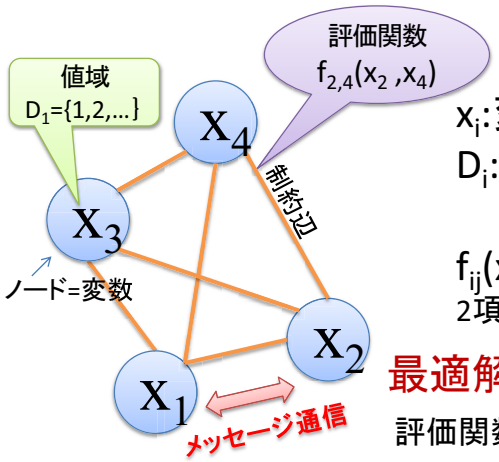


- タスク集合
  ε:={1,2,3,⋯,N}
- 資源集合
  R:={A,B,C,⋯}
- 時間領域
  τ:={1,2,3,⋯,T}

最適なスケジューリング
- 同じ時間領域内で同じ資源を使用しない(実行可能解)
- 全てのタスクの終了する時間領域長が最小

分散制約最適化問題の形式化

各ノードは協調処理により評価関数を最適化



x_i:変数
D_i:値域

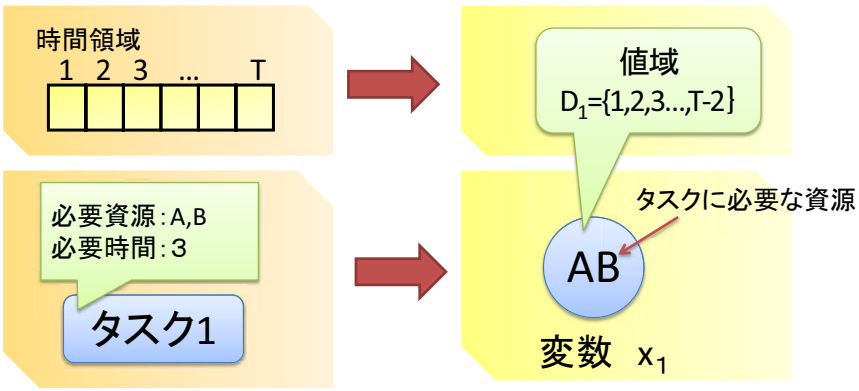
$f_{ij}(x_i, x_j)$:評価関数
2項関係のある変数に対する評価

最適解
評価関数の和(Σf_{ij})が最大となる解

スケジューリング問題から
分散制約最適化問題への変換

Events as Variable [Rajiv T.Maheswaran,2004]

各タスクごとに変数を作り、タスクの開始時間を値域とする

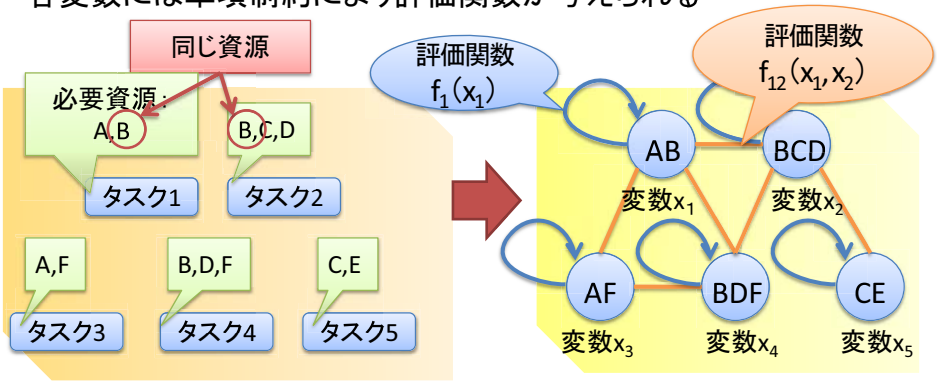


コアロケーションスケジューリグ問題　　分散制約最適化問題

スケジューリング問題から
分散制約最適化問題への変換

Events as Variable [Rajiv T.Maheswaran,2004]

- 同一の資源を用いるタスク間には2項制約により評価関数が与えられる
- 各変数には単項制約により評価関数が与えられる

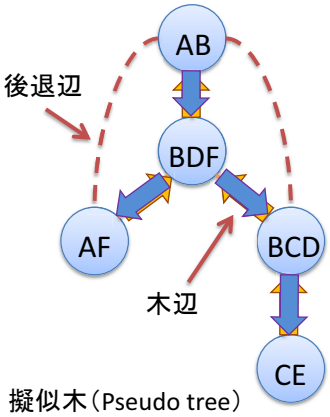


コアロケーションスケジューリグ問題　　分散制約最適化問題

分散制約最適化問題の解法

DPOPアルゴリズム [Adrian Petcu,2005]

- 動的計画法に基づいた分散アルゴリズム
- 最適解を得る厳密解法



擬似木（Pseudo tree）
ノードに半順序関係を与えたもの

UTILメッセージ

- 葉から木辺を通じてボトムアップに送信
- 各ノード間の計算方法は、動的計画法に基づいている

VALUEメッセージ

- 根から木辺を通じてトップダウンに送信
- 送信したUTILメッセージを基に、解の決定を行う

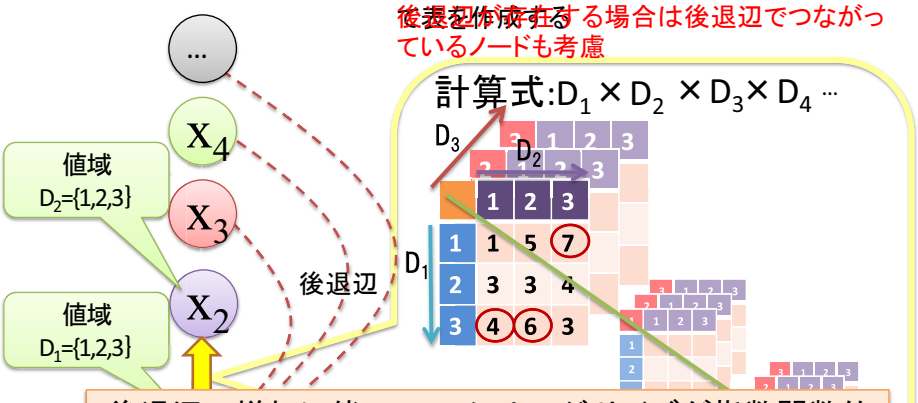
特徴：
•メッセージ数と実効サイクル数が問題の規模に対して線形に増加
•比較的容易に実装可能

UTILメッセージサイズの増加

UTILメッセージ

2. 各ノードは上位ノードから受け取ったUTILメッセージと、自分の変数の値域の値から、評価関数を用いて効用表を作成する

後退辺を含む場合は後退辺でつながっているノードも考慮



•後退辺の増加に伴いUTILメッセージサイズが指数関数的に増加
•制約密度の高いスケジューリング問題では顕著に現れる

問題の緩和による手法

DPOPを基とし,問題を緩和する2つの手法を提案

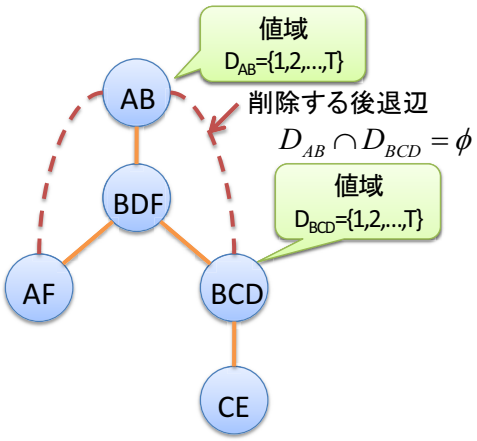
– **後退辺の削除**
- 特定の後退辺の削除を行う
- UTILメッセージサイズの指数関数的な増加を抑制する

– **問題の分割**
- 部分問題に分割することで問題の規模を小さくする
- UTILメッセージサイズを小さくする

提案手法1～後退辺の削除を用いた近似解法～

後退辺の削除

後退辺の数を減らしUTILメッセージサイズを削減する



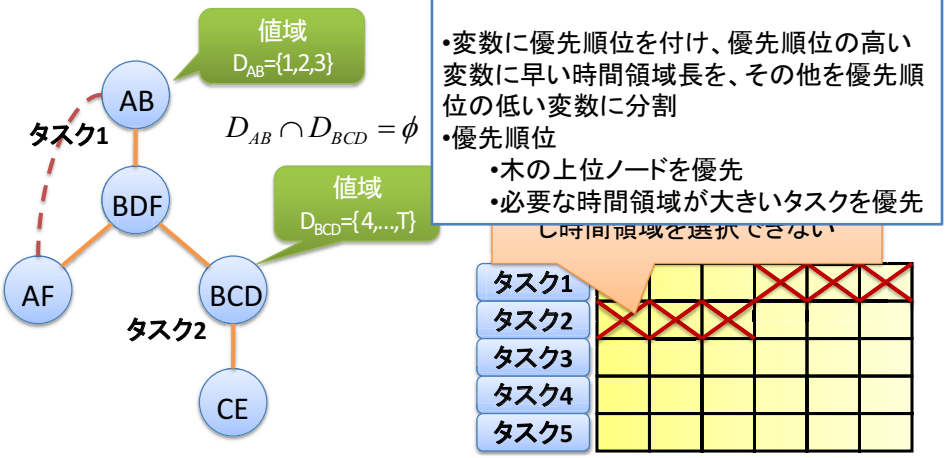
後退辺の削除の手順

1. 削除する後退辺の選択を行う
2. 選択された後退辺で繋がれているノードに対して値域の分割を行う
3. 選択した後退辺の削除を行う

提案手法1～後退辺の削除を用いた近似解法～

後退辺の削除

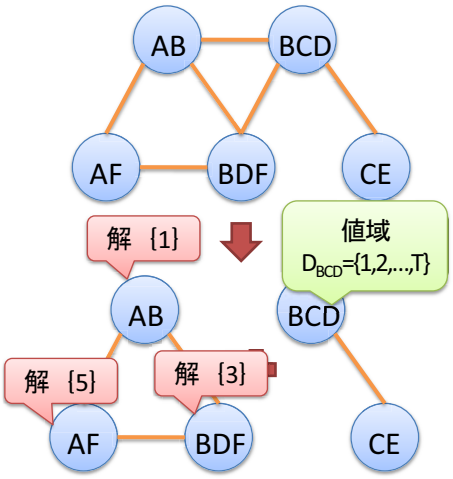
後退辺の数を減らしUTILメッセージサイズを削減する



提案手法2～問題の分割を用いた近似解法～

問題の分割

問題の規模を小さくしUTILメッセージサイズを小さくする



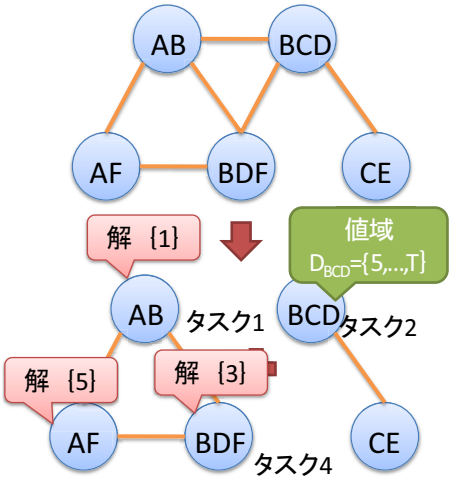
問題の分割の手順

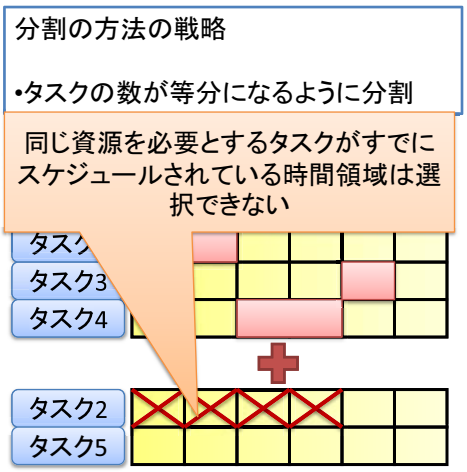
1. 問題を2つに分割を行う
2. 片一方のグラフの解探索を行う
3. もう片方のグラフで分割時に制約辺が削除されたノードに対して値域の変更を行う
4. もう片方のグラフの解探索を行う

提案手法2～問題の分割を用いた近似解法～

問題の分割

問題の規模を小さくしUTILメッセージサイズを小さくする





シミュレーションによる評価1

- 問題のサイズ(タスクの数)を変化させた時の結果を比較
- 各手法のパラメータと戦略
  - 提案手法1～後退辺の削除～
    - UTILメッセージの部分解に含まれる変数の数(ハイパーキューブの次元)を4に制限
    - 値域の分割の優先順位(木の上位ノードを優先)
  - 提案手法2～グラフの分割～
    - 初期の検討として2分割のみ評価
- スケジューリング問題は制約密度の違う2種類の問題を作成

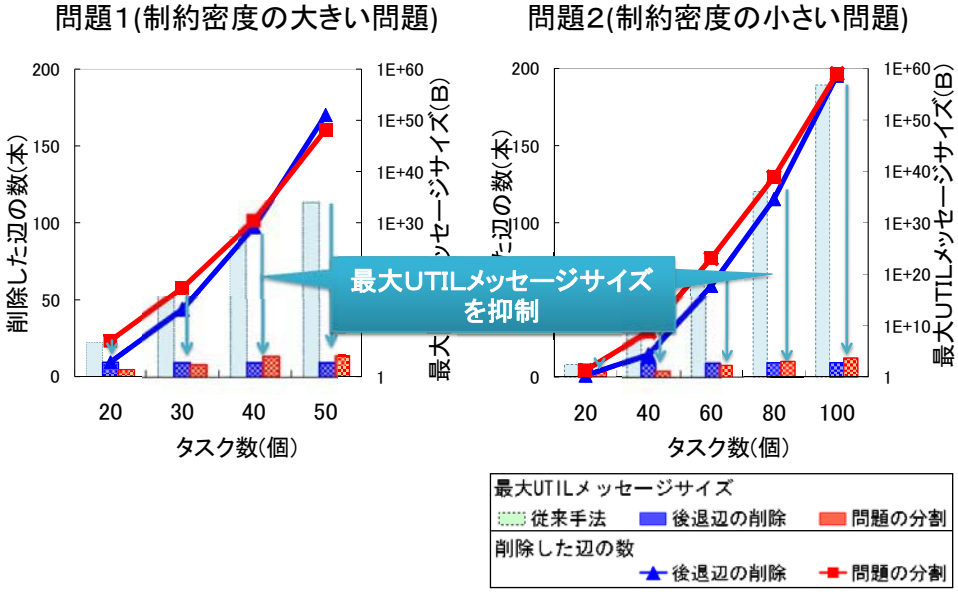
	資源数(個)	タスクに必要な資源(個)	タスクに必要な時間領域	タスク数(個)
問題1(比較的制約密度の大きい問題)	30	2	1～3(ランダム)	20,30,40,50
問題2(比較的制約密度の小さい問題)	50	2	1～3(ランダム)	20,40,60,80,100

シミュレーションによる評価1

- 評価対象
  - 最大UTILメッセージサイズ
  - 削除された辺（後退辺）の数
  - 解の精度：$\frac{\text{最適解の時間領域長}}{\text{提案手法により得られた実行可能解の時間領域長}}$
    - 実行可能解　：同じ時間領域内で同じ資源を使用しない解
    - 最適解　　：時間領域長が最少となる実行可能解

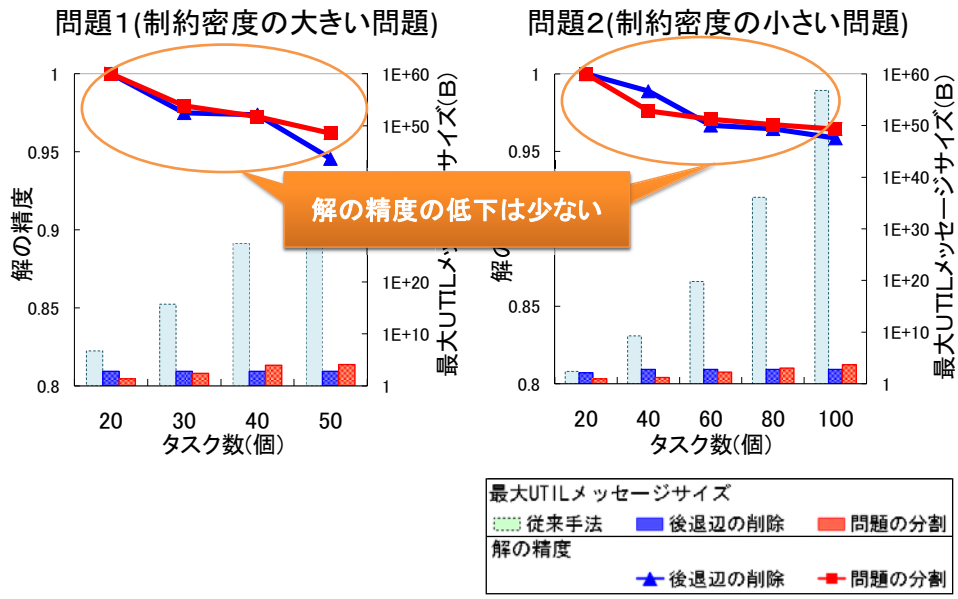
評価1の結果

削除した辺の数・最大UTILメッセージサイズの変化



評価1の結果

解の精度・最大UTILメッセージサイズの変化



シミュレーションによる評価2

- 提案手法1(後退辺の削除)とGreedy手法による解の精度の比較

  提案手法1の戦略
  - 戦略1:値域の分割の優先順位(木の上位ノードを優先)
  - 戦略2:値域の分割の優先順位(必要な時間領域が大きいタスクを優先)

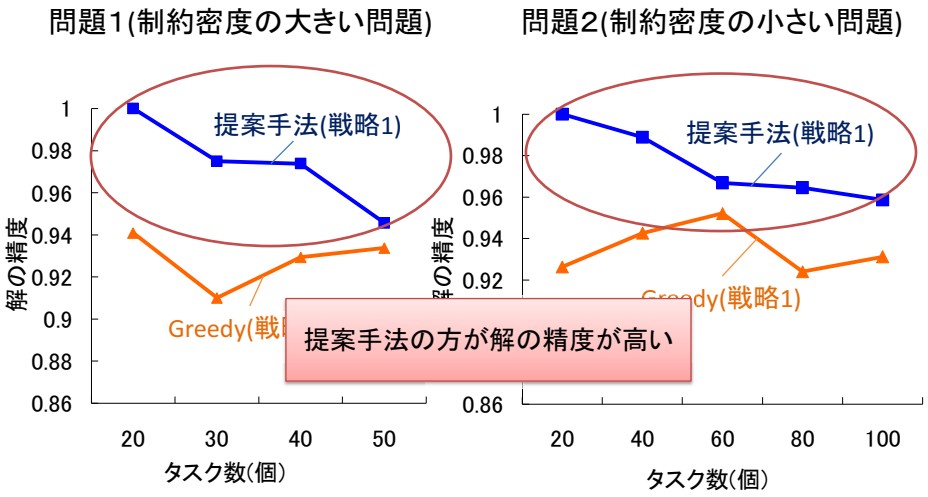
  Greedy手法の戦略
  - 戦略1:タスクの順番はランダム
  - 戦略2:タスクに必要な時間領域が大きい順にソート

- スケジューリング問題は評価1と同じ問題を使用
- 解の精度を比較

比較1	提案手法1(戦略1)	Greedy(戦略1)
比較2	提案手法1(戦略2)	Greedy(戦略2)
比較3	提案手法1(戦略1)	提案手法1(戦略2)

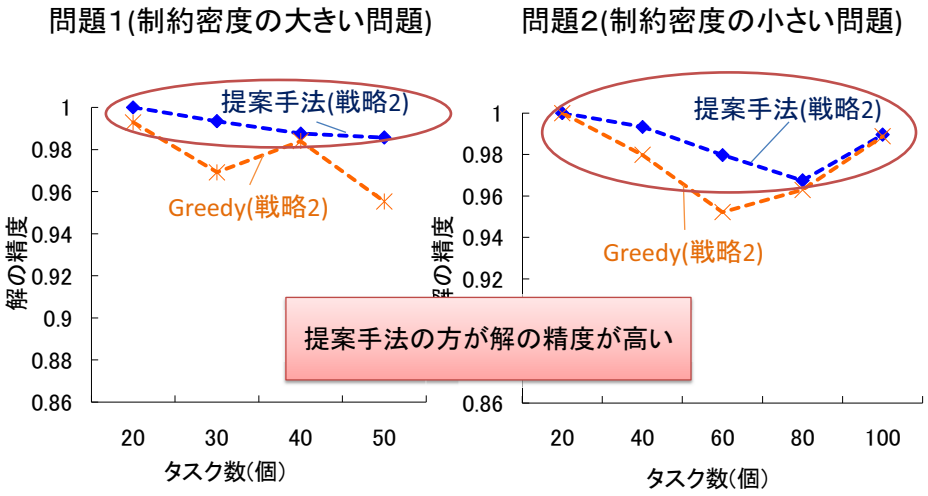
評価2の結果

提案手法1(戦略1)とGreedy手法(戦略1)の比較



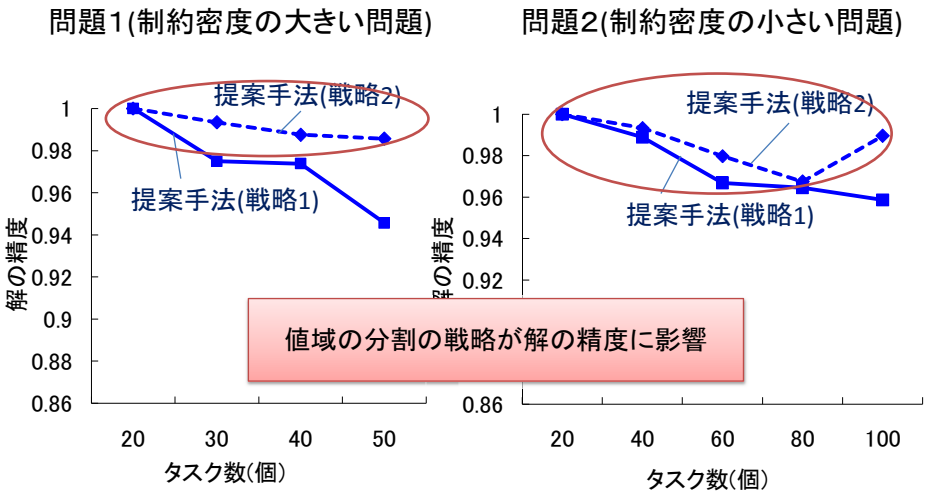
評価2の結果

提案手法1(戦略2)とGreedy手法(戦略2)の比較



評価2の結果

提案手法1(戦略1)と提案手法2(戦略2) の比較



まとめ

○まとめ

- 分散制約最適化問題をグリッド計算環境のコアロケーションのための分散スケジューリングに適用した
- 分散制約最適化手法の問題点に対して2つの緩和手法を提案した
- シミュレーションを用いた初期の評価により,緩和手法による解の精度の低下が小さいことを示した
- Greedy手法と比較して同等以上の精度の解を得た

○今後の課題

- 値域の分割、グラフの分割の戦略の検討
- 実質的なコアロケーションスケジューリング問題に適用
- より複雑なスケジューリング問題への適用

Cell プロセッサを対象としたデータ自動分割手法

種田 聡†

立命館大学大学院 理工学研究科†

近年，CPU ではシングルコアの微細化における消費電力，発熱量，サイズに対する性能比が限界に近付いている．そこで，微細化技術を使わず CPU の性能を向上させるマルチコア化が注目されている．マルチコアとはシングルコアに用いられている汎用プロセッサを 1 コアとして，デュアルコアもしくはクワッドコアで構成されている．この構成に対して IBM，Sony，東芝らは共同で PPE（Power Processor Element）と呼ばれる汎用プロセッサエレメントと，浮動小数演算に特化したプロセッサエレメント SPE（Synergistic Processor Element）を搭載するこれまでのマルチコアとは異なる構成の Cell Broadband Engine（以降，Cell）を提案した．

Cell は SPE を搭載していることでマルチメディアデータのリアルタイム処理や科学技術計算に向いたアーキテクチャである．現在，PPE を 1 基，SPE を 8 基搭載し，各エレメントをリングバスで接続している First-Generation CELL Processor がリリースされている．この Cell の SPE は SPU（Synergistic Processor Unit）と MFC（Memory Flow Controller）で構成されており，SPU は 128bit SIMD（Single Instruction Multiple Data）RISC アーキテクチャと領域サイズ 256KB の LS（Local Store）を搭載している．MFC は DMA（Direct Memory Access）転送を可能とするデータ転送エンジンであるが，アラインされていないデータの DMA 転送はサポートされておらず，転送サイズにも制限があり，さらに，16B より小さい転送サイズは 16B にアラインされるためオーバーヘッドが生じる．

また Cell アーキテクチャ特性を考慮したプログラムは，PPE と SPE それぞれの特性を活かした機能分割を行う必要があり，プログラムの演算がメインとなる部分を複数個ある SPE に対して MFC や LS を考慮したデータの分割を行い，SIMD 化させたタスクを SPE に割り当てる．これらのタスクが割り当てられた SPE を PPE で制御，管理し，処理能力の向上を行わなければならない．

現在，SPE や，PPE 向けの各コンパイラや，OpenMP，OpenMP ライクなシングルソースから PPE/SPE ソースを出力するコンパイラやプログラミングモデルが提案されている．しかし，逐次処理記述のソースから機能分割やデータ分割を自動で行うコンパイラが存在しない．そのためプログラマは機能分割とデータ分割，割り当て，アーキテクチャ特性を考慮したプログラムを作らなければならない．そのため，Cell は敷居の高いプロセッサとなっている．

そこで，各 SPE に逐次処理記述のソースから主な演算となる中粒度のループ部分を解析し，MFC のサポート範囲でかつ，LS のデータ領域を考慮し，DMA 転送を効果的に用いたデータの分割と割り当てを行い，複数の SPE で並列処理させる．そしてこの並列タスクを PPE で制御，管理するソースを生成する Cell 向けのコンパイラを作成することを目的とする．このコンパイラをプログラマが用いることで，Cell のアーキテクチャ特性と並列手法を意識せず Cell の演算性能を引き出せるプログラム環境をプログラマに提供することができる．この目的を果たすコンパイラには中粒度並列化，マルチスレッド化，SIMD 化が必要となり，データ分割においては Cell の構成に合った手法が必要となる．そこで，本発表では Cell を対象としたデータ自動分割手法を提案し，この提案手法により分割されたデータから既存変換手法の中粒度並列化，マルチスレッド化，SIMD 化を行う．

本発表で，逐次処理記述のソースから PPE/SPE ソースを出力するための Cell を対象としたデータ自動分割手法について述べる．この手法は逐次処理記述のソースを解析するコンパイラに実装し，Cell のアーキテクチャ特性と並列手法を意識せずに Cell の演算性能を引き出せるプログラム環境を提供する．

Automatic Data Distribution Scheme for Cell Processor

†Satoshi Taneda

†Graduate School of Science and Engineering, Ritsumeikan University

Cellプロセッサを対象とした データ自動分割手法

種田　聡
立命館大学 大学院 理工学研究科
國枝・桑原研究室
e-mail：rs027026@se.ritsumei.ac.jp

目次

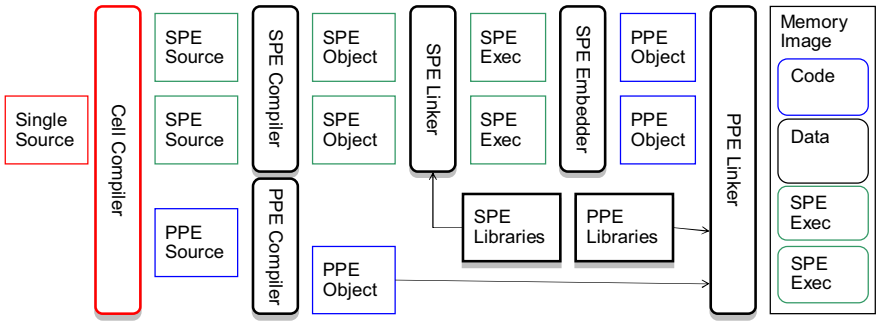
- 研究背景・目的
- 研究概要
- Cellの構成
- Cellのプログラミングモデル
- Cell データ分割手法
- まとめ

研究の背景

- シングルコアの限界
  - 電力消費，発熱量，サイズに対する性能
- マルチコア
  - AMD Opteron Processor
  - Intel Core 2 Duo Processor
  - Cell (Cell Broadband Engine)
    - 特殊なマルチコア
      - 機能別に違うコアを1チップに搭載
    - 浮動小数点演算能力
      - 200GFlops以上
    - コンパイラの手法が不十分
    - プログラマの負担が大きい

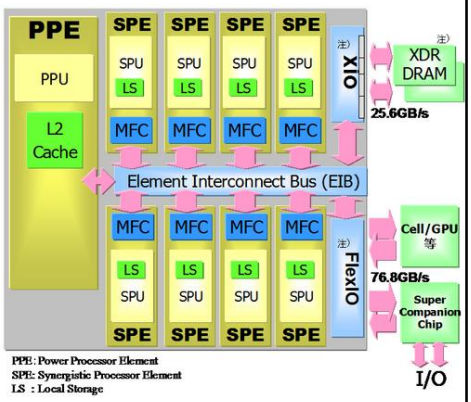
研究の目的

- Cellコンパイラの開発
  - Cellのアーキテクチャ特性と並列化手法を
    意識せずに書かれた**逐次言語のソース**から
    Cellの演算能力を引き出せるコードを生成



Cell の概要

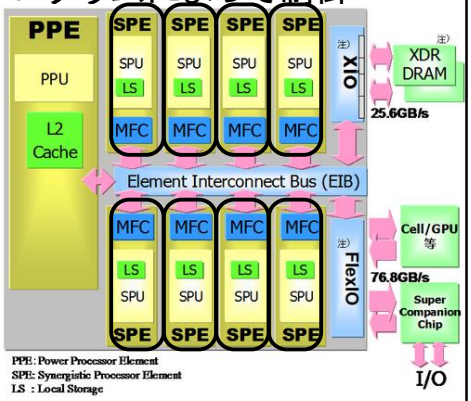
- ヘテロジニアスなマルチコアプロセッサ
  - 汎用コア，演算専用コア
- The First-Generation CELL Processor
  - 汎用コア1基
  - 演算専用コア8基
  - 実用例
    - PlayStation 3
    - IBM Blade Server



http://www.cellusersgroup.com/modules/product/toshiba-cell-chip-set.php

SPE の概要

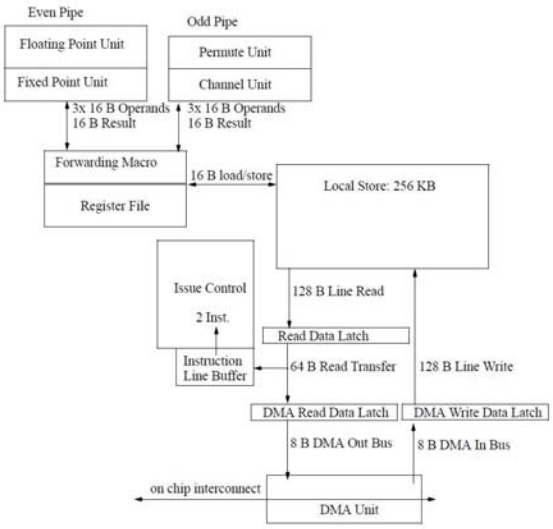
- Synergistic Processor Element
  - 128bit SIMD RISC アーキテクチャ
  - SPE用にコンパイルされたバイナリが動作
  - PPEで動作するOSやユーザプログラムによって制御
- レジスタセット
  - 128 bit長
  - 128本
- LS: Local Store
  - 命令とデータを格納
  - 256KB



http://www.cellusersgroup.com/modules/product/toshiba-cell-chip-set.php

SPE の構成

- インオーダ型
- ２本の非対称パイプライン
  - 演算系: Even
  - ロードストア系: Odd
- 深い構成のパイプライン
  - 分岐ミスのペナルティが大きい
- アクセスサイクル
  - レジスタ　　:1 cycle
  - Local Store :1+6 cycle
  - メインメモリ：数百cycle
- キャッシュは搭載されていない



MFC の構成

- Memory Flow Controller
- メインメモリとLS間に関するメカニズムを提供
  - データ転送 (DMA)，保護，同期
  - 但し，アラインされていないデータはサポート外
- 転送可能サイズとアラインメント

転送サイズ	アライン	特徴
1, 2, 4, 8byte	転送するbyte境にアラインメント	転送には非効率
16byteの倍数で最大16KBまで	16byte境界にアラインメント	分割転送は非効率

Cell のプログラミングモデル

- 2種類のコア特性を活かして機能分担
  - PPE：制御，管理
  - SPE：演算
- 従来のマルチスレッドプログラミングモデル
  - PPE：メインスレッド
  - SPE：演算スレッド
- 複数ある演算コア向けに並列化
  - データ分割
    - データ量が多く単純な処理に有効

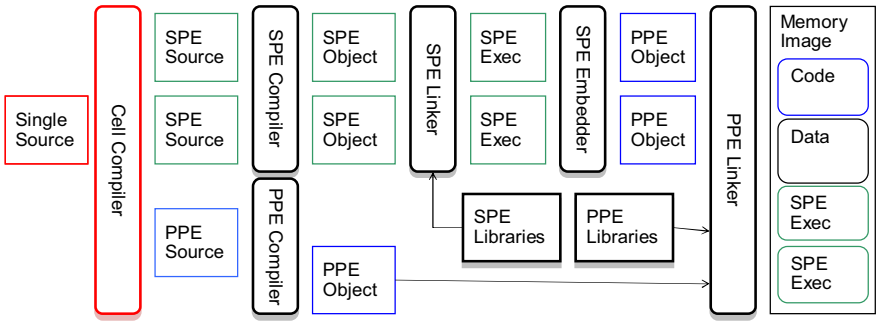
Cell Compiler

Cell Compiler

- 逐次言語を解析
- 各SPEに中粒度のループタスクを割り当て
- PPEで各SPEを制御，管理
- PPE/SPEソースを生成

手法

- 中粒度並列化
- マルチスレッド化
- SIMD 化
- Cell 向けデータ自動分割

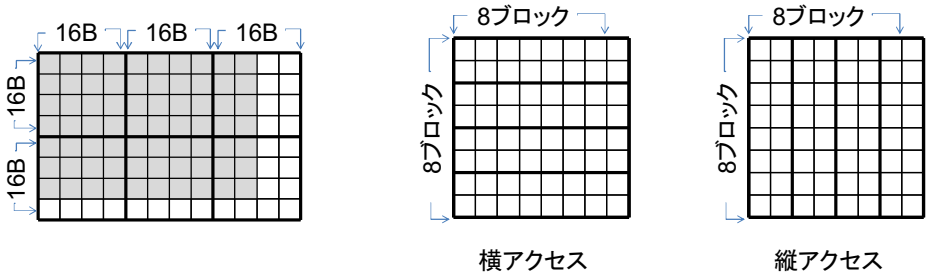


データ分割の問題点

- 各SPEにデータを分割
  - LSサイズの256KB内に収めなければならない
- 分割されたデータを転送
  - 転送サイズに合わせたアラインメント
  - 転送サイズに合った境界からしか転送できない
  - 16byteまでは転送にむだがある
- SPEで演算
  - 128bit長 (16byte) に収まらなければSIMDの効果が弱まる
  - 128bit長 (16byte) レジスタを128本備えている
  - 深いパイプライン
    - SIMD命令のループをアンローリング

データ分割手法 –アライン

- n次元配列の分割
  1. 先頭アドレスを16byte境界でアラインメント
  2. 全ての次元に対して16byteでアラインメント
  3. (16byte × n)を1ブロックとする
  4. ブロック単位で均等にSPEに割り当てる
     - アンローリングを考慮



データ分割手法 -オーバレイ

- 並列実行ループで使用される変数のリスト作成
  - Input, Output, Input/Outputに分ける
    - Input: 演算処理に必要となる変数
    - Output: 演算結果を代入する変数
  - あるループの回転数で必要となる領域を算出
    - Ex. ループ1回転で

      int 型変数が i 個，float 型の配列が j 個とする
      - size (n) = i × INT_SIZE + n × j × FLOAT_SIZE

13

データ分割手法 –オーバーレイ

- 割り当てられた変数の領域がLSを超える場合
  1. LSに割り当て可能な範囲のinputを転送
  2. LSに割り当てられたデータを処理
  3. 処理したoutputを転送
  4. 不要になったinputとoutputを解放
  5. 新たなinputを転送して割り当てる

データ分割手法 –ダブルバッファ

- 転送が遅延の原因になる場合
  1. LSにinputをバッファ1に転送
  2. LSにinputをバッファ2に転送指示
     1. Inputをバッファ2に転送中
     2. バッファ1を処理
  3. LSにinputをバッファ1に転送
     1. Inputをバッファ1に転送中
     2. バッファ1を処理

まとめ

- Cellの構成
  - SPU
  - MFC
- Cell データ分割手法
  - アラインによるブロック化
  - アンローリングを考慮した分割
  - オーバーレイ
  - ダブルバッファ
- 今後
  - 境界処理の問題を検討
  - 単純な配列演算からレイトレイシング等様々な角度から検証
  - コンパイラインフラストラクチャとしてCOINSを用いて実装

参考資料

- Cell Broadband Engine Architecture Version 1.01
  - Sony Computer Entertainment Incorporated
  - http://cell.scei.co.jp/pdf/CBE_Architecture_v101_j.pdf
- The Design and Implementation of a First-Generation CELL Processor
  - Dac Pham, et al.
  - ISSCC, 2005, Pages 184-185
- A Streaming Processing Unit for a CELL Processor
  - B Flachs, et al.
  - ISSCC, 2005, Pages134-135
- COINS
  - http://www.coins-project.org/

携帯電話 Java 環境向け XML データベースコンポーネントの開発

佐々木 悠†
東京農工大学大学院工学府
情報工学専攻 †

並木 美太郎‡
東京農工大学大学院
共生科学技術研究部 §

1 はじめに

1.1 背景

携帯電話に Java 実行環境が搭載され，端末の進化に伴いその性能は強化されてきた．携帯電話 Java はセキュア且つ，通信機能を備えたアプリケーションプラットフォームとしても，注目を浴び広く普及している．しかしながら，開発されるアプリケーションは，いまだに，ゲーム等のエンターテインメント系のものが主流であり，携帯電話の通信機能を生かした実用的なアプリケーションは少ない．

一方，Web 上では Web サービスのデータフォーマットとして XML が普及している．XML は，Web サービス間における相互のデータ交換をはじめ，クライアント指向の Web アプリケーションであるリッチクライアントにおけるデータフォーマットとしても利用されている．

1.2 目的

現在の携帯電話 Java 実行環境では，移植性・機能性に乏しい API と，使用できる資源の少なさが原因で，通信機能を活用した XML をデータフォーマットとする実用アプリケーションの開発が困難である．そこで，本研究ではこの問題点を解消し，XML をデータフォーマットとする携帯電話 Java 用アプリケーションの開発を支援するために，携帯電話 Java 向けの XML データベースコンポーネントを開発する．

2 XML データベースコンポーネントの概要

本コンポーネントは，端末内記憶領域への XML データベースの構築・アクセス及び，インターネット上に存在する，XML データベースサーバへのアクセスの機能をプログラマに提供する．インターフェースは，XML:DB API 仕様のサブセットを用いる．XML:DB API 仕様は，XML:DB Initiative が策定した XML データベース API 仕様で，実質的に標準仕様となっているものである．この仕様を採用することによって，プロファイル独立で，利用しやすい API をプログラマに提供することができる．

システムの全体構成を，図 1 に示す．プログラマは，XML:DB API 仕様のサブセットの API を用いて，本コンポーネントにアクセスすることができ，携帯電話端末内のストレージに構築したデータベースと，インターネット上に存在するデータベースサーバを等価的に扱えることが特徴である．

3 実現方法

3.1 端末内データベースの構築方法

XML データベースの，コレクション・リソースの概念は，ファイルシステムの，ディレクトリ・ファイルの概念と等価である．よって，XML データベースを構築するためには，コレクション・リソースを，それぞれ，ディレクトリ・ファイルにマッピングすればよい．携帯電話 Java には，ディレクトリ・ファイルの概念が存在しないため，先行研究であるファイルシステムライブラリ FML[1] を用いて，端末内ストレージに，ディレクトリツリー形式のファイルシステムを構築した．FML で構築したディレクトリとファイルに，コレクションとリソースをマップすることで，端末内データベースの構築を実現した．

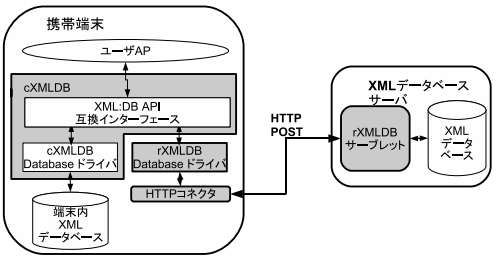


図 1: システムの全体構成

3.2 インターネット上のデータベースへのアクセス

携帯電話 Java の通信には，HTTP プロトコルのみ利用可能という制限が存在する．また，対象とするデータベースによって，アクセス方法が異なるという問題もある．これらの問題は，データベースサーバに，サーブレットを設置する手法で解消した．携帯電話 Java とサーブレット間のデータ転送は，独自に定めたプロトコルを HTTP POST に載せる方式で実装した．

3.3 実装

本コンポーネントのサイズは，34.5KB となり，携帯電話 Java アプリケーションに組み込むには十分軽量なサイズとなった．データベースアクセスに必要な機能も実装できており，実際に，端末内部データベースとインターネット上のデータベース間で XML データをやりとりするサンプルアプリケーションを開発することができた．

3.4 終わりに

本研究では，携帯電話 Java 向け XML データベースコンポーネントを開発することができた．これにより，XML をデータフォーマットとする携帯電話 Java アプリケーションの開発を可能とした．今後の課題としては，本コンポーネントの有用性を示せるサンプルアプリケーションの開発と，詳細な性能測定が挙げられる．

参考文献

[1] 小高健二, 阿部 大将, 山口 真吾, 並木 美太郎:携帯電話用 Java 実行環境における XML 処理のためのコンポーネント, 情報処理学会シンポジウム論文集,Vol.2005, No.6, pp.717-720 (2005.7) .

[2] XMS System Programming Guide, http://www.xmldb.jp/

A Development of an XML Database Library for a Java Runtime Environment on Cellular Phones.

† Yuh Sasaki
Tokyo University of Agriculture and Technology
§ Mitaro Namiki
Tokyo University of Agriculture and Technology

携帯電話Java環境向け
XMLデータベースコンポーネントの開発

2007年9月13日
東京農工大学大学院
並木研究室 佐々木 悠

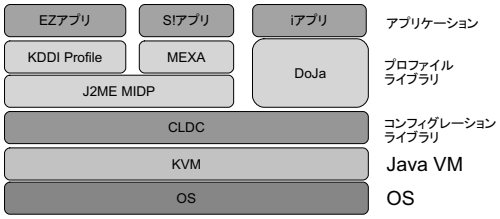
背景

- 携帯電話Java実行環境の進化
  - セキュア且つオープンなプラットフォーム
- 携帯電話で動作する実用アプリケーションの要求
  - マルチメディアデータベース
  - Webサービス，コンテンツと連携
- Web上でXMLの広がり
  - Webサービス・リッチクライアントのデータフォーマット

XMLをデータフォーマットとする携帯アプリを開発したい

携帯電話Javaアーキテクチャ(1)

- キャリアによってAPIに差異がある
- 共通仕様のCLDCは、J2SEから多くのクラスを削減



携帯電話Javaアーキテクチャ(2)

- ヒープメモリ
  - 1～10MB
- アプリケーションサイズ
  - 10KB～1MB
- ストレージサイズ
  - 10KB～1MB
- 通信制限
  - HTTPのみ利用可能
  - DoJaは接続先に制限あり

問題点

- 移植性、機能性に乏しいAPI
  - プロファイル毎のAPIの差異
  - DOM,SAX等のXML関連のAPIは使えない
- 限られた資源

XMLをデータフォーマットとする
アプリの開発が困難

先行研究

- XMLパーサ
  - cDOM
  - CSAX

XMLのパース
データ入出力
環境は整っている

- ファイルシステム
  - FML

XMLデータを手軽に入出力できる環境を
整えたい→XMLデータベース

XMLデータベース

- RDB型XMLデータベース
  - 文章をそのまま格納（直接型）
  - RDBのカラムにマップする（マッピング型）
  - インデックスを作成（ハイブリッド型）
- ネイティブXMLデータベース
  - 木構造を崩さずに格納
  - 高パフォーマンス

ネイティブXMLデータベースを対象とする

目的

- 携帯電話Javaで動作する、XMLをデータフォーマットとするアプリケーションの開発を支援する
- XMLデータベースコンポーネントを開発し、XMLデータの保存・検索・編集等の機能を実現する
  - ネット上のXMLデータベースサーバとの連携
  - 端末内へデータベースの作成
  - XPathを使った検索

設計方針

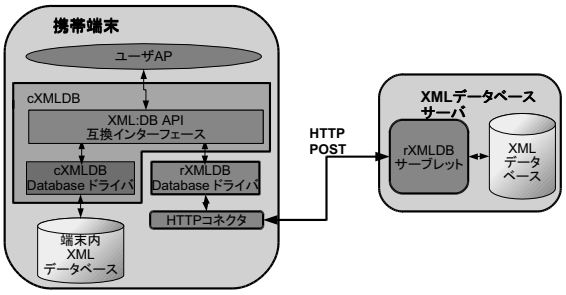
- 軽量・省メモリ
- 最小限の機能の実装
- プロファイル独立のAPIを提供

- XML:DB API仕様のサブセットを利用
- JARで50KB以下のサイズ、1MB以下のヒープメモリ

XMLデータベース コンポーネントの概要

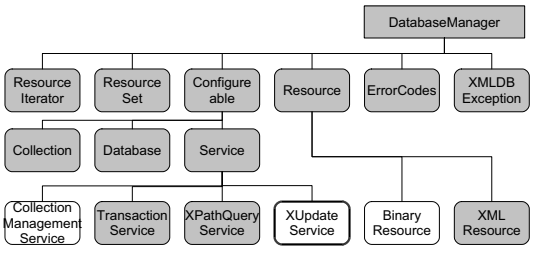
- XMLデータを保存・検索・更新するためのコンポーネント
  - 保存・検索にはXPathを利用可能
  - データ更新にはXUpdateを利用可能
    - 現状はDOMを利用
- XML:DB APIを共通のインターフェースとして端末内部のDBと、外部のDBを等価に扱える

システム構成



API仕様

- XML:DB API仕様のサブセットを利用
  - XML:DB Initiative 策定の実質的標準仕様



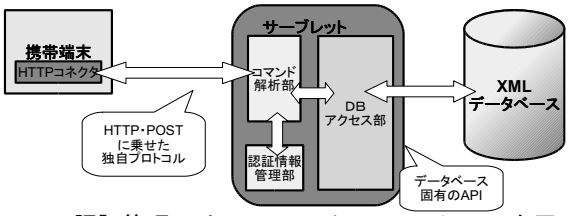
端末内部DB(cXMLDB)実現方法

- 『コレクション・リソース』を『ディレクトリ・ファイル』に置き換え、先行研究のFMLを用いて実現
- XPathは、仕様を削って実装
  - 述語の簡略化
  - 関数の大幅な削除

端末外部DB(rXMLDB)実現方法

- 外部データベースサーバにサーブレットを設置し、HTTPコネクタ・サーブレット間をPOSTに乗せる独自プロトコルでやりとり
  - 『HTTPのみ』という制限をクリア
- XPathクエリーは外部データベースへ投げる
  - 端末側でパースしない
  - XPath仕様をフルに利用可能
- HTTPコネクタ部でプロファイルの差異を吸収

端末とサーブレット



- 認証管理にはJava Servletのセッションを用いる
- コマンド解析部、認証情報管理部はデータベースの種類に依存しない

実装結果(1)

- ライブラリサイズ：
  - cXMLDB:24.3KB
  - rXMLDB:19.8KB
  - cXMLDB + rXMLDB:34.5KB
  - Servlet : 4KB (Apache Tomcat 5.x)
    - 携帯Javaに載せるには十分小さなサイズ
    - XUpdateServiceは未実装

実装結果(2)

- 機能
  - 端末内ストレージにXMLデータベースを構築
  - XML:DB API仕様のインターフェースによる、端末内ストレージ、インターネット上のXMLデータベースへのアクセス
  - XPathによるデータの検索

サンプルアプリケーション

- 外部・端末内，両方のデータベースを使用
- 携帯電話の機能を生かす
- 『携帯メモ帳』
  - 画像・バーコード・テキスト に任意の見出し，コメントをつけて格納
  - 外部DBから拾ったデータは、内部DBにキャッシュされる　→　オフラインでデータ閲覧可能
  - 任意の単語で検索が可能

まとめ

- データベースコンポーネントcXMLDB, rXMLDBを開発した
  - 携帯Javaアプリで、XMLデータの保存・検索・更新を実現
- XMLをデータフォーマットとするアプリケーションの開発を可能にした
- 外部アプリケーションや、ストレージとの連携を可能にした

今後の予定

- XUpdateの仕様を確定させる
  - スペック的に可能な範囲で実装したい
- トランザクション機能実装
  - 現在作業中
- アプリケーションの作成
  - もう少し使いやすく改良していきたい
- 性能測定
  - Xpriori以外のデータベースとベンチマークを用いて、性能測定を行いたい

マイグレーション可能な携帯電話向け Scheme 言語処理系

松崎 泰裕† 並木 美太郎‡
東京農工大学大学院情報工学専攻 †/東京農工大学大学院共生科学技術研究院 ‡

1 はじめに

近年の携帯電話は Java VM を搭載しており，携帯電話上で動作する応用プログラム（以下 AP）を自由に開発できる．しかし，使用できる言語は Java に限られている．このことは Java 言語以外のプログラマに対して携帯電話向け AP 開発の敷居を高めていると言える．また同じ Java 言語であっても，キャリアによってプロファイルが異なり，AP は各キャリアに依存するものとなっている．

さらに携帯電話の普及により携帯端末は一人一台持つものとなっているが，それら複数の端末に存在する情報やハードウェア資源を利用するシステムの開発方法には，モバイルエージェントの仕組みを用いた開発方法が挙げられる．モバイルエージェントでは，状況に応じてサーバーや携帯端末間でプログラムのマイグレーションを行うが，携帯電話の Java 言語では対応しておらず，開発基盤が必要である．

携帯電話で動作する言語処理系には既存のものがいくつか存在し，例えば Scheme の JAKLD[1] が挙げられる．しかし，携帯電話の機能を操作する API が提供されていない，また末尾呼び出しの最適化や継続に非対応であるなどの問題がある．

そこで本研究では，携帯電話の Java VM 上で動作するマイグレーションに対応した Scheme 言語処理系の設計と実装を行う．

2 本研究の目標

本研究では携帯電話の Java VM 上で動作する Scheme 言語処理系を開発し，Scheme 言語を用いて携帯電話向け AP を開発する環境を提供する．Scheme は関数型言語であり，また継続をファーストクラスオブジェクトとして扱えるため，モバイルエージェントの仕組みを用いた開発に適している．本処理系の核部分は PC でも動作するものとする．Scheme の言語仕様は R5RS[2] に準拠する．

本処理系では，携帯電話の液晶画面やネットワーク機能を扱うための手続きを提供し，GUI，グラフィックやネットワークを用いたプログラムを Scheme によってプログラミング可能とする．これらの API はプロファイルにかかわらず共通とし，プロファイルの差を本処理系で吸収する．また，継続を含めた任意の Scheme オブジェクトを携帯電話のネットワークを通してマイグレーションすることに対応し，携帯電話や計算機間をマイグレーションするモバイルエージェントシステムなどに応用する．

3 本処理系の概要

本処理系の全体構成は図 1 のとおりである．本処理系では，Scheme プログラムをあらかじめ計算機上で作成して本処理系の JAR パッケージへ格納して実行する方式の他に，携帯電話上で Scheme プログラムを入力して実行する方式をサポートする．ただし今回の実装では携帯電話上でプログラムを入力するためのエディタの実装は行わず，今後の課題とする．

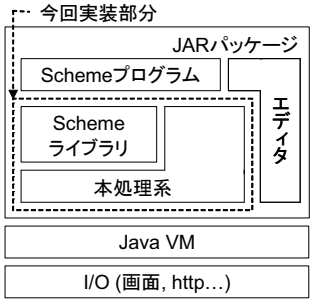


図 1: 本処理系の全体構成

プログラムを処理する方式には，携帯電話で JAR パッケージのサイズが制限されること，また動的に生成した Java バイトコードを実行できない制限を考慮し，S 式をそのまま辿りながら実行するツリートラバース型インタプリタ方式を採用する．

4 設計

4.1 データ表現

本処理系では，すべての Java クラスの基底である Object クラスで S 式を表わす．各データを表わすクラスは，図 2 のとおりである．点線で囲まれたクラスは Java の標準クラスであり，これらを多く用いることにより，処理系サイズの低減を行う．

真偽値，文字や数値は対応する Java の各プリミティブ型の配列で保持する．Java には各プリミティブ型を保持するための標準クラスが存在するが，よりアクセスが高速な配列を採用する．

4.2 評価器

式の評価は，評価対象の式の各要素を部分式として再帰的に評価し，先頭要素の評価結果に応じて処理を行う．Scheme には，継続をファーストクラスオブジェクトとして扱う機能があり，任意のタイミングでのコンテキスト情報のコピーや復元が必要となる．本処理系ではこの継続をマイグレーションし，別の端末で実行することに対応する．この機能を実現するために，再帰呼び出しを用いずにループのみで処理をし，処理される部分式のコンテキスト情報をスタックではなくヒープ領域へ格納する．本処理系では Continuation クラスのインスタンスがリストを構成することで，各部分式の評価におけるコンテキスト情報を保持する．

4.3 マイグレーション

現在の携帯電話は端末間で直接通信することができないため，マイグレーションを行う場合には，まずマイグレーション元の端末で http 通信を用いてサーバーへオブジェクトを送信し，次にマイグレーション先の端末で http 通信を用いてオブジェクトを受信すること

Scheme Interpreter for Cellular Phones which can migrate programs

† Yasuhiro Matsuzaki
Department of Computer and Information Sciences, Tokyo University of Agriculture and Technology
‡ Mitaro Namiki
Graduate school of Engineering, Tokyo University of Agriculture and Technology

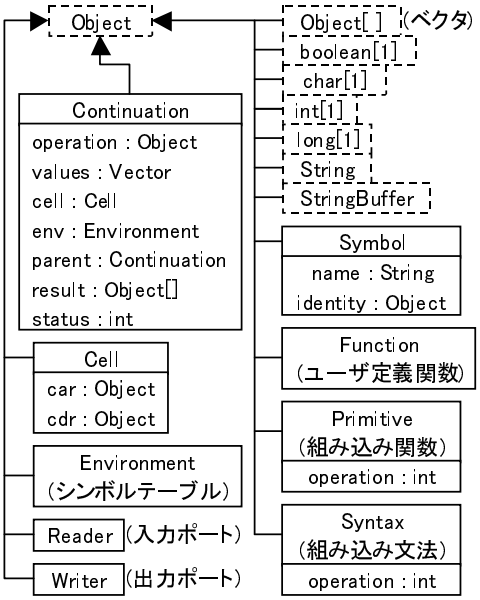


図 2: 各データを表わすクラス

表 1: 携帯電話上での実行速度

コード	本処理系	JAKLD
tak	189s	27s
単純ループ	746μs	-
関数付ループ	1.468ms	-
継続ループ	764μs	-
do 構文ループ	849μs	76μs

表 2: PC(Pentium4-3GHz) 上での実行速度

コード	本処理系	JAKLD	SISC[3]
tak	406s	491ms	80ms
fib	12.2s	17.5s	3.52s
単純ループ	3.73μs	-	0.89μs
関数付ループ	4.89μs	-	1.39μs
継続ループ	4.82μs	-	6.50μs
do 構文ループ	4.51μs	3.81μs	0.88μs

でマイグレーションを行う．

オブジェクトを http 通信を用いて転送する際には，オブジェクトを http 通信で扱えるバイト列に変換する必要がある．本処理系では，オブジェクトからたどれる全てのメンバを再帰的にシリアライズすることによってマイグレーションを実現する．循環オブジェクトに対応するために，シリアライズの際には入出力順にオブジェクトに番号を付け，既出のオブジェクトはその番号で参照する．

4.4 携帯電話用 API

携帯電話の機能を操作するための API は，Java で提供されるものと同様のものを提供する．ただし Java と違い，Scheme はオブジェクト指向ではないため，作成されたオブジェクトを引数に渡す形で操作を行う．

GUI やネットワーク通信処理における非同期イベントの処理は，Scheme における procedure を登録する形で記述する．Scheme における procedure には組み込み手続きやユーザ定義手続き，継続が含まれるため，プログラマはコールバック関数や継続などで処理を記述できる．

5 実装と評価

実装を行った結果，Scheme で書かれたライブラリなどを含めた JAR パッケージサイズは約 55 KB となった．またファイルシステムライブラリである FML を追加した合計のサイズは，約 73 KB となった．本処理系では新たにクラスを定義せずになるべく Java の標準クラスを利用することで省サイズを実現した．この 55 KB というサイズは，一般的な携帯電話のサイズ制限である 100 KB の約半分であり，処理する Scheme プログラムを JAR パッケージに埋め込む十分な余地が残っている．また JAKLD の約 40KB と比較しても，継続などの機能を追加した上で同程度のサイズを実現できた．

携帯電話の実機上で本処理系を動作させ，メモリ使用量を測定した．測定直前に Scheme 上から Java の GC を起動し，その後に全メモリ容量と使用量を取得することでメモリ使用量を算出した．起動時におけるメモリ使用量は，約 230 KB，長さ 1000 のリストを作成した時の消費メモリは約 20 KB となった．これは，数 MB〜10MB のヒープメモリを搭載している最近の携帯電話において問題ない消費量である．

次に携帯電話と PC 上で実行速度を測定した．比較のために，Java で記述された他の 2 つの Scheme 処理系においても測定を行った．JAKLD は継続と末尾呼び出しの最適化に対応していないため，一部の動作しないコードは測定できていない．結果は表 1 と表 2 のとおりとなった．ループは 1 ループあたりである．携帯電話向けとして問題ない速度を確保できている．携帯電話上では一部のコードで JAKLD より遅い結果が見られるが，PC 上では同等，または逆に速い結果が見られる．これはコンテキスト情報の格納先がスタックとヒープで異なるため，Java VM の性能差がでていると考えられる．SISC と比較すると速度の差が見られるが，これは Scheme プログラムの処理方式の違いによるものと考えられる．

6 おわりに

本論文では，マイグレーションに対応した携帯電話向け Scheme 言語処理系の設計と実装について述べた．本処理系により Scheme 言語による携帯電話向け AP の開発が可能となった．またマイグレーションに対応したことにより，モバイルエージェントの仕組みを用いた AP の開発が可能となった．

今後の課題は，携帯電話上で Scheme プログラムを入力するためのエディタの開発，マイグレーションを用いた分散処理や情報処理などの応用システムの開発である．

参考文献

[1] 湯浅太一: Java アプリケーション組み込み用の Lisp ドライバ, 情報処理学会論文誌, Vol.44, No.SIG4, pp.1-16 (2003).

[2] Richard Kelsey, William Clinger and Jonathan Rees, editors: The revised5 report on the algorithmic language Scheme. In Higher-Order and Symbolic Computation 11(1), pp.7-105 (1998).

[3] Scott G. Miller and Matthias Radestock: SISC for Seasoned Schemers, http://sisc.sourceforge.net/ (2006).

マイグレーション可能な携帯電話向けScheme言語処理系

JSASS 2007年09月13日

東京農工大学 工学府 情報工学専攻
松崎 泰裕
並木 美太郎

背景(1/2)

- 携帯電話向けのプログラミング環境
  - JavaやC言語が既存
- 言語がJava・C言語限定
- 各キャリアでAPIが非統一
- 携帯電話上でプログラミングが困難

背景(2/2)

- 携帯電話の特性
  - 一人一台持っている端末
  - 各端末に資源が存在

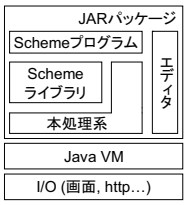
  →マイグレーション/モバイルエージェント
    - 情報検索、情報共有(交換日記)、AIプログラム(ペット)
- 携帯電話で動作する言語処理系
  - BASICやSchemeが既存
  - JAKLD
    - 携帯用APIや継続に非対応
    - マイグレーション不可

目的

- R5RS準拠Scheme言語処理系を開発
  - Java VM上で動作する処理系として開発
  - 携帯電話だけでなくPCやサーバーでも動作
- 携帯電話用APIの提供
  - グラフィック、GUI、ネットワーク
  - 各キャリアに対応してAPI差を吸収
- 継続などのマイグレーションを実現
  - モバイルエージェントシステム
    分散・情報処理システムへ応用

全体構成

- Schemeプログラム
  - JARパッケージへ格納
  - 携帯電話上で入力
- 二次記憶へのアクセス
  - ファイルシステムライブラリを利用
    - プロファイルによる差を吸収



評価器での実行方法

方式1:ネイティブにコンパイルして実行
- 携帯のJava VMでは実現不可能

方式2:中間コードにコンパイルしてVMで実行
- 処理系のサイズ制限の問題

方式3:ツリートラバース型インタプリタ
- 速度に欠けるがサイズが小さい

- 携帯電話のJavaのサイズ制限を考慮
  - ツリートラバース型を採用

オブジェクト設計

- Javaの継承関係で表現
- 携帯電話のJavaはサイズ制限が厳しい
  - 全クラスの基底のObjectクラスでS式を表現
    - 既存クラス使用でコードサイズ削減

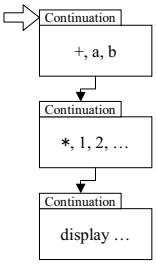
Schemeアトム	Javaオブジェクト	説明
空リスト	Object	ある特定のインスタンス
シンボル	Symbol	独自に定義するクラス
真偽/文字/数値	boolean[]/char[]/int[]/long[]	Javaプリミティブ型の配列
ベクタ	Object[]	Java標準クラスの配列
文字列	String/StringBuffer	Java標準クラス

式の評価方法(1/2)

- 単純な部分式の評価方法
  - 再帰呼び出しで処理
    - コンテキスト情報はスタックに格納
    - Javaではスタックへアクセス不可能
      - 継続に対応不可能
- 解決策
  - ループのみで処理
    - コンテキスト情報をヒープへ格納
    - コンテキストを評価器のスタックと分離

式の評価方法(2/2)

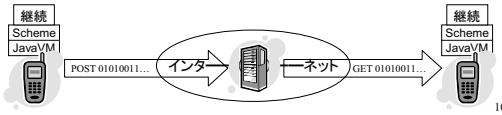
- コンテキスト情報がリストを構成
  - 継続マイグレーション
    末尾最適化に対応
- Continuationクラス
  - 命令(評価したリスト先頭)
  - 評価済み要素
  - 未評価残り要素
  - 環境(シンボルテーブル)
  - 親の継続



9

マイグレーション

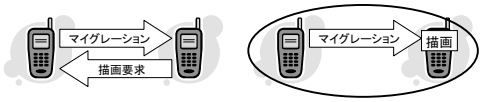
- 任意オブジェクトのマイグレーションに対応
  - 継続・画面I/O・HTTP通信Port・二次記憶入出力Port
- 対象オブジェクトからたどれる全オブジェクトを
  バイナリにシリアライズ
  - 順に番号をつけ循環参照に対応
- HTTP通信でサーバーを中継して転送
- 携帯間のみでなく、携帯とPC間も可能



10

マイグレーション(画面I/O)

◆ マイグレーション後はマイグレーション先で表示
  - 携帯の通信の制限
- キャンバス
  - JPEG形式でエンコードして転送
- GUIコンポーネント
  - 追加されたコンポーネントを記録して再構築



11

マイグレーション(データ入出力Port)

- HTTP通信Port
  - 接続は単一、接続中の操作は不可
  - 通信前
    - 送信データを転送してByteArrrayOutputStream作成
  - 通信後
    - 受信済データを転送してByteArrayInputStream作成
- 二次記憶入出力Port(スクラッチパッド/レコードストア)
  - 全部転送してメモリ上のコピーをFMLで操作

12

携帯電話用API

- Java用のAPIと同様のAPIを提供
  - グラフィック・GUIコンポーネント・HTTP通信
  - オブジェクトは引数に渡す形式
- 非同期イベント処理はScheme手続きを登録
  - コールバック関数や継続で記述可能

```
(define cvs (make-Canvas))
(define g (Canvas-getGraphics cvs))
(draw-Line g 0 0 50 50)
…
(define http (Connector-open "http://hoge/"))
(http-connect http)
…
```

13

サイズ/メモリの評価

- Javaで記述された他のScheme処理系と比較
  - JAKLD: ツリートラバース型

JARパッケージサイズ

プロファイル	本処理系	JAKLD
DoJa	55KB	40KB
MIDP	55KB	-
PC	45KB	36KB

実行時メモリ(ガベージコレクト後)

処理	本処理系
起動時	230KB
リスト作成	21KB

- 省サイズ・省メモリを実現

14

処理速度の評価(携帯)

- 携帯実機上では本処理系はJAKLDより遅い
  - 式の評価方法の違いが原因
    - Java VMの性能によるスタックとヒープの速度差
      - JAKLDは再帰で処理 → コンテキスト情報はスタックへ格納
      - 本処理系はループで処理 → コンテキスト情報はヒープへ格納

コード	本処理系	JAKLD
(tak 18 12 6)	189.2[s]	26.89[s]
再帰ループ	746[μs]	-
関数付ループ	1.468[ms]	-
継続ループ	764[μs]	-
do構文ループ	849[μs]	76[μs]

※ループは1ループあたり

15

処理速度の評価(PC)

- PC(Pentium4-3GHz)ではJAKLDと同等
- SISCよりは遅い:ツリートラバースとVM方式の差

コード	本処理系	JALKD	SISC
(fib 30)	**12.2[s]**	17.5[s]	3.52[s]
(tak 18 12 6)	**406[ms]**	491[ms]	80[ms]
再帰ループ	3.74[μs]	-	0.89[μs]
関数付ループ	4.89[μs]	-	1.39[μs]
継続ループ	**4.82[μs]**	-	6.50[μs]
do構文ループ	4.51[μs]	3.81[μs]	0.88[μs]

※SISC: Javaで記述されたVM型Scheme処理系
※JAKLDは継続や末尾最適化に非対応

16

マイグレーション デモ動画

17

まとめ

- ◆携帯Java VM上のScheme処理系を実現
- 携帯電話向けAPIを提供
  - Schemeによる携帯アプリ開発環境を提供
  - APIはキャリア非依存
- 継続や末尾呼び出しの最適化に対応
- 継続などのマイグレーションに対応
  - モバイルエージェントシステム
  - 分散処理や情報処理システム開発の基盤

18

今後の課題

- 携帯電話上のエディタ
  - T9方式＋識別子辞書
    - 526232 → lambda
  - 括弧入力特化
  - S式の整形表示
- 応用システム
  - マイグレーションの活用
    - モバイルエージェント
      - 情報検索、AIプログラム

19

解像度非依存型動画像処理ライブラリ RaVioli の提案と実装

岡田　慎太郎[†]　津　邑　公　暁[†]　松　尾　啓　志[†]

RaVioli: a Resolution-Independent Video Processing Library

Shintaro Okada,[†] Tomoaki Tsumura[†] and Hiroshi Matsuo[†]

1. はじめに

計算機の高性能化による画像処理能力の高度化と，高速なインタフェースの普及によるリアルタイムストリーミングの実現から，汎用 PC および汎用 OS 上でのリアルタイム動画像処理が今後広く一般に行われると予想される．

我々はこの汎用的な環境でのリアルタイム動画像処理を実現するにおいて，動画像のリアルタイム処理を保証できないという問題と，動画像処理プログラミングの在り方に着目する．そこから，1 フレームの処理量に関わらず動画像のリアルタイム処理が可能である一方で，プログラマに対し対象画像の解像度および画素を隠蔽して，抽象度の高い記述を容易とした RaVioli を提案する．

2. 提案手法

2.1 演算量の自動調整

一般に 1/30 もしくは 1/60 秒のキャプチャ間隔でリアルタイムストリーミング処理を行う場合，1 フレームに対する処理もそれに応じた時間で処理する必要がある．しかし 1 フレームの処理量や計算機のリソース量によっては，実時間での処理が不可能な場合がある．そこで処理フレームレートと処理解像度を自動的に低減させることにより動画像処理の処理速度を調整し，リアルタイム性を維持する手法を提案する．また，ユーザが処理調整の対象をフレームレートと解像度で選択可能とする．

† 名古屋工業大学
Nagoya Institute of Technology

2.2 動画像処理の抽象化

そもそも視覚情報を認識する際，人間の脳には，「視覚イメージは画素の集合である」という意識は存在していない．また処理を考える際には変換をするための方法 (以下変換ルール) を思い浮かべるが，それを全画素に対して適用するのは計算機特有のものである．この人間と計算機におけるイメージの扱い方の違いが，動画像プログラミングを複雑にしている原因の一つと考える．よって本研究では，人間の視覚イメージを計算機上でも実現するため，画素や解像度を隠蔽することでプログラミングを簡易化，抽象化するライブラリの作成を目標とする．また，時間軸上においても解像度と同様にフレームレートを隠蔽することで，単位時間あたりの絶対フレーム数を意識する必要のない環境を提供する．

3. 実　　装

例えばグレースケール化のように画像の構成画素全てに対して同じ処理を適用する場合を考える．従来の画像処理プログラミングでは，まず入力画像の構成画素数に応じたループを構成し，そのループ内で各座標に対応する画素に対してユーザが変換ルールを記述した関数を適用することでこれを実現する方法が一般的である．

これに対して本ライブラリでは，入力画像をカプセル化した画像インスタンスに対し，そのインスタンスが持つ「構成画素全てに処理を適用する」高階メソッドに，各画素に対して行うべき変換ルールを記述した処理を定義した関数を渡すことで，この処理を可能とする．この場合，プログラマは入力画像の構成画素数を用いた記述をする必要がなく，解像度が変更された場合でもその差はライブラリにより吸収され，自動的

に必要な画素にのみ処理が適用される．

またフレームも解像度と同様に扱い，カメラからキャプチャした動画はインスタンスを形成してフレームを隠蔽する．またそのインスタンスは，1 枚のフレームもしくは 2 枚のフレームを引数に取る高階メソッドを持つ．

本ライブラリでは，この機能を実現するにあたり，PIXEL,IMG および STREAM の 3 つのクラスといくつかの高階メソッドを持つライブラリを C++で実装した．PIXEL クラスは画像を構成する 1 画素の情報を保持するクラスであり，RGB 値を操作するためのメソッドを持つ．IMG クラスは画像の実体を表すクラスであり，メンバ変数として PIXEL インスタンスを構成画素数ぶん配列の形で保持する．また，一般に構成画素全体に対しプログラマが変換ルールを記述した関数を適用する高階メソッドを実装している．STREAM クラスは動画像を処理するクラスであり，代表的なものにカメラから動画像をキャプチャするメソッドと，数枚のフレームを配列で格納しているリングバッファおよびそのメソッドがある．

4. 評　　価

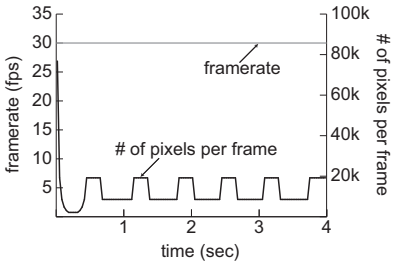
4.1 実 験 環 境

動画像処理の評価として，本ライブラリを用いてフレーム間差分を作成し，解像度と処理フレームレートの調整による疑似リアルタイム処理の評価を行った．環境として，CPU は AMD Opteron Dual-Core 2GHz，メモリは 2GB の計算機上で実験を行った．ビデオカメラは SONY の DCR-TRV900，ビデオキャプチャーボードは I・O DATA の GV-VCP2M を用い，S1 端子から NTSC 形式で映像信号を取り込んだ．

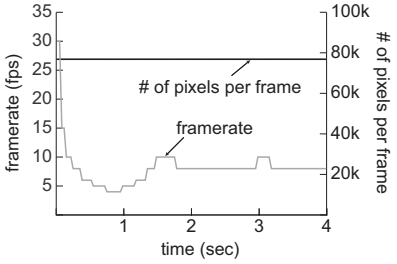
4.2 出 力 結 果

カメラから 30fps で転送される解像度 320×240 のフレームをキャプチャし，前項で示したフレーム間差分プログラムを出力フレームレート優先と解像度優先でそれぞれ適用させることで，評価を行った．

図 1 の（1）および（2）は，出力フレームレートおよび解像度のそれぞれを優先した場合の，処理開始から 4 秒後までの時間変化を表したグラフである．(1) では出力フレームレートは常に最大値が維持されている一方で，解像度は処理開始直後より低下して行き，0.5 秒程度でほぼ一定の値に収束している様子が読みとれる．なお一度収束値よりも低い値まで落ち込んでいるのは，処理開始直後の過負荷により遅れてしまったフレーム処理をキャプチャフレームに追いつかせるために，一時的に定常状態よりも低い負荷にまで抑える必要があったためである．(2) においても同様に，解像度は最大値を維持しつつ，出力フレームレートが適切な値にまで自動的に低減し，処理開始から 1.5 秒程度で定常状態に収束していることが分かる．



（1）Framerate is preferred.



（2）Resolution is preferred.

図 1　出力フレームレート優先時及び解像度優先時の処理結果

以上の結果から，本ライブラリがプログラマの指定する優先度に基づき正しく自動負荷調整を行えること，および解像度やフレームレートが変動した場合でもプログラムが正しく動作することが確認できた．

5. お わ り に

本稿では，リアルタイム動画像処理の保証を解像度またはフレームレートを変動させることにより実現する解像度非依存型動画像処理ライブラリについて述べた．また本ライブラリは，人間と計算機の画像の扱いの違いを吸収し，人間が画像に対して持つ処理イメージに近い形で記述できるプログラミング環境を提供するにあたり，解像度やフレームレートを隠蔽した動画像に対する高階メソッドを実装することでこれを実現した．そして，フレーム間差分の検出プログラムを用いてその評価を行い，解像度優先とフレームレート優先のそれぞれの場合において，自動的に演算負荷の軽減が行われること，やはりプログラムの変更なしにリアルタイム性を保証できることが確認できた．

今後は，現在実装済みの高階メソッドの仕様の再考と新たな高階メソッドの実装や，省電力プロセシングに寄与する拡張も行っていきたい．

解像度非依存型画像処理ライブラリ RaVioliの提案と実装

○岡田慎太郎（名工大）
津邑　公暁（名工大）
松尾　啓志（名工大）

研究の背景

- 汎用計算機の高性能化
  - 実時間での高度な画像処理が可能

- 高速なインターフェースの普及
  - カメラ等からリアルタイムで動画像を取り込める環境

⇒汎用計算機上でのリアルタイム動画像処理

着目点と提案

A) **リアルタイム性を維持できない**
- キャプチャフレーム間隔 ＜ 1フレーム分の処理時間
- 出力フレームレートか
  処理解像度を削減して処理量を自動調整

⇒ **リアルタイム性を疑似的に維持**

B) **動画像プログラミングの在り方**

着目点A:リアルタイム処理の維持

A) **リアルタイム性を維持できない**
- キャプチャフレーム間隔 ＜ 1フレーム分の処理時間

リアルタイム性が維持できている場合

着目点A:リアルタイム処理の維持

A) リアルタイム性を維持できない

- キャプチャフレーム間隔 ＜ 1フレーム分の処理時間

リアルタイム性が維持できていない場合

フレーム間隔内に終わる保障は無い

↓

処理量の削減により処理時間を短縮

Aの提案:動画像処理の処理量調整

- 2種類の処理時間調整方法
  - 処理フレームレートの変更による調整

15 fps

  - 処理解像度の変更による調整

30 fps

Aの提案：動画像処理の処理量調整

- 2種類の処理時間調整方法
  - 処理フレームレートの変更による調整

常に最高の解像度を維持

ユーザがどちらかを優先する

  - 処理解像度の変更による調整

常に最高のフレームレートを維持

着目点と提案

A) リアルタイム性を維持できない

- キャプチャフレーム間隔 ＜ 1フレーム分の処理時間
- 出力フレームレートか
  処理解像度を削減して処理量を自動調整

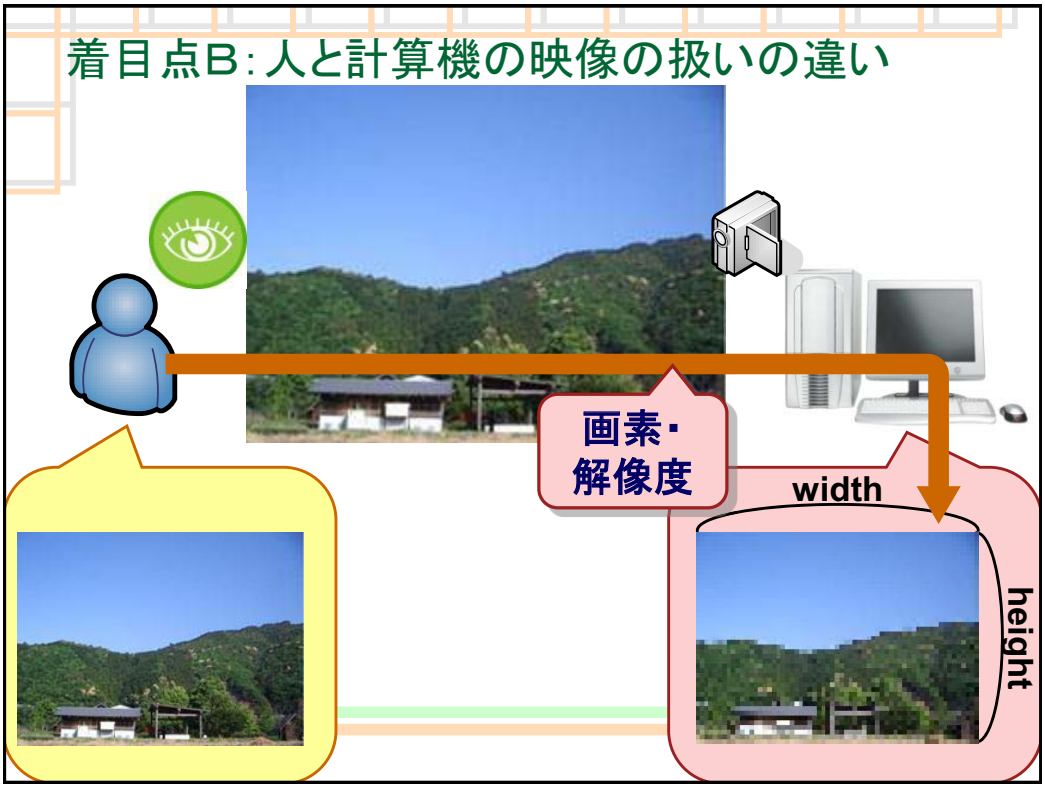
⇒ **リアルタイム性を疑似的に維持**

B) 動画像プログラミングの在り方

- 解像度・フレームレートを考慮したプログラミング
- 人間は視覚イメージを量子化をしない
- 解像度やフレームレートを隠蔽

⇒ **ユーザプログラムの簡易化**

着目点B：人と計算機の映像の扱いの違い



ユーザインターフェース：
提案の概要

ユーザインターフェース：実装方針
解像度の隠蔽

画像のカプセル化

width

height

procPix(**func**);

procPix

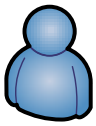
width / 2

height / 2

```
for(j = 0; j < height; j++){
    for(i = 0; i < width; i++){
        func(pixel[i][j] );
    }
}
```

従来の書式

ユーザインターフェース：実装方針
フレームレートの隠蔽



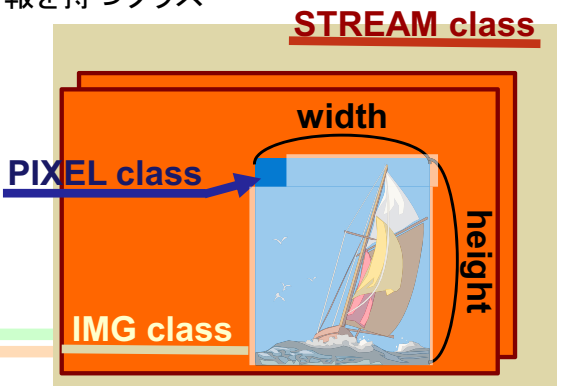
```
while(1){
// target : 現在のフレーム
// before: 1つ前のフレーム
    capture(target);
    frameDiff(target,before);
    before=target;
}
```

従来の書式

ユーザインターフェース：実装方針
フレームレートの隠蔽

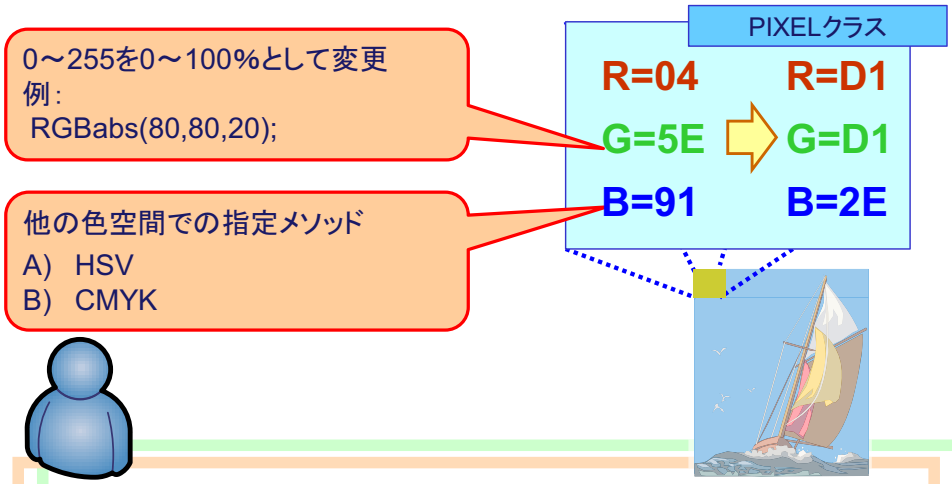
実装

- C＋＋言語による動画像処理ライブラリを実装
  - PIXELクラス
    - 1ピクセルの情報を持つクラス
  - IMGクラス
    - 1枚あたりの画像の情報を持つクラス
  - STREAMクラス
    - 複数枚の画像の情報を持つクラス



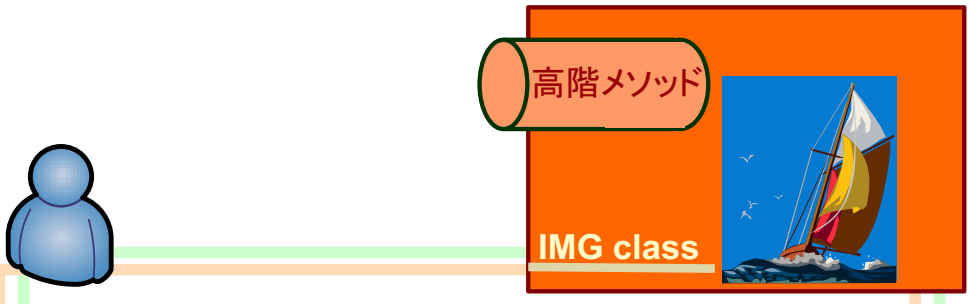
PIXELクラス

- 1ピクセルあたりのRGBの値を保持
- RGB値を変更するさまざまなメソッドを実装



IMGクラス

- 画素数分のPIXELインスタンスを持つ
- 複数の高階メソッドを実装
  - 1ピクセル分の処理関数を画像全体に施す
  - ユーザ定義の処理関数を引数にとる

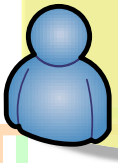


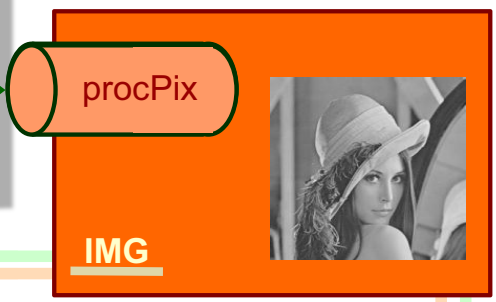
IMGクラスを使ったプログラミング例
グレースケール化プログラム

例：グレースケール化プログラム

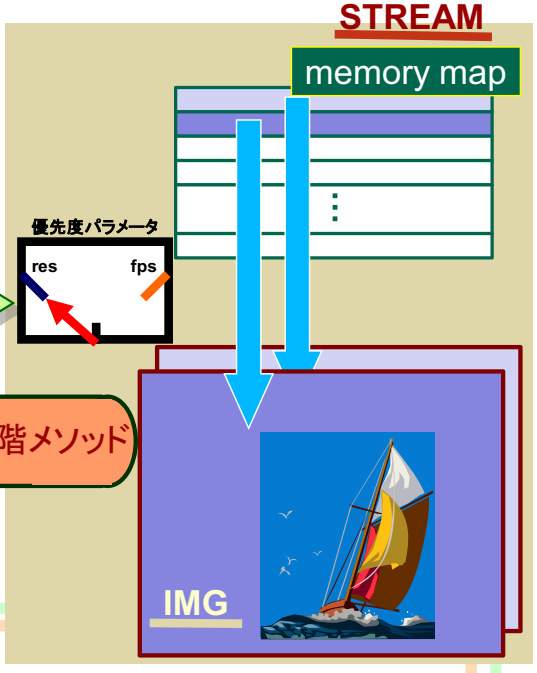
```
toGray(PIXEL* target){
    target-> HSVrel(-1,-1,100);
};

void main(argv[2]){
        :
    IMG Lenna;
    Lenna.procPix(toGray);
        :
```





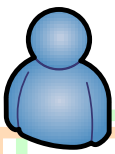
STREAMクラス



解像度 or フレームレート

フレーム間差分プログラム

高階メソッド



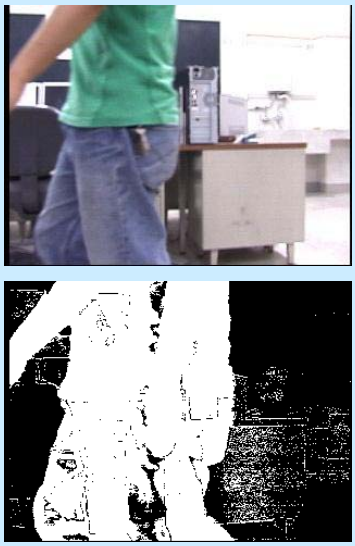
本ライブラリの評価

- 評価
  - サンプルプログラムとしてフレーム間差分を作成
- 検証
  - フレームレート優先，解像度優先の場合において，処理開始後4秒間の変化を観察
  - リアルタイム処理が正確に行われていることを検証

動画像の検証

処理対象フレーム

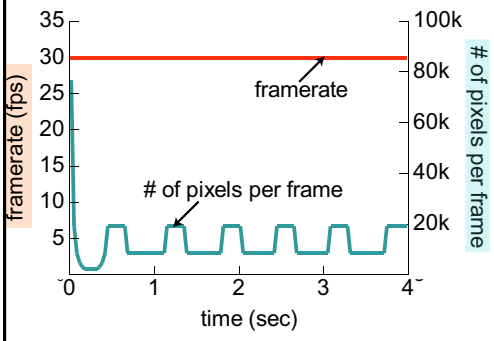
出力フレーム



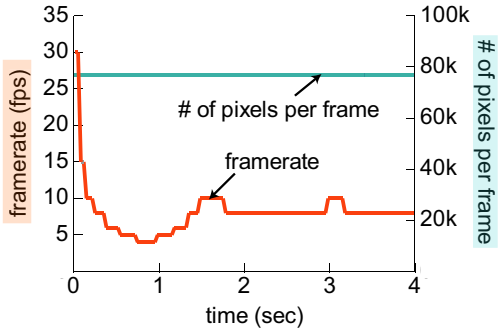
フレームレート優先

解像度優先

ピクセル数とフレームレートの時間変化



フレームレート優先



解像度優先

今後の課題

- 新たな高階メソッドの追加
- 処理の高速化
- 動画像処理の省電力化

並列化および再利用によるGAの高速化

新美明仁[†] 池内康樹[††,☆] 鈴木郁真[††,☆☆]
津邑公暁[†] 松尾啓志[†] 中島康彦[†††]

Speed-up of GA by Parallelization and Auto-Memoization

Akihito Niimi,[†] Yasuki Ikeuchi,[††,☆] Ikuma Suzuki,[††,☆☆]
Tomoaki Tsumura,[†] Hiroshi Matsuo[†]
and Yasuhiko Nakashima[†††]

1. はじめに

我々は，再利用および並列事前実行を用いた高速化手法を提案している．しかし，並列事前実行では投機に多数のコアを割り当てても更なる高速化は見込めず，逆に速度の低下を招くこともある．また，GA のように並列事前実行の効果が得られないプログラムも存在することから，全てのコアを投機に割り当てる方法は有効とは言えない．本稿では，従来投機に割り当てていたコアをメインコアとして用いた場合の並列処理について考え，2 つのメインコアが再利用表を共有する機構を提案し，GA においてその有効性を検証した．

2. 再利用

2.1 再利用

再利用は，既に実行した関数などの入出力を記憶しておき，次に同じ入出力の関数を実行する場合，既に実行済みの出力を用いることにより，関数の実行自体を省略し高速化を図る手法である．我々の実行モデルでは，再利用対象とする命令区間として，多くの命令を含みかつ始点と終点を用意に特定できる，関数とループを仮定している．具体的には，関数に含まれる命令では，call/jump 命令の分岐先から return 命令を再利用対象として検出する．ループの場合には，1 度後方分岐命令が検出され，その分岐が成立した後，再び同じ後方分岐命令が検出されたときその区間を再利用対象とする．

2.2 並列事前実行

我々が提案する再利用は，既存のプログラムを変更せずに高速化を図ることのできる手法であるが，どんなプログラムでも効果があるわけではない．関数の引数が単調に変化する場合などが該当する．この様な場合では並列事前実行を用いることができる．並列事前実行は，再利用表に登録されている命令区間に対してこれから出現するであろう入力を予測し，メインコアと並行して投機コアにおいて，その予測された入力を用いて投機実行し，結果を再利用表に登録する手法である．そこで，投機コアに割り当てるコアをメインコアに割り当て並列処理を行うことで高速化を図る．本稿では GA を対象とした並列処理を考える．

3. 提案手法

従来の並列事前実行機構ではメインコアが単一で残りを投機コアとしていたが，本稿ではメインコアを複数とする．よって，複数のコアが再利用表を共有することによりそれぞれ再利用が可能となるばかりでなく，それぞれのコアが再利用表に登録したエントリを互いに利用することが期待される．今回の実装では GA への適用を考え，2 つのメインコアで 1 つの再利用表を共有する機構の実装を行った．

† 名古屋工業大学
Nagoya Institute of Technology
†† 豊橋技術科学大学
Toyohashi University of Technology
††† 奈良先端科学技術大学院大学
Nara Institute of Science and Technology
☆ 現在，(株) ACCESS
Presently with ACCESS Co.,Ltd.
☆☆ 現在，トヨタ自動車 (株)
Presently with Toyota Motor Corp.

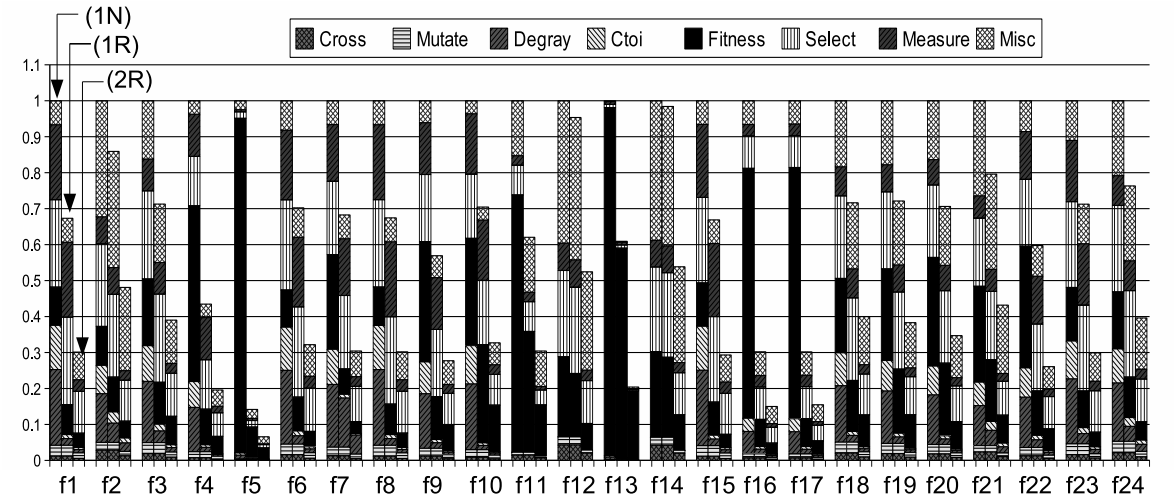


図 1 評価結果

4. 評　　価

評価には，メインコアを 2 つにした並列事前実行プロセッサシミュレータを用いた．キャッシュや命令レイテンシは，SPARC64-III6) を参考にしている．また，評価プログラムには汎用 GA ソフトウェアである GENEsYs を用いた．今回は適合度関数計算を再利用対象とし，以下の場合について評価を行った．

- (1N) コア数 1，再利用なし
- (1R) コア数 1，再利用あり
- (2R) コア数 2，再利用あり

実験の結果を**図 1** に示す．図 1 の横軸は GENEsYs の持つ 24 種類の適合度関数を示している．縦軸は，サイクル数の比を表しており，左から順に (1N)，(1R)，(2R) をそれぞれ示し，(1N) のときを 1 として正規化している．なお，(2R) のグラフは 2 つのコアの内，サイクル数の多い方を示した．凡例は，GENEsYs の持つ処理時間の内訳をサイクル数の比で示している．GENEsYs の主な処理は，交叉 (Cross)，突然変異 (Mutate)，グレイコードの逆変換 (Degray)，char 型配列から整数への変換 (Ctoi)，適合度計算 (Fitness)，個体選択 (Select) に分けられる．Mutate，Degray は遺伝子表現の変換に用いられる処理である．また GENEsYs は，各世代ごとに個体の適合度平均などの計算を行っている．この処理を Measure とし，上記以外の初期化などの処理を Misc とした．図 1 より (2R) は (1N) に対して最大 93 %のサイクル数削減，14.2 倍の高速化を実現したことになる．さらに平均では 68 %のサイクル数削減，3.1 倍の高速化となった．まず Fitness を見ると，すべての関数で再利用の効果が得られサイクル数が削減されていることが分かる．特に f4，f5，f11，f13，f16，f17 は，総サイクル数に対して Fitness の割合の高い関数であり，提案手法である (2R) は (1R) の半分以下にまでサイクル数の削減を実現しているこれは，一方のコアが再利用表へ登録したエントリを他方のコアが再利用することができているためである．

5. ま と め

本稿では GA において，並列化と再利用に着目し，2 つのコアを用いて再利用表を共有する手法を提案した．GENEsYs での評価においてコア数 2 で再利用ありではコア数 1 で再利用無しと比較して最大 14.2 倍，平均 3.1 倍の高速化を実現した．本来並列化しただけの手法では最大 2 倍の高速化しか実現できないが，提案手法ではほぼ全ての関数において 2 倍以上の高速化を実現したことから，再利用表を 2 つのコアで共有することで，更なる再利用効果が得られることを確認した．提案手法は並列事前実行の効果が得られるプログラムでは投機コアを実行に多く割り当て，効果の得られないプログラムでは投機コアの数を減らし，その分をメインコアに割り当てるように投機コアとメインコアの数を実行時に動的に変えることを想定している．よって今後はプログラムに応じて，投機コアとメインコアの数の割り当てを動的に変更する手法について検討する予定である．

また，今回は GA を対象として評価したが，GA 以外のプログラムに対する本提案手法の有効性も検証して行きたい．

並列化および再利用によるGAの高速化

†新美明仁　‡池内康樹　‡鈴木郁真
†津邑公暁　†松尾啓志　††中島康彦

†名古屋工業大学
‡豊橋技術科学大学
††奈良先端科学技術大学院大学

研究の背景

- これまでに再利用と並列事前実行によりプログラムの高速化を実現してきた

平均サイクル数削減
Stanfordベンチマーク ⇒ 42%
SPEC CPU95 ⇒ 30%

研究の背景

- これまでに再利用と並列事前実行によりプログラムの高速化を実現してきた
  - 複数のコアを投機実行に用いる高速化手法
- 近年CPUの搭載コア数が増加している
  - 多数のコアを投機実行に使用しても性能に寄与しない

多くのコアを有効活用する高速化手法が必要

発表の流れ

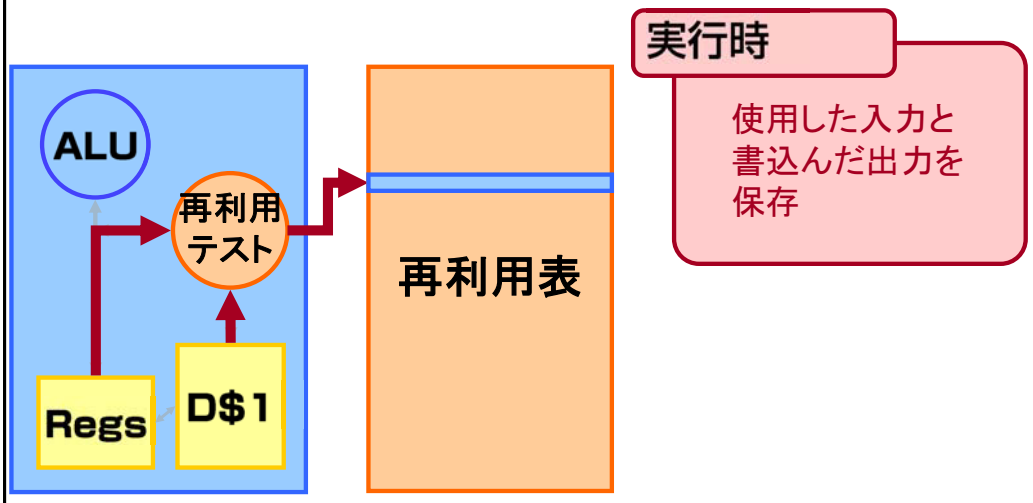
① 再利用および並列事前実行

② 並列事前実行の問題点
- 投機コア数と性能の関係

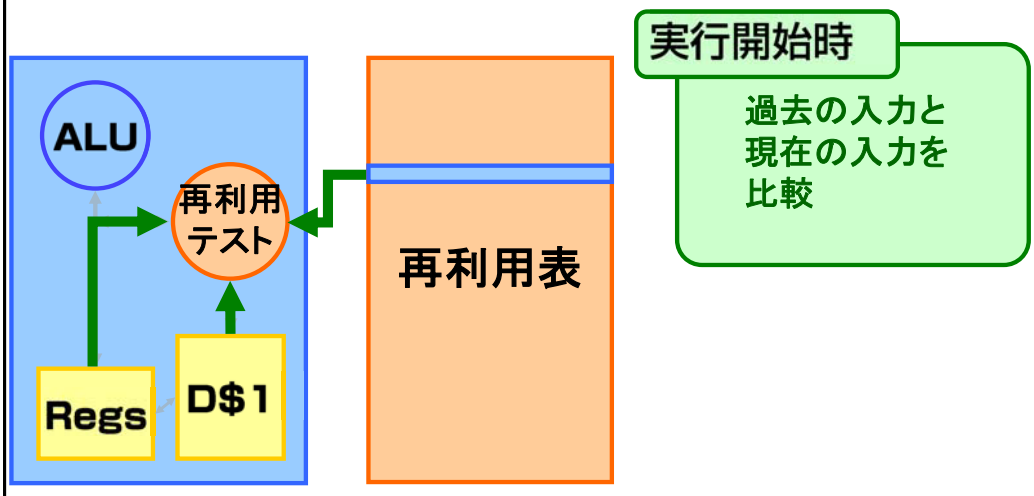
③ 提案手法
- 並列処理と再利用の融合

④ 評価

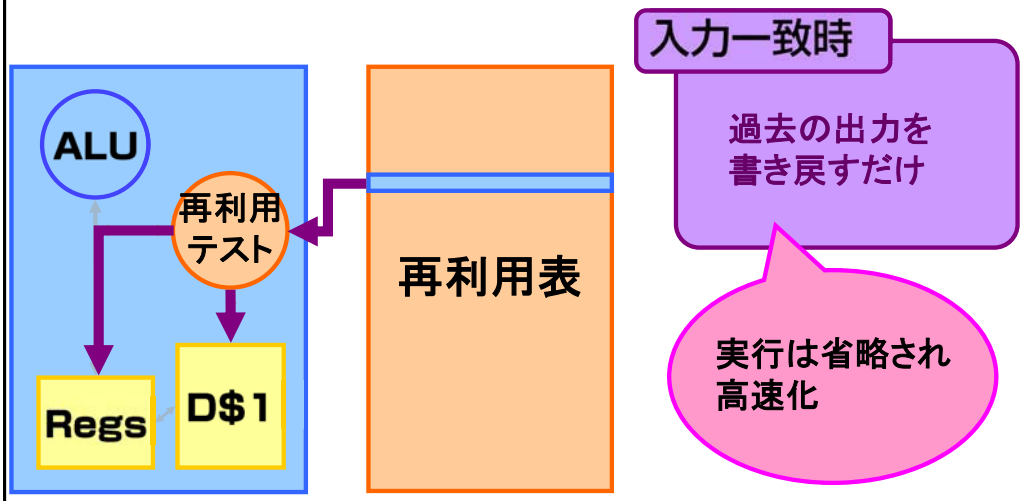
再利用機構


ALU
再利用
テスト
Regs
D$1
再利用表
実行時
使用した入力と
書込んだ出力を
保存

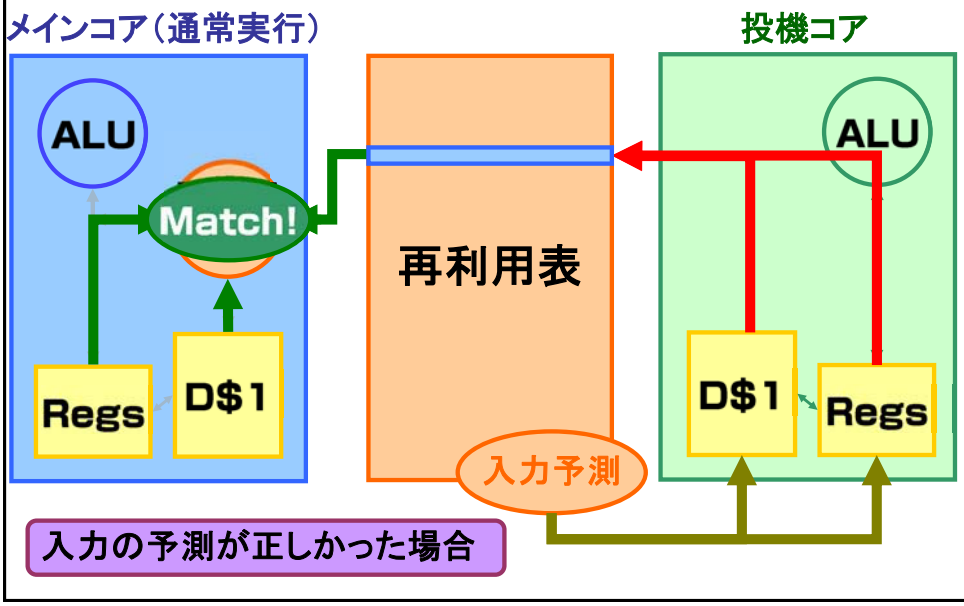
再利用機構


ALU
再利用
テスト
Regs
D$1
再利用表
実行開始時
過去の入力と
現在の入力を
比較

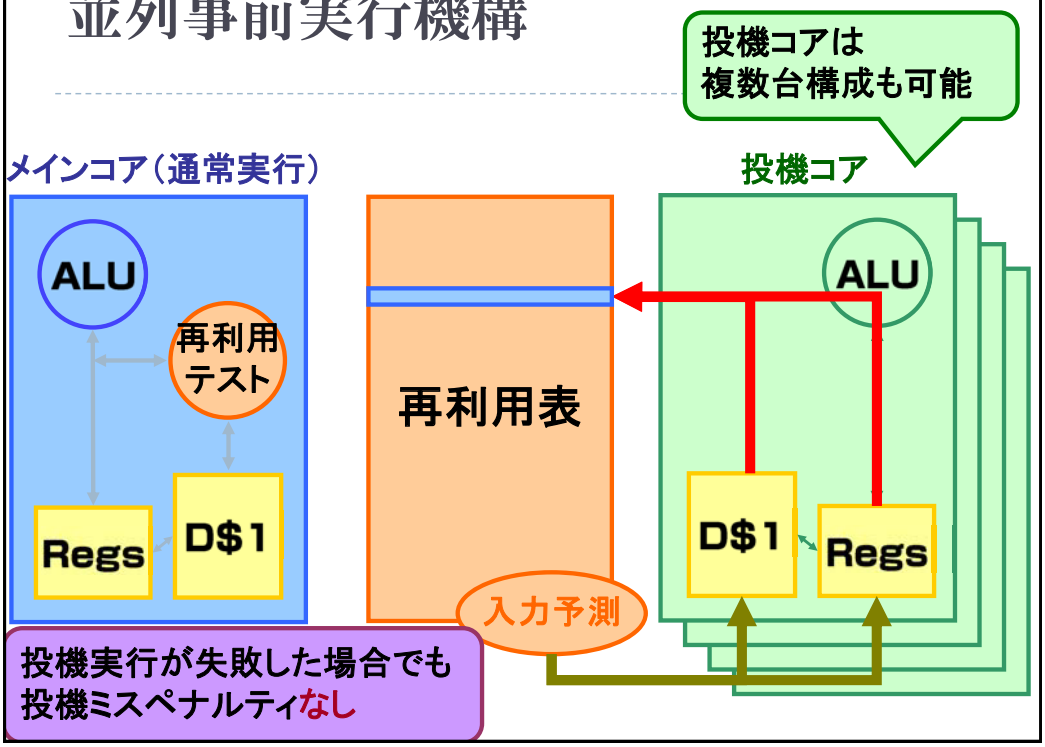
再利用機構



並列事前実行機構



並列事前実行機構



並列事前実行が効果的なプログラム

- ストライド予測
  - 実装 ⇒ 単純な構造で実装コストが低い
    - 入力が単調変化する場合に大きな効果が得られる

例：

```
for(i=0; i<5; i++){
    j = i * i;
}
```

```
for(i=0; i<5; i++){
    for(j=0; j<5; j++){
        func(j,i);
    }
}
```

  - 予測 ⇒ 複雑な予測はできない

近年のCPUの搭載コア数の増加

- CPUに搭載されるコアの数が増加している
  - 並列事前実行は投機コアを複数構成にできる
- 予測方法と投機コア数との関係
  - 複雑な予測はできない

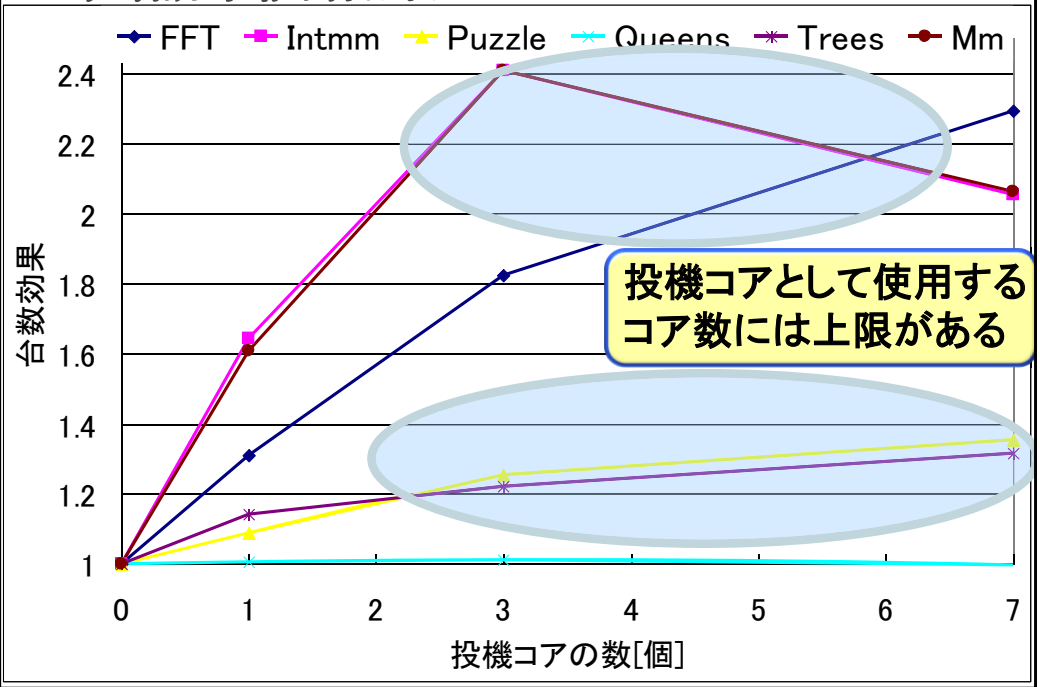
投機に多くのコアを費やしたした場合
コア数に応じた性能向上が得られないのでは？

予備実験

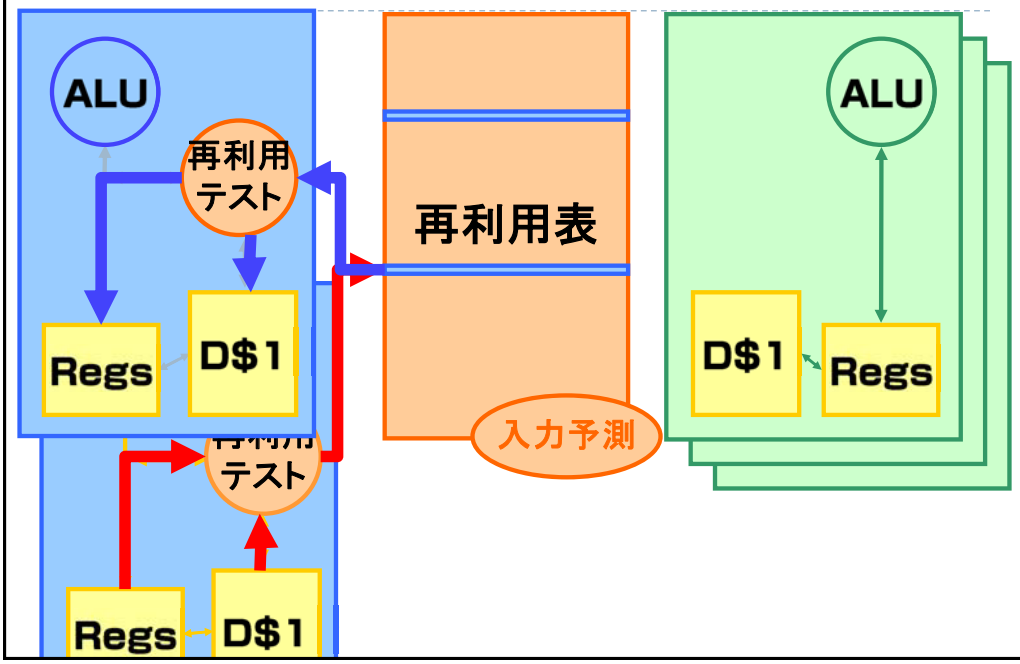
- 目的
  - 投機コアを増やしたら性能はどうなるかを確認する
- ベンチマーク
  - Stanfordベンチーマークを用いた

メインコア数：1
投機コア数　：1，3，7と変化させた

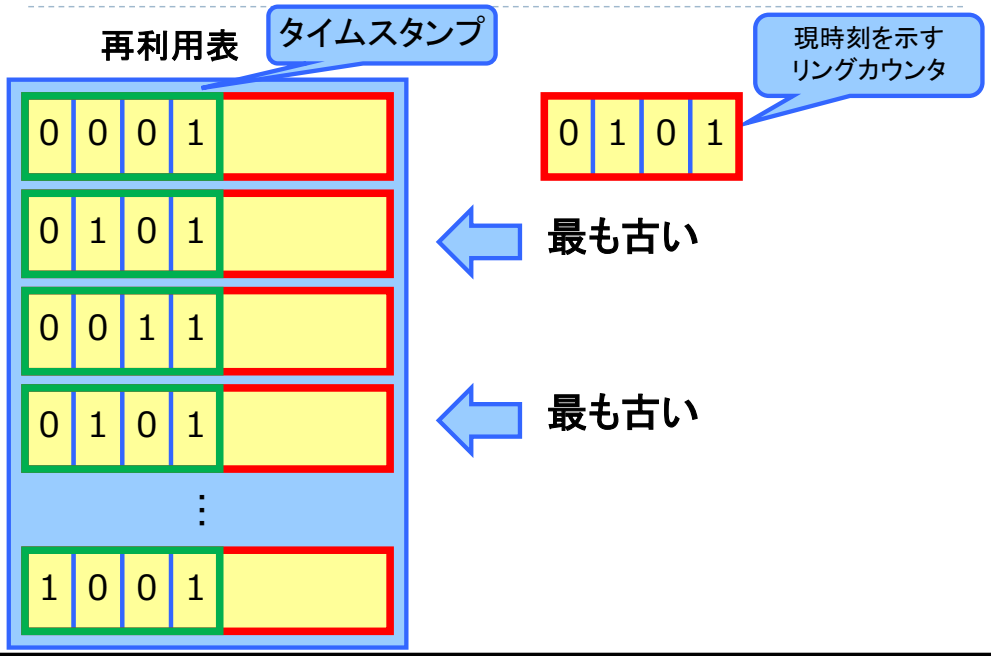
予備評価結果



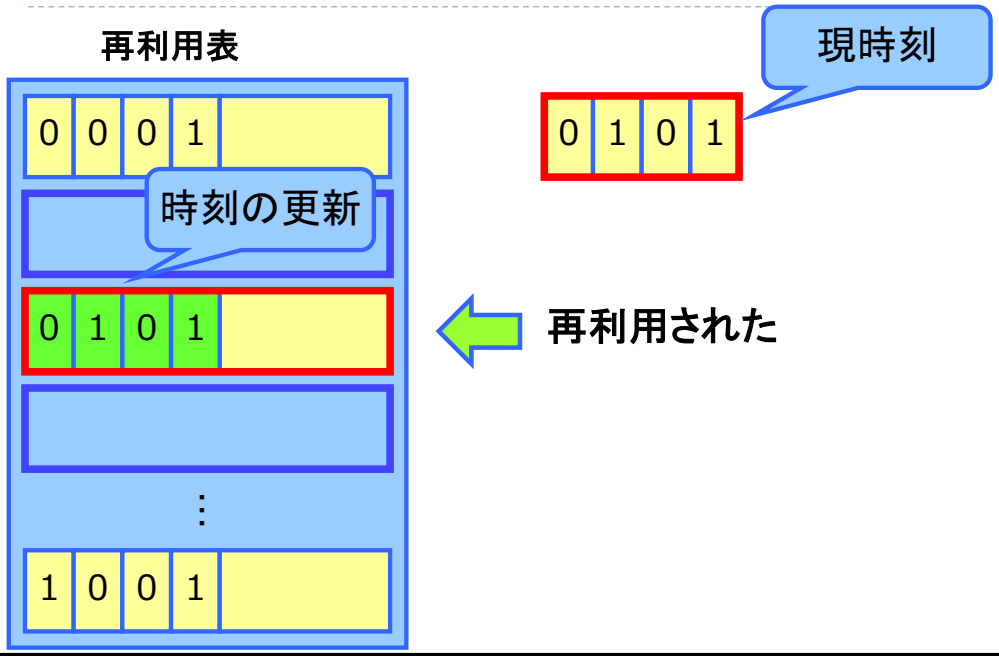
本研究の提案手法



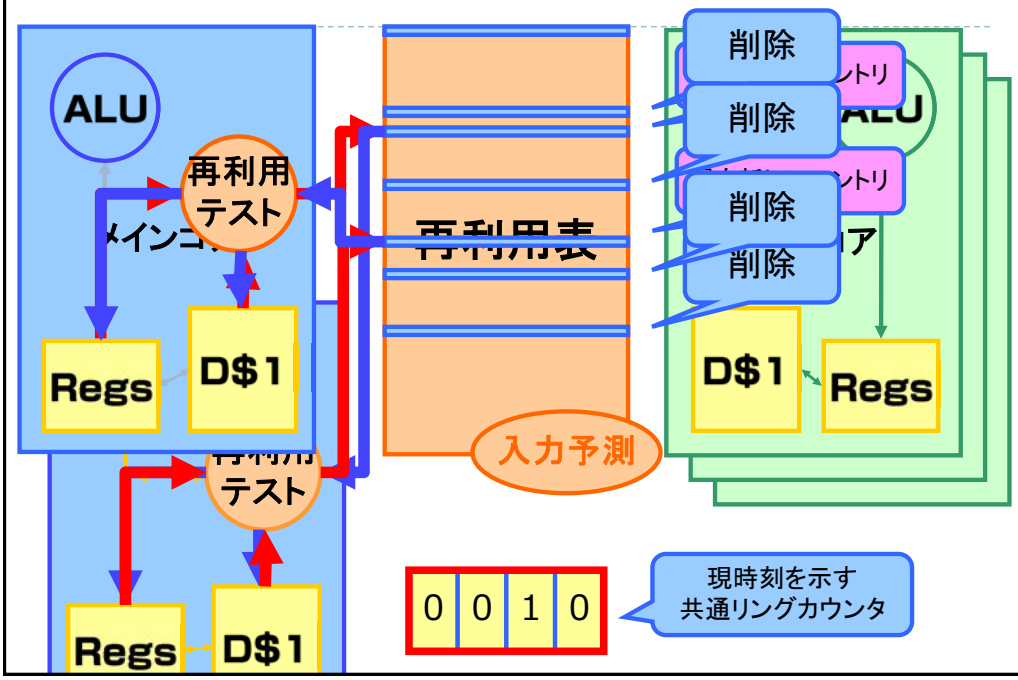
パージ機構：エントリの削除



パージ機構：TSIDの更新



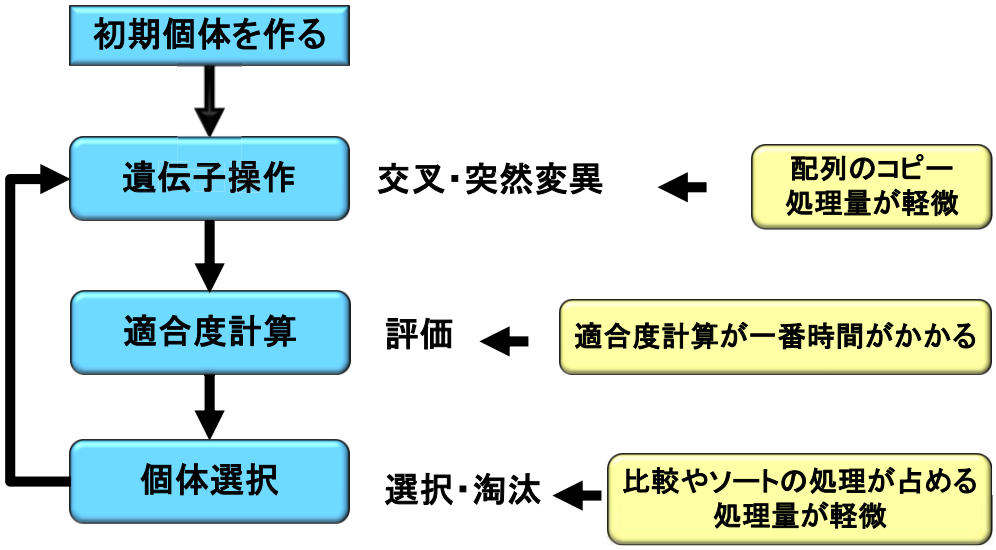
提案手法におけるパージの実装



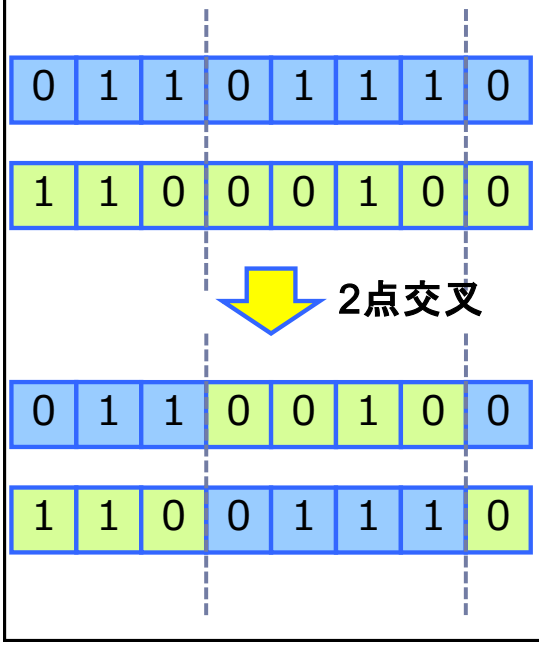
評価プログラム：GA

- GAは再利用が非常に効果がある
- 並列化が容易である
  - メインコアが再利用表を共有し並列処理する機構で得られる効果を検証することができる
- 並列事前実行の効果が得にくい

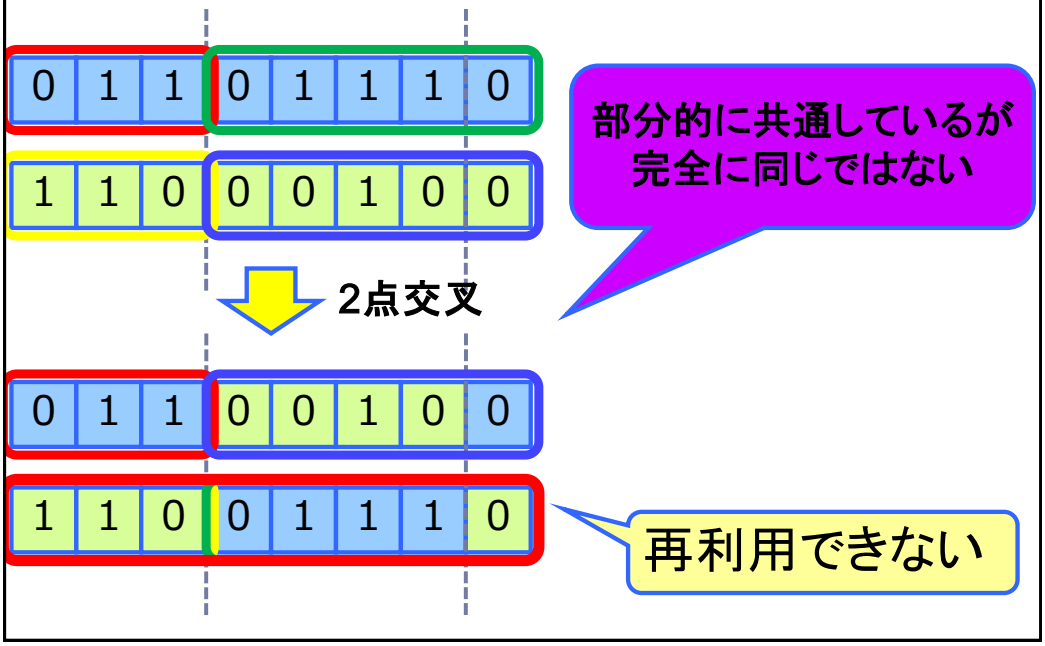
GA（遺伝的アルゴリズム）



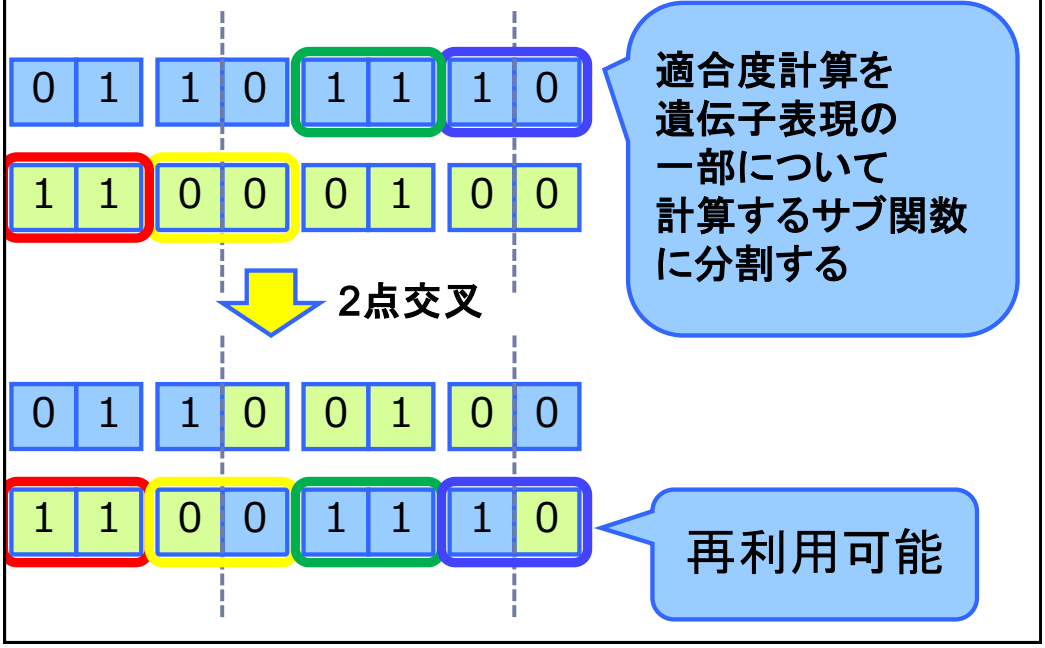
GAへの再利用の適用



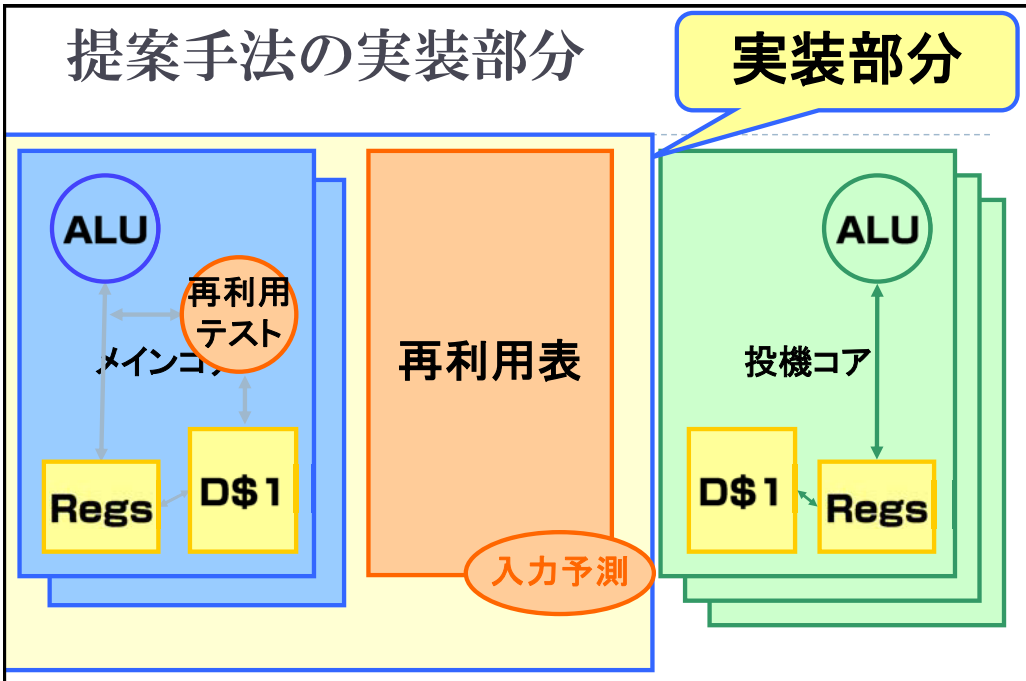
GAへの再利用の適用



GAへの再利用の適用



提案手法の実装部分



評価環境

- 評価には汎用GAソフトウェアGENEsYsを使用する

測定環境

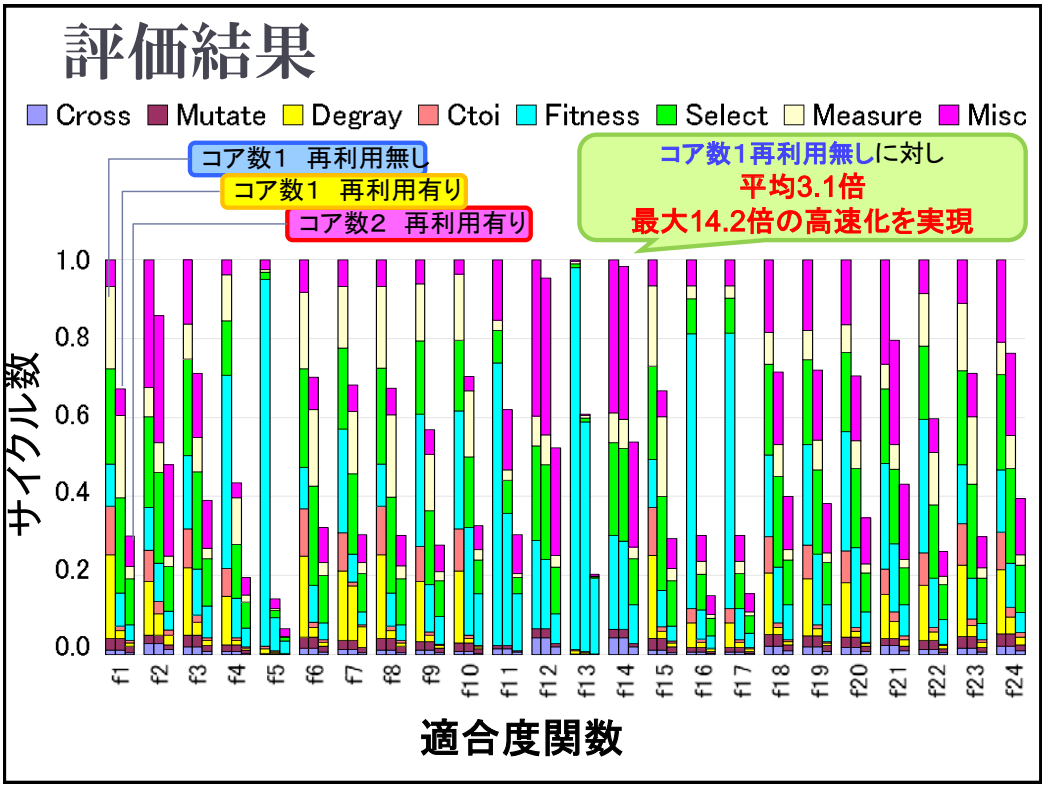
◆コア数1:再利用無し
◆コア数1:再利用有り

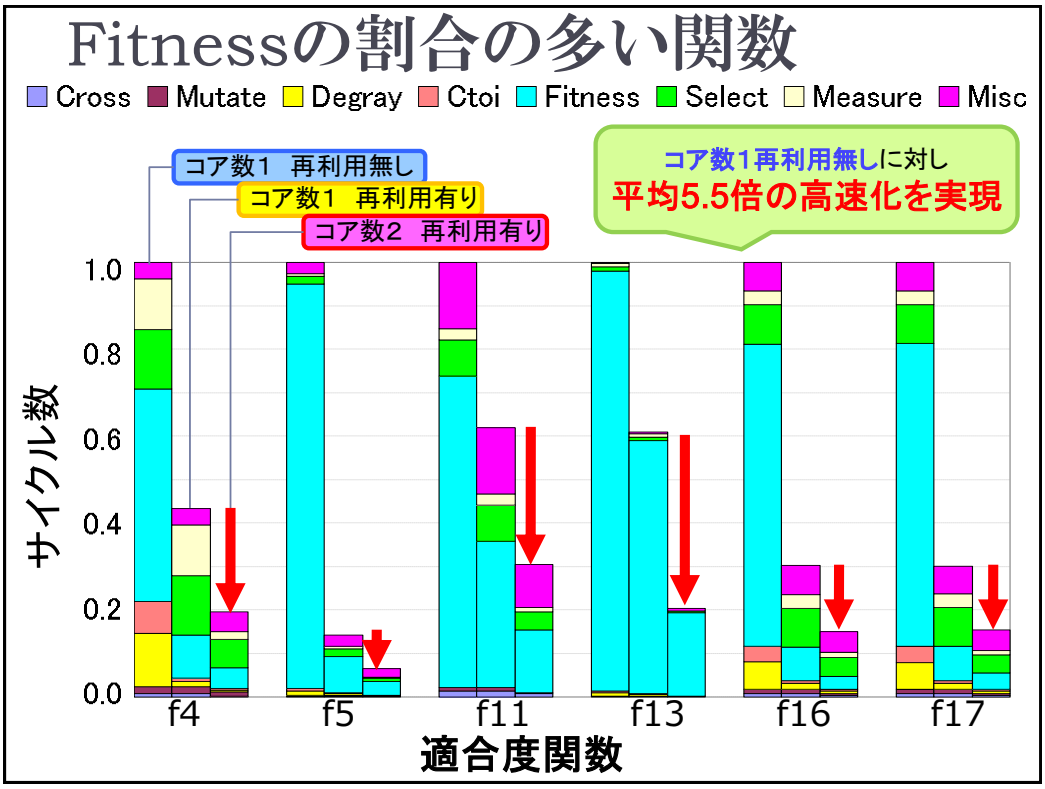
◆コア数2:再利用有り

提案手法

パラメータ

個体数	50 個
交叉率	60 %
交叉点数	2 点
突然変異率	0.1 %
世代数	25 世代


評価結果
Cross
Mutate
Degray
Ctoi
Fitness
Select
Measure
Misc
コア数1　再利用無し
コア数1　再利用有り
コア数2　再利用有り
コア数1再利用無しに対し
平均3.1倍
最大14.2倍の高速化を実現
サイクル数
1.0
0.8
0.6
0.4
0.2
0.0
f1
f2
f3
f4
f5
f6
f7
f8
f9
f10
f11
f12
f13
f14
f15
f16
f17
f18
f19
f20
f21
f22
f23
f24
適合度関数


Fitnessの割合の多い関数
Cross
Mutate
Degray
Ctoi
Fitness
Select
Measure
Misc
コア数1　再利用無し
コア数1　再利用有り
コア数2　再利用有り
コア数1再利用無しに対し
平均5.5倍の高速化を実現
サイクル数
1.0
0.8
0.6
0.4
0.2
0.0
f4
f5
f11
f13
f16
f17
適合度関数

今後の課題

- プログラムの振る舞いに合わせた投機コアとメインコアの動動的割り当てを考える

まとめ

- 並列化と再利用に着目し、複数のメインコアを用いて再利用表を共有する手法を提案
- 2つのメインコアが再利用表を共有する形で実装
- GENEsYsにおいてコア数2つで再利用ありはコア数1つで再利用なしと比較して最大14.2倍の高速化を実現

自動メモ化プロセッサの消費エネルギー評価

島崎裕介[†1] 池内康樹[†2] 津邑公暁[†1]
中島浩[†3] 松尾啓志[†1] 中島康彦[†4]

Energy Consumption of Auto-Memoization Processor

Yusuke Shimazaki,[†1] Yasuki Ikeuchi,[†2] Tomoaki Tsumura,[†1]
Hiroshi Nakashima,[†3] Hiroshi Matsuo[†1]
and Yasuhiko Nakashima[†4]

表 1 SPEC CPU95 による結果

	従来手法	提案手法
中断プログラム数（11 個中）	0	8
平均削減サイクル数	6.8 %	4.9 %
平均消費エネルギー比	+14 %	+1.5 %
（内，メモ化中断プログラム）	+19 %	+0.90 %

表 2 GENEsYs による結果

	従来手法	提案手法
中断プログラム数（24 関数中）	0	15
平均削減サイクル数	17 %	14 %
平均消費エネルギー比	+1.8 %	— 4.7 %
（内，メモ化中断プログラム）	+12 %	+1.0 %

1. 導　　入

我々は，計算再利用技術に基づく自動メモ化プロセッサを提案している．本稿では，自動メモ化プロセッサに Wattch[1] を参考とした電力評価モジュールを実装し，消費エネルギー評価を行った．メモ化により高速化を図ることは可能だが，プログラムによってはメモ化の効果が現れず，メモ化機構追加分のエネルギーを余分に消費する場合がある．そこで計算再利用率に応じてメモ化機構への電力供給を遮断する機能を実装した．

2. 評　　価

2.1 SPEC CPU 95

初めに，一般的なベンチマークである SPEC CPU95 (train) を用いてプロセッサの消費エネルギー評価を行った．**表 1** にシミュレーション結果を示す．

メモ化機構の追加により，消費電力はおよそ 25 %増加する．その為，実行時間を 8 割以下へと削減できたいくつかのプログラムにおいては，メモ化機構を持たない普通のプロセッサに対し低消費エネルギーを実現しているが，メモ化の効果が現れないプログラムもまた存在する．そこで，上述したプログラムに対し，その計算再利用率からメモ化機構の停止を決定する機能を提案し実装したところ，従来の自動メモ化プロセッサに比べ大きく消費エネルギーを抑制することができた．中断のおきた 8 つのプログラムにおける平均消費エネルギーからも，メモ化効果の現れにくいプログラムに対し効果的にメモ化中断が行われたことがいえる．

2.2 GENEsYs

次にメモ化の効果が得られやすい，遺伝的アルゴリズムのベンチマークである GENEsYs を用いて評価を行った．今回は，メモ化効果の高い適合度関数を更にメモ化効果が得られやすいように書き換えたもの[2] を用いている．**表 2** にその結果を示す．

削減サイクル数は少し下がったが，提案手法により全体の消費エネルギー削減を実現することができた．

†1 名古屋工業大学
Nagoya Institute of Technology
†2 （株）ACCESS
ACCESS Co.,Ltd.
†3 京都大学
Kyoto University
†4 奈良先端科学技術大学院大学
Nara Institute of Science and Technology

参 考 文 献

1) Brooks, D., Tiwari, V. and Martonosi, M.: Wattch: A Framework for Architectural-Level Power Analysis and Optimizations, *Proc. of the 27th Annual Intl. Symp. on Computer Architecture*, pp.83–94 (2000).
2) 鈴木郁真，池内康樹，津邑公暁，中島康彦，中島浩：再利用による GA の高速化手法，情報処理学会論文誌：コンピューティングシステム， Vol.46, No.SIG 16(ACS 12), pp.129–143 (2005).

JSASS 2007

自動メモ化プロセッサの消費エネルギー評価

○ 島崎 裕介 （名工大）
池内 康樹 （ACCESS）
津邑 公暁 （名工大）
中島 浩 （京大）
松尾 啓志 （名工大）
中島 康彦 （奈良先端大）

2007年 9月13日

研究の背景 -メモ化-

更に削減

消費エネルギーの視点から検討

- 命令レベル並列性 による高速化手法
- 並列計算 による高速化手法
- メモ化（計算再利用）

  - 過去に**計算**した関数の出力値を**再利用**
  - 入出力を記憶するためのユニットは CAM 及び RAM を用いて構成 ⇒ 消費電力の増加
  - プロセッサ全体の消費エネルギーを大きく増大させてしまう可能性がある
  - 高速化が起きない場合にはエネルギーの浪費

流れ

自動メモ化プロセッサ

↓ 電力測定機能

エネルギー評価1

↓ 消費エネルギーの削減機能

エネルギー評価2

- 計算再利用による高速化
- 高速化が起こらない場合 消費エネルギーが増大
- 提案:「メモ化中断」による 消費エネルギーの抑制
- 「メモ化中断」による 消費エネルギーの評価

高速化手法 –メモ化–

プログラム

```
a = 3; b = 1; c = 5;
x = f (a, b, c);
a = 2; b = 9; c = 1;
y = f (a, b, c);
a = 3; b = 1; c = 5;
z = f (a, b, c);
```

関数	入力			出力	
f	3	1	5	23	
f	2	9	1	17	

演算済み

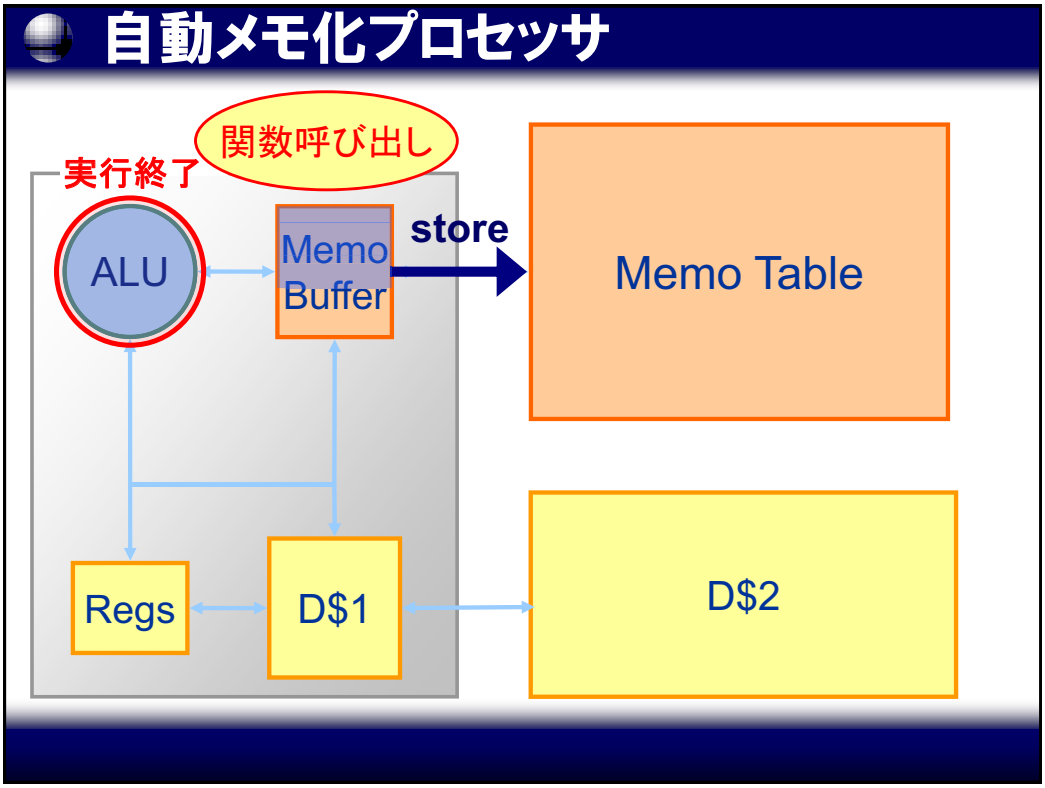
実行時間

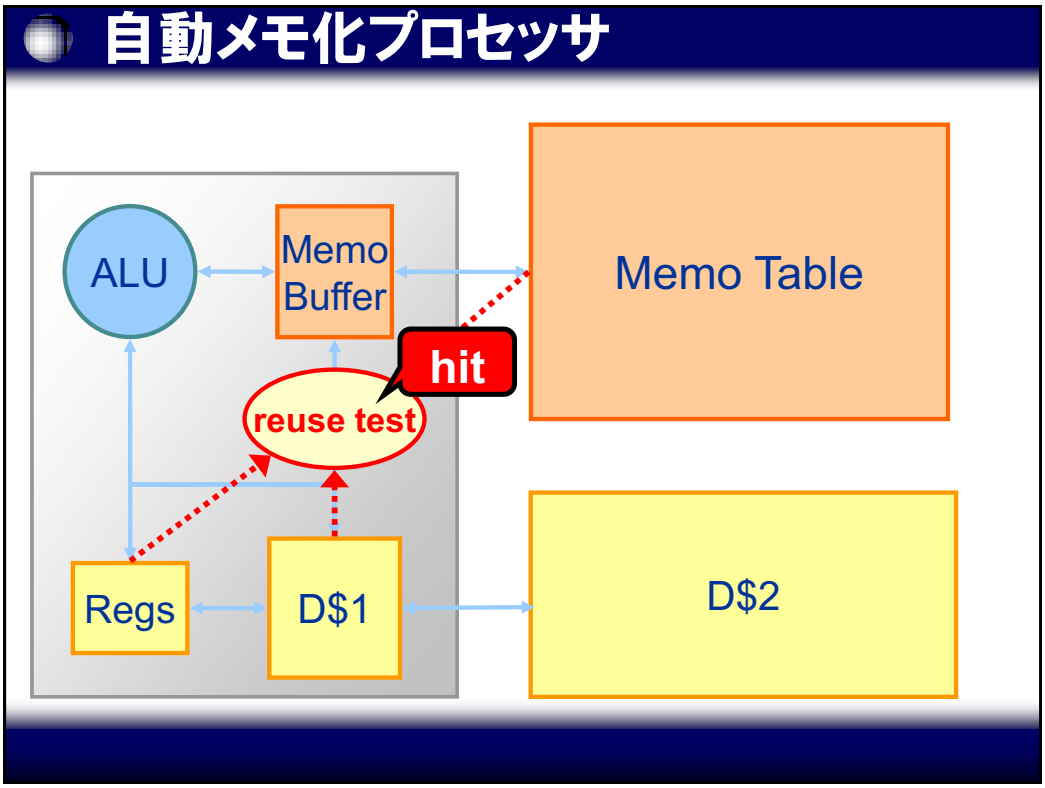
f (3, 1, 5);

f (2, 9, 1);

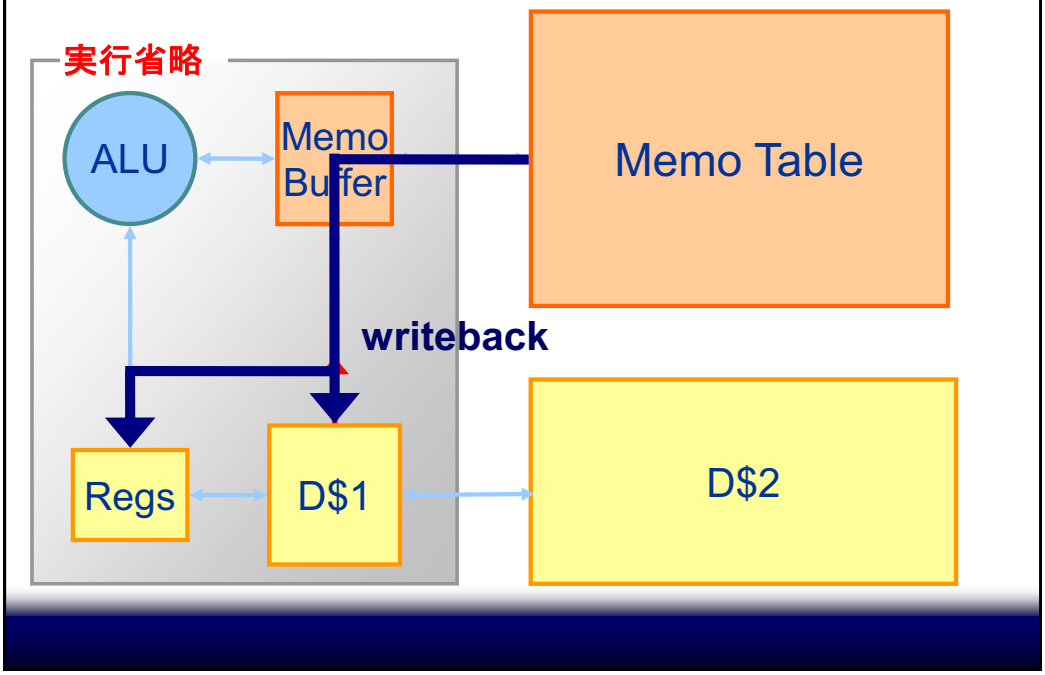
高速化

f (3, 1, 5);

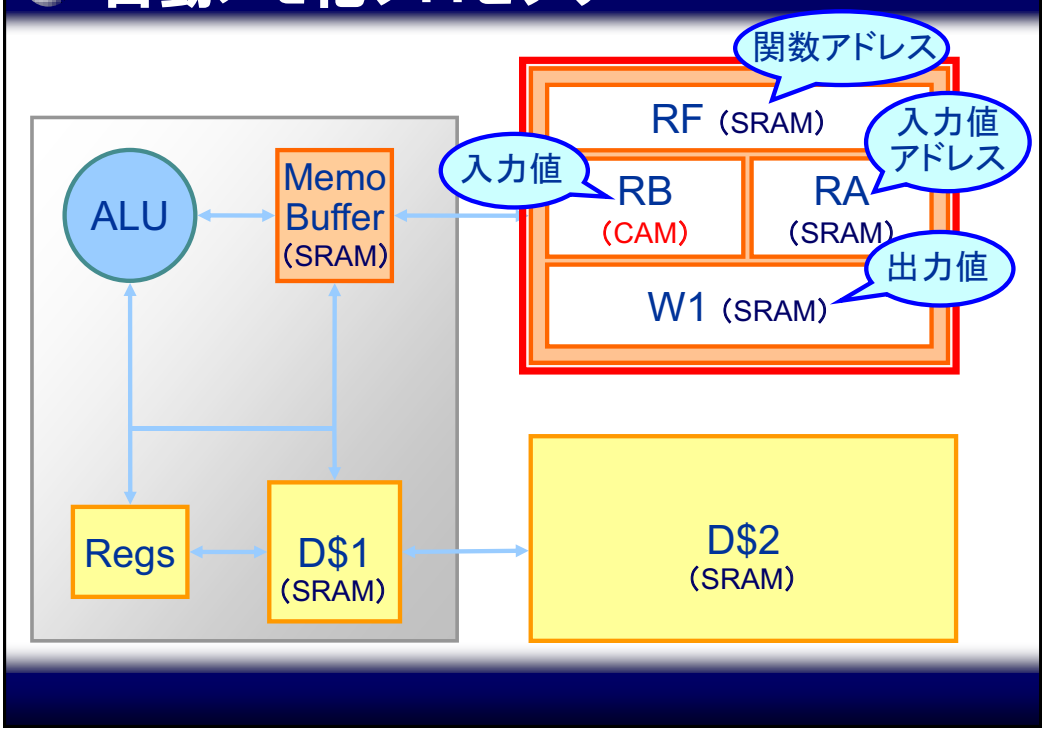

自動メモ化プロセッサ
関数呼び出し
実行終了
ALU
Memo Buffer
store
Memo Table
Regs
D$1
D$2


自動メモ化プロセッサ
ALU
Memo Buffer
Memo Table
hit
reuse test
Regs
D$1
D$2

自動メモ化プロセッサ

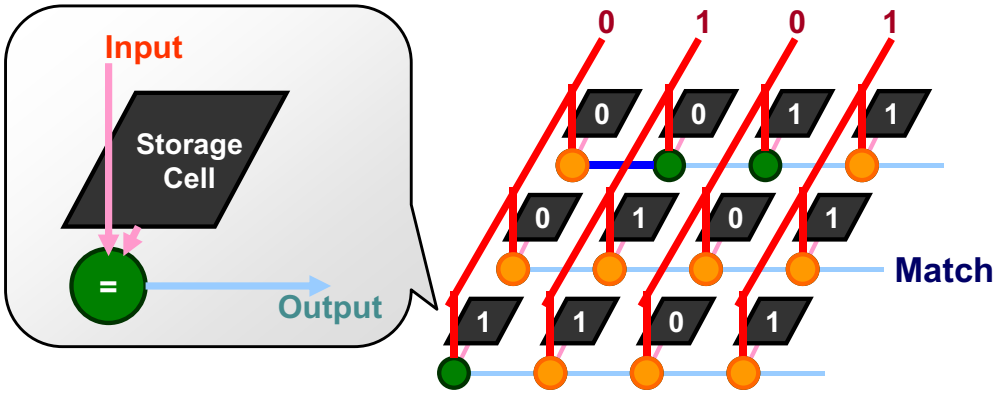


自動メモ化プロセッサ

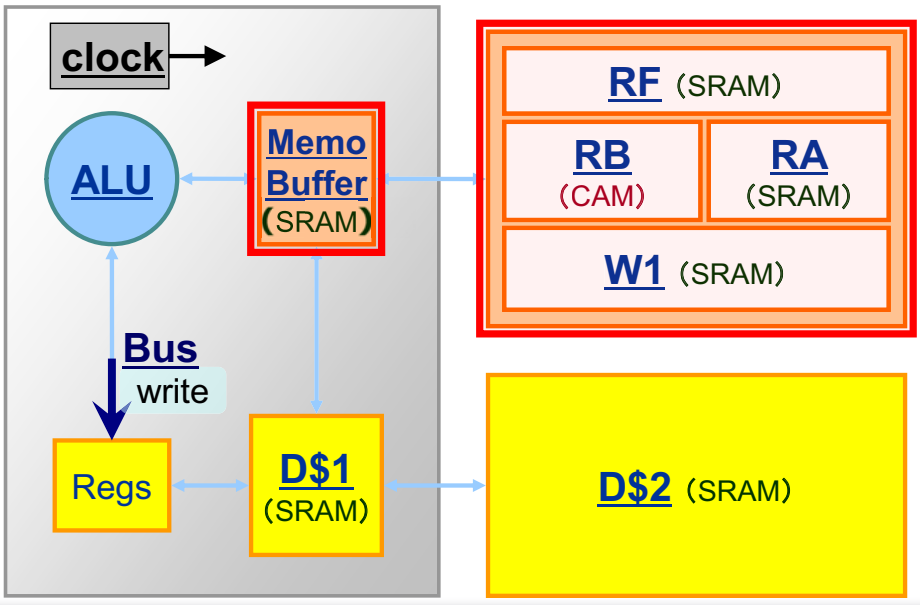


CAM（連想メモリ）

- 高速な検索の必要性
  - 検索オーバヘッドが大きいと意味がない
  - CAM (Content Addressable Memory)



エネルギー消費ユニット



シミュレーション環境

D1 Cache	**容量**	**32**	**KBytes**
	ラインサイズ	32	Bytes
	ウェイ数	4	
	ミスペナルティ	10	cycles
D2 Cache	**容量**	**2**	**MBytes**
	ラインサイズ	32	Bytes
	ウェイ数	4	
	ミスペナルティ	100	cycles

SPEC CPU95ベンチマークを用いて評価

→ メモ化を行わない通常のプロセッサ

→ メモ化を行うプロセッサ

メモ化機構ユニットの容量

Memo Buffer		(RAM)	19 KBytes
Memo Table	RB	(CAM)	36 KBytes
	RF	(RAM)	12 KBytes
	RA	(RAM)	25 KBytes
	W1	(RAM)	75 KBytes

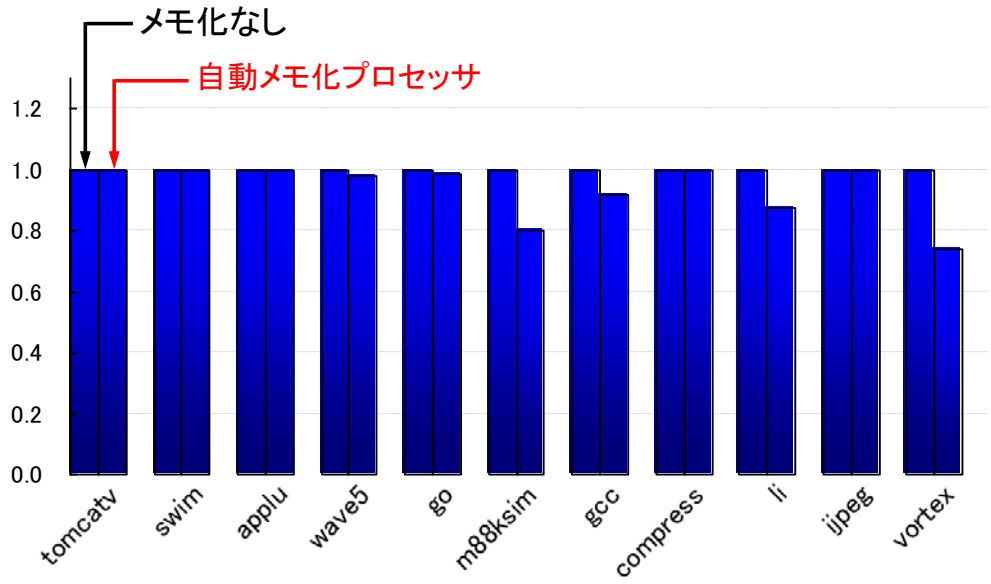
= **112** KBytes

SPEC CPU95ベンチマークを用いて評価

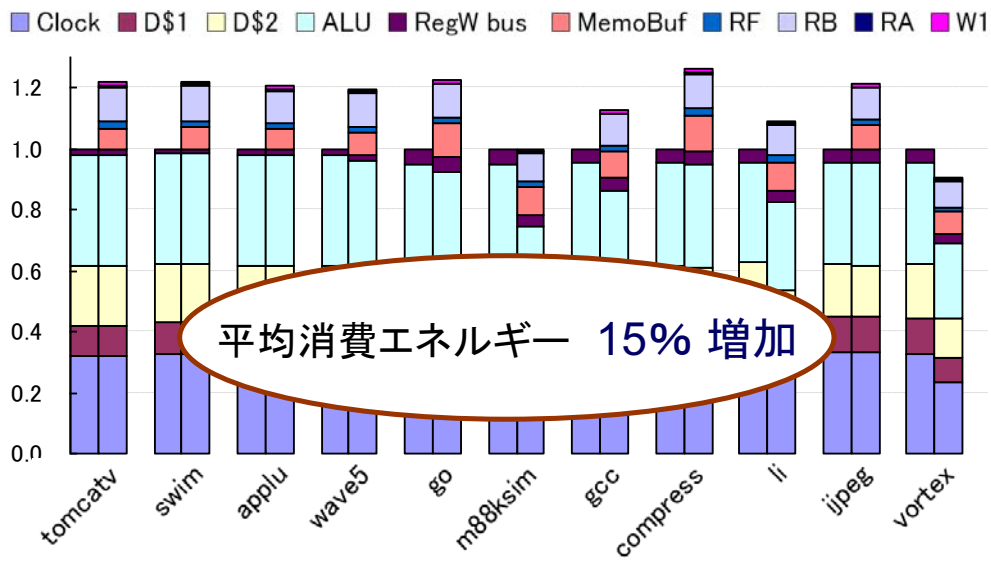
→ メモ化を行わない通常のプロセッサ

→ メモ化を行うプロセッサ

実行サイクル数比（SPEC CPU95）



消費エネルギー比（SPEC CPU95）



流れ

自動メモ化プロセッサ

↓ 電力測定機能

エネルギー評価1

↓ 消費エネルギーの削減機能

エネルギー評価2

- 計算再利用による高速化
- 高速化が起こらない場合 消費エネルギーが**増大**
- 「**メモ化中断**」による **消費エネルギーの抑制**
- 「メモ化中断」による 消費エネルギーの評価

提案 -メモ化中断-

- メモ化の**hit** 回数 を 一定間隔おき に観測
  - n_{hit}
  - 一定回数の関数呼び出し =1024回
  - 回数 n_{test}

関数呼び出し

- 閾値判定
  - n_{hit} ≧（閾値） → メモ化継続
  - n_{hit} ＜（閾値） → メモ化中断 → **電力供給を遮断**

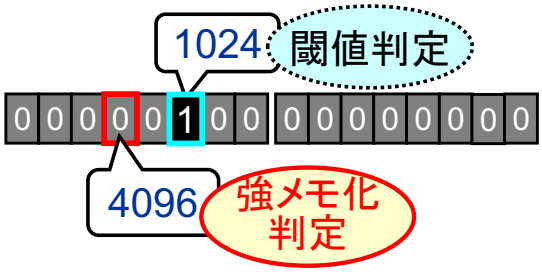
※メモ化効果が見込める場合には中断を起こりにくくする

（ n_{hit}≧閾値 が2^k 回継続　k=2 ）

カウンタの構造

メモ化のhit 回数: n_{hit}

関数のcall 回数: n_{test}

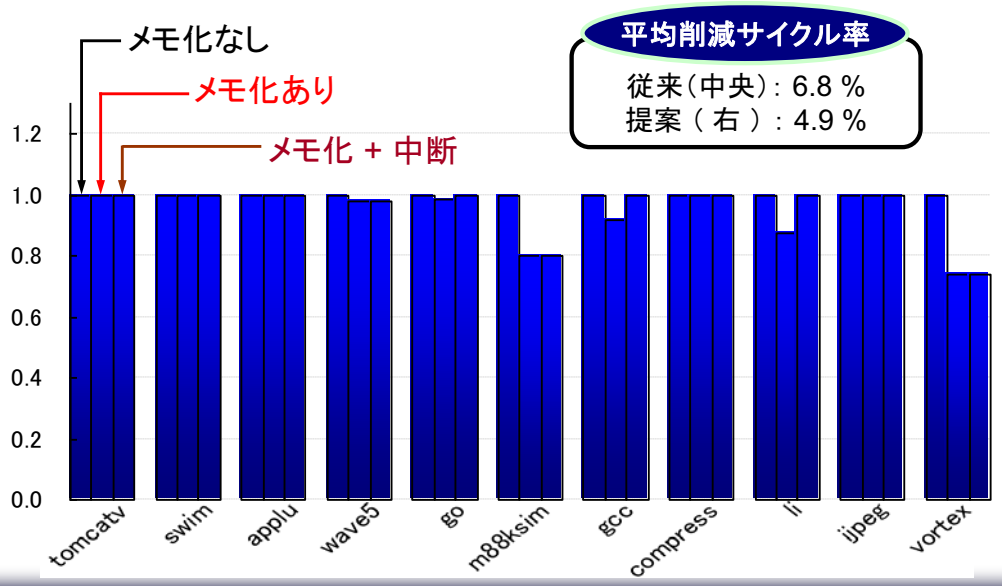


閾値判定時に n_{hit}のビットを参照すればよい

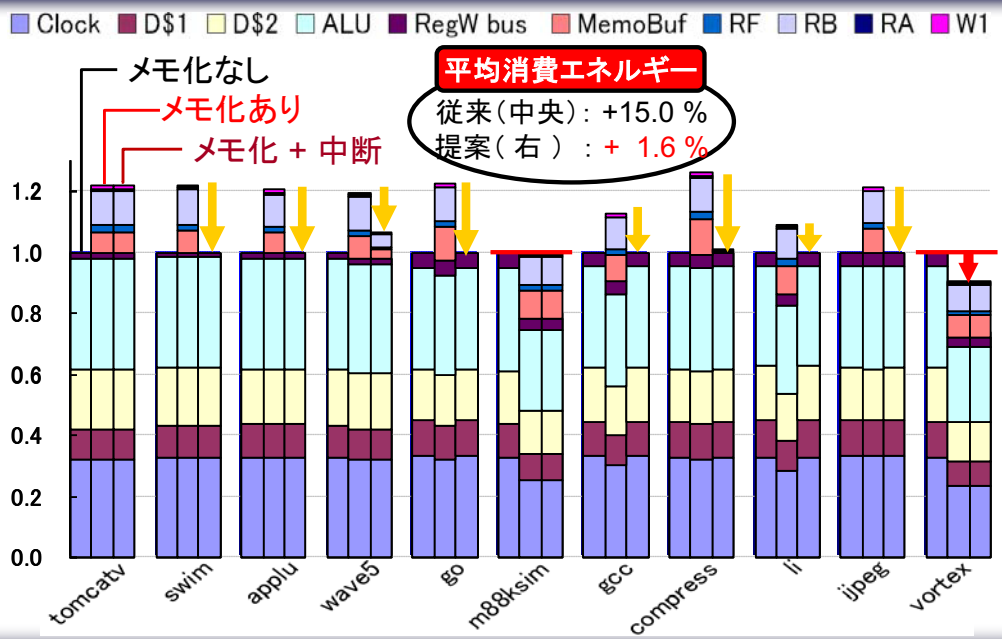
提案手法の検証　その1

- SPEC CPU95
  - 一般的なベンチマーク
- GENEsYs
  - 遺伝的アルゴリズムのベンチマーク
  - **メモ化の効果が現れやすい**

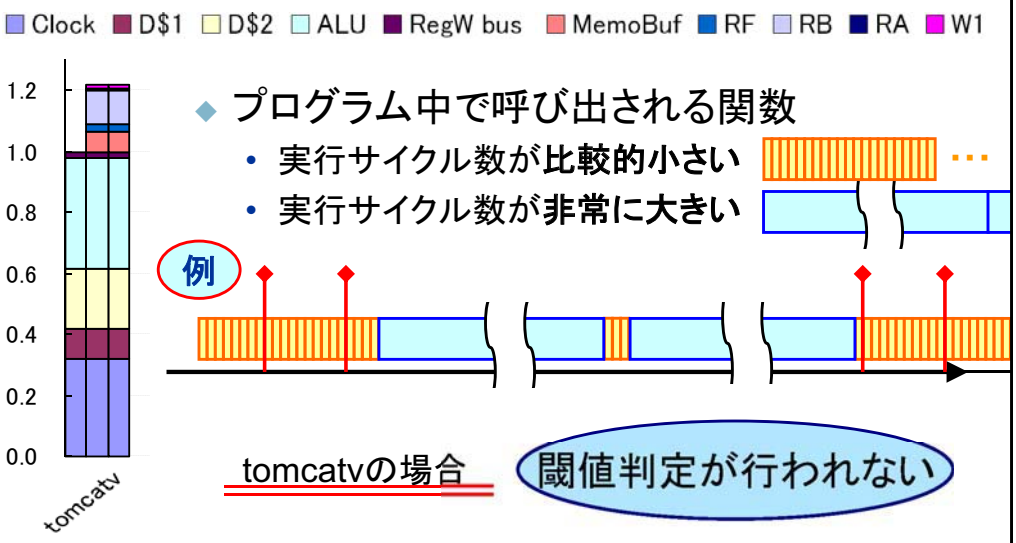
実行サイクル数比（SPEC CPU95）



消費エネルギー比（SPEC CPU95）



考察 （"tomcatv" in SPEC CPU95）



結果 （SPEC CPU95）

	従来手法	提案手法
中断プログラム数	0	8
平均削減サイクル率 （%）	6.8	4.9
平均消費エネルギー （%）	**＋15.0**	**＋1.6**
中断プログラムのみ （%）	**＋19.0**	**＋0.9**

メモ化中断により消費エネルギーを大きく抑制

提案手法の検証　その2

- SPEC CPU95
  - 一般的なベンチマーク
- GENEsYs
  - 遺伝的アルゴリズムのベンチマーク
  - **メモ化の効果が現れやすい**

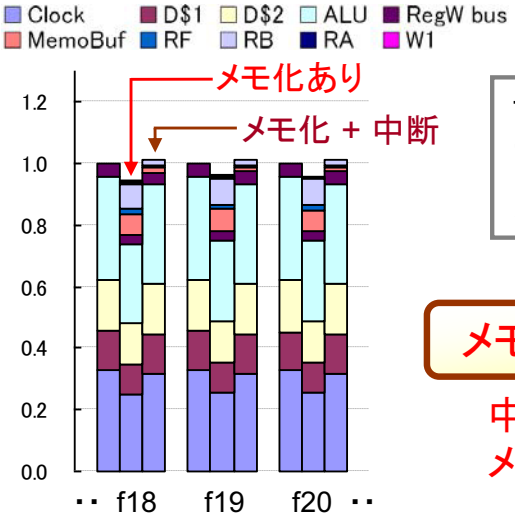
世代数	25	世代
個体数	50	
突然変異数	0.1	%
交叉率	60	%

結果　（GENEsYs）

	従来手法	提案手法
中断プログラム数	0	15
平均削減サイクル率　（%）	18	14
平均消費エネルギー　（%）	**＋1.3**	**－4.9**
中断プログラムのみ　（%）	**＋12.0**	**＋1.0**

メモ化中断により消費エネルギーが削減

課題 （GENEsYs）



f18, f19, f20 は、
プログラムの序盤では
メモ化の効果が出ない

メモ化再開アルゴリズム

中断後、一定間隔おきに
メモ化の**再開**動作を行う

まとめ

- 自動メモ化プロセッサシミュレータに電力測定機能を追加
  - 一部のプログラムでは消費エネルギーが削減
  - メモ化効果の出ないプログラムでは、メモ化機構分にあたる消費エネルギーが増大
- 提案　メモ化中断による電力供給の遮断
  - 多くのプログラムで消費エネルギーを抑制
  - SPEC CPU95, GENEsYsベンチマーク共に、提案手法により多くのプログラムで消費エネルギーが削減された

平均消費エネルギー	+15 % ⇒ +1.8%	（SPEC CPU95）
平均消費エネルギー	+1.3% ⇒ −4.9%	（GENEsYs）

センサノード向けOSにおける協調型タスクスケジューリング

松尾 英治
立命館大学大学院理工学研究科

1 研究背景

近年，無線デバイスの普及や低コスト化により，無線を用いた広範囲なネットワークやユビキタスコンピューティングが研究されている．このようなネットワークにおいて，実世界の状態を取得するための手段として，センサノードと呼ばれる無線小型端末による無線センサネットワーク技術が注目されている．

本研究は，センサノード向けOSにおけるタスクスケジューリング機構として，他ノードと協調してスケジュールを決める協調型タスクスケジューリングについて手法の提案，および実装を行う．本スケジューリングでは，従来のセンサノード向けOSにおけるタスクスケジューリングの問題点であるハードウェアの待ち時間やノード間の待ち時間を改善し，ネットワーク全体における応答性を極力損なうことなく，ノードの協調動作を踏まえたネットワーク全体における消費電力の削減を実現し，センサネットワークの高寿命化などを図る．

1.1 センサノード向けOSの特徴

センサノードは，周囲の状況を観測し無線を利用して情報を送信するのが主な機能であり，その最たる特徴として，電池など有線を使わない電源やコスト削減のための計算機資源の制限などが挙げられる．このようなハードウェア資源の制限から，センサノード向けOSは，小型かつ低消費電力な設計でなければならない．また，センサネットワークを効率よく稼動させるために，イベントを効率よく行うスケジューリングや，通信回数の軽減，無線やセンサ周りのI/O管理，柔軟性の高いネットワークのサポート，電池残量が均一になるような分散的なリソースのスケジューリングなどが必要になる．

1.2 既存スケジューリングの問題点

TinyOS[1]などに代表される既存のセンサノード向けOSの多くは，イベント駆動型のスケジューリング(図1)を利用している．イベント駆動型は，ハードウェアなどの割り込みをイベントとしてとらえ，それを起点とし連鎖的に処理を進める．イベントによって呼び出されたタスク，つまり，割り込みタスクは，それまで処理を行っていたタスクの処理を阻害しないためにも迅速に処理を終了させる必要がある．そのため，ユーザは，イベントによって呼び出されたタスク内において複雑な処理を行うのではなく，ユーザタスクを別に作成しその内部で行うようソフトウェアを設計する必要がある．通常のユーザタスクは，タスクが数珠繋ぎに終了するモデル(Run to Completion)に基づいているため，割り込み以外の処理からは阻害されずに自身の処理を完了できる．イベント駆動型は，タスク切り替えや割り込み待ちに費やすCPU時間などを軽減できるといった利点を持つ．

しかし，イベント駆動型には，次のような問題点が挙げられる．(1) 割り込み元となるハードウェアは常に動作してなければならない，(2) イベントに対する処理がいつ行われるかの保証がない，(3) たとえ単一ノード内でうまくスケジューリングされていたとしても他のノードが情報伝達に関し即反応できるとは限らない，といった点から，センサノードが長期動作する際の足枷となっている．

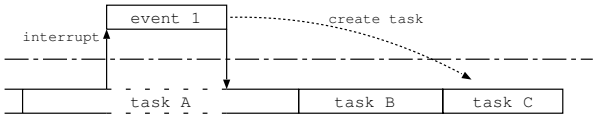


図1 イベント駆動型スケジューリングの例

イベント駆動型では，ソフトウェアが割り込みを待つ必要が無くてもハードウェアが常に割り込みを待つことになる．そのため，CPUやその周りに待ち時間が存在しなくても，原則として，割り込みを発生させる源である無線デバイスなどのハードウェアは，常に通電し動作してなければならない．これは，デバイスが通電している限り起こる問題であり，デバイス側で効率良い電力消費モデルをサポートする必要がある，

次に，イベント駆動型では，個々の内部においてのみスケジューリングを行うため，他のノードの動作状況を踏まえたスケジューリングとなっておらず，内部で処理を迅速に行ったとしても，その情報伝達がノードのネットワーク全体で迅速に行えるとは限らないといった問題を持つ．そのため，割り込みタスクが処理されてからその割り込みタスクで生成されたユーザタスクが実際に処理されるまでの時間を保障できず，ネットワークにおいて遅延や待ちの影響を把握することが困難となる．

このように，イベント駆動型は，無線デバイスによる他ノードとの相互通信やネットワーク全体におけるスケジューリングなどにおいては，依然として電力消費面において問題があり，ノードの高寿命化において難があるものと言える．

2 協調型タスクスケジューリング

協調型タスクスケジューリングは，ネットワークにおいて管理されたモデルを用いる．これは，ネットワークが部分的にせよ生存していれば良い，センシング情報が確実に取得できさえすれば良いというセンサネットワークに対する要求特徴を踏まえ，情報が厳守されるのであれば各ノードは常時に反応しなくてもよいと仮定し，ネットワークの状態を元に各ノードにおける処理の期間や順序を同期的に決定するものである．ネットワークにおいて管理することで，ノード間における無駄な待ち時間を低減や分散や同期における円滑なスケジューリングを可能とする．また，本スケジューリングにおいては，同期期間が明示されるためハードウェアがアクティブに

長期間待つ必要が無く，デバイス自体の電力消費量の低減が期待でき，総じて，ネットワーク全体の高寿命化やデータ配送の高効率化を実現する．反面，ネットワークにおいて各ノードの状態を共有するため，通信における管理コストの増加や，ネットワーク全体における高精度な絶対時刻を必要とするなどの面を持つ．

2.1 提案手法

本スケジューリングでは，複数のノードでクラスタを構成し，そのクラスタ単位で各ノードのタスクのスケジューリングを決定する．スケジューリングは，各ノード間における同期において無駄が生じないように最適化を行う．なお，各ノードがどのような状態かはクラスタ内全ノードが把握している必要があるが，他クラスタ内のノードの情報を把握する必要は無い．

本手法では，ネットワークを構成する各クラスタの範囲内でスケジューリングを決定するだけでよく，ネットワーク全体で一応に同期を取る必要は無い．そのため，同期における応答性の低下がネットワーク全体に広がり難く，スケーラビリティの高いネットワークを構築できるものと考えられる．

2.1.1 クラスタ構成

本スケジューリングにおいては，クラスタの詳細な構成について細かく規定はしない．ただし，クラスタ内のどのノードからでも隣接クラスタ内の全ノードへ通信できること，クラスタヘッドは随時変更可能なことなどを満たしている必要がある．

2.1.2 代理ノードによる伝達

本スケジューリングでは，クラスタに分けてスケジューリングを行うため，クラスタ間の通信において効率よい同期が行えない．そのため，クラスタ間における各ノードの通信においては，代理ノードを用いた非同期の通信を行うものとする．

本スケジューリングにおけるクラスタ間の通信には，代理ノードとなるクラスタヘッドが専属で通信を行う．クラスタ外部との通信は，代理ノードのみが可能であり，クラスタ別に振り分けられる識別番号によって相手の識別を行う．一般ノードは，代理ノードから外部の情報を受け取る．なお，クラスタ内の通信は，センサネットワーク全体において各ノード固有に振り分けられた識別番号によって相手を識別する．代理ノードは，電力消費量の均一化のために随時変更される．この変更は，クラスタ内部の状態遷移と共に行われ，担当の引継ぎと同時にデータも引き受けるものとする．

2.2 ノード内スケジューリングの考察

スケジューリングの詳細を決定するにあたっては，スケジューリングにおける判断要素の策定が必要となる．これら要素に関しては，センサノードの利用範囲が多く予想されるため，各々の利用における電力消費量や必要とされるプロセスの分析を詳細に行う必要がある．なお，判断要素の策定において，各要素がスケジューリングにおいてどのように反映されるかの評価項目としては，3つの項目が挙げられる．まず，センシング周期などのつながる周期性の確かさである．次に，通信やその他タスクの生成などにおける即時応答性，そして，外部デバイス含めたノード全体の消費電力の効率である．

2.2.1 スロットによる同期

本スケジューリングにおいては，ネットワークにおいて同期済みである絶対時刻をスケジューリングの同期に利用する．この絶対時刻を用いて時間をスロットという期間に分け，そのスロット毎に処理を当てはめていくことで同期を擬似的に行う．各スロットの処理の配置には，優先度や送受信同期を考慮した動的な配置が行われる．受信ノードと送信ノードの送受信処理に関しては双方同時刻のスロットに配置され，また，センシングなどはその周期を満たすよう優先的にスロットに配置される．

本スケジューリングでは，スケジューリング側で送受信などの全てのタイミングを決定するため，ハードウェアの制御をスケジューリング側にて行うことが可能となる．各タスクがどのようなハードウェア資源を必要とするかなど，処理自体に対してメタ情報を持たせることで，ユーザが意識することなくハードウェアの細かい電力制御が行えるものと考えられる．なお，スロット配置における詳細な手法に関しては現在考察中である．

2.2.2 処理モデルの設計

現在センサノード向けOSにおいては，アプリケーションという概念が希薄となっている．これは，既存OSの多くが，カーネル内部とユーザ定義関数を結びつけた，単一層による処理が中心となっているからである．そのため，OS全体が割り込みを起点としたマルチスレッドのようなモデルに近く，処理の流れが見通しづらいなどといった欠点を持つ．本スケジューリングにおいては，各ノードの処理が既知である必要があるため，このような処理の流れが一見して特定できない処理モデルは実行が困難になるものと考えられる．

そのため，本スケジューリングの実装にあたっては，そのOSにおける処理モデル自体も新規に定義する必要がある．この処理モデルに関しては，現在，遷移を中心にしたモデルを考察している．このモデルでは，タスクとなる各コンポーネント間においてデータが遷移することになる．ユーザは，コンポーネントを遷移の条件で結びつけ，それらを遷移のグラフを複数利用することで並列された処理を実装することになる．

3 おわりに

本稿では，センサノード向けOSにおける協調型タスクスケジューリングについて述べた．今後の課題としては，スケジューリングにおける同期の考察，スケジューリングの簡単な見積もりとして早期に様々なスケジューリングを仮想的に実装し評価することが挙げれる．

参考文献

[1] TinyOS, http://www.tinyos.net/

[2] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, Kristofer Pister, “System Architecture Directions for Networked Sensors”, Architectural Support for Programming Languages and Operating Systems (ASPLOS IX), pp.93-104, 2000.

センサノード向けOSにおける協調型タスクスケジューリング

立命館大学大学院 理工学研究科
大久保・横田研究室
M2 松尾 英治

発表内容

- 研究概要
  - 研究背景
  - センサノード技術の概要
  - 既存スケジューリングの問題
  - 提案手法の概要
- まとめ

2

研究背景

- 無線デバイスの小型化よるセンサの無線化
- センサネットワーク技術の研究が活発に
  - 現在のセンサネットワークは簡素な機能が中心
  - ユーザは測位データの種類と値をクエリによって参照
- 今後は様々な分野での利用が期待され、より高度な機能提供が望まれる
  - 無駄な資源消費の低減
    - 複数のノードによって処理や担当を分散
    - 冗長なデータの削除などの効率よいデータ管理
  - センサネットワーク側での自律的な動作
    - 故障時などにおける自動的な代替観測
    - トラッキングなどのオートメーション化
  - セキュリティやセンサ端末の管理支援

3

センサノードについて

- センサと無線デバイスからなる小型無線端末
  - 周囲の状況を観測し、無線を利用して情報を送信するのが主な機能
- 数百など複数のノードから観測ネットワークを構成
- 無線化とコスト削減に伴う制限の厳しい計算機資源
- 例 MICAz
  - 早くに販売され普及している、汎用型のセンサノード
  - CPU ATmega128L 7.37MHz
  - メモリ
    - 内部 128kBフラッシュ+4kB SRAM
    - 外部 512kBフラッシュ
  - 電源
    - 単三電池2本による供給

4

センサノード向けOSの特徴

- 小型 & 低消費電力
  - 消費電力において効率よいスケジューリング
  - 通信回数の軽減
    - MICA MOTEでは1bitの送信 = 800～1200命令
- 汎用性の高いモジュール設計
  - センサノードの利用状況が限定されていない
  - 無線デバイスやセンサなど搭載デバイスも不定
  - コンポーネント指向モデル
- 無線やセンサ周りのI/O管理
- 柔軟性の高いネットワークのサポート
  - ネットワーク構成やプロトコルスタックにおける自由度
- 分散的なリソースのスケジューリング
  - 電池残量が均一になるような処理の分担

5

イベント駆動型スケジューリングについて

- イベント駆動型の特徴
  - 割り込みを起点として処理を行う
  - 複雑な処理はユーザタスクにおいて行う
    - ユーザタスクはRun to Completionによる実装が殆ど
  - ハードウェアに対するソフトウェアの待ち時間が生じ難い

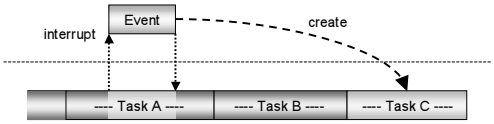


図1 イベント駆動型スケジューリングの例

6

既存スケジューリングの問題点

- 往々にして簡素すぎる
  - イベント駆動によるシングルタスク
  - I/OなどにおいてTPOに合った処理が困難
- ネットワークを考慮した設計ではない
  - ノード内のハードウェア割り込みが中心
    - ノード間における処理の同期効率はユーザ側に依存
  - ノード間においてユーザタスク同士での同期が取り辛い
    - スケジューリングだけではなく通信においても遅延などの障害が発生
    - 単純なwake upモデルでは1:mの処理が困難
  - 同期処理が複雑になり同期処理自体の保障も困難に
- ハードウェアが常に動作している必要がある
  - デバイスを完全に停止し待機電力を減らすことができない
  - 少量であれ無視できない　→　待機電力を省くだけで倍以上の稼動

7

センサネットワークにおける処理の特徴

- センサノード間における通信の形態
  - 定期通信
    - センシングデータの受け渡しやノード状態の監視など
    - ボトムアップな通信が中心
  - 突発的なイベント通信
    - ノード同士の連携における自発的なイベント処理など
    - ボトムアップだけではなく横への通信も重要に
- 注目する点
  - ネットワークが部分的にせよ生存していれば良い
  - センシング情報が確実に取得できればよい
  - 多くの場合複数のノードで連携が可能である

  - 個々のノードが常時反応する必要はない

8

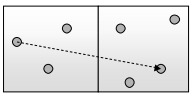
協調型タスクスケジューリングのコンセプト

- ネットワークにおいて管理されたモデル
- ネットワークの状態を元にノードの処理順序を決定
  - ノードをまたぐ処理における無駄な待ち時間の低減
    - 通信における再送処理などの低減
  - 分散や同期における円滑な処理
    - ノード間における複雑な同期処理への対応
- ハードウェアの待ち時間を低減
  - ハードウェアの必要期間が既知に
  - 無線やセンサなどにおける待機電力の低減
  - OS側でハードウェアの制御が可能に

9

協調型タスクスケジューリングにおける前提条件

- 前提となる条件
  - クラスタ技術
    - クラスタ化されたネットワークモデル
    - 隣接クラスタ間の全ノードが電波到達距離内に位置



    - クラスタの詳細な方法については特に規定はしない
  - 時刻同期技術
    - 高精度な時刻同期が行われていることが前提
      - 精度は利用状況に依存
    - 同期の詳細な手法については特に規定はしない

10

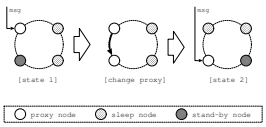
協調型タスクスケジューリングの概要

- クラスタ単位でスケジュールを決定
  - スケジューリングはクラスタの状態によって変化
  - クラスタの状態はクラスタ内でのみ共有
    - クラスタ間では共有しない
- ノード間の同期処理を中心に処理の優先度を決定
  - 通信、センシングにおける同期処理
  - ネットワーク管理などの突発な制御処理も考慮
- ハードウェアの電力制御をOSが担当
  - 同期により通信期間などが既知に
  - 通信デバイスの停止が容易に
  - 電力消費量の低減

11

代理ノードによる伝達

- クラスタ間の通信には、クラスタヘッドとなる代理ノードが専属で通信を行う
  - クラスタ外部との通信は代理ノードのみが可能
    - クラスタ間の通信はクラスタの識別番号によって選別
  - 一般ノードは代理ノードから外部の情報を受け取る
    - クラスタ内の通信はノードの識別番号によって選別
- 代理ノードは随時変更される
  - クラスタ内部の状態遷移と共に担当ノードが変更される
  - 担当の引継ぎと同時にデータも引き受ける

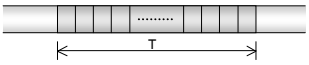


- 代理ノードを用いることで、クラスタ内部の状態に関わらずクラスタをまたぐノード間での通信を保証する

12

スロットによる同期

- スロットという時間枠を設けて処理を当てはめていく
  - ネットワークにおいて同期済みの絶対時刻を利用



- 処理は動的に各スロットへ配置される
  - 配置には優先度を考慮する
    - 受信-送信の処理が同時刻のスロットに位置しなければならない
  - 依存するノード間においていかに同期させるかが問題
    - 極力同期が保証される配置アルゴリズムによって解決
- 各スロットごとの役割が明確に
  - 処理自体にメタ情報を持たすことで対応
  - スケジューリング側による電力制御が可能に

13

スケジューリングにおける評価項目

- スケジューリングおける評価項目
  - 周期性
    - センシング周期の保証
  - 即時応答性
    - 制御信号や連携したセンシングなどにおける応答性
  - 電力消費量
    - 外部デバイス含めたノード全体の消費電力
- 各評価項目の重要性
  - 利用状況によって項目の重要度が変化
    - 車の追従モニタリングでは応答性が最重要に
  - 各々の項目どれを優先すべきはユーザが選択できるように設計すべきである

14

センサネットワークに合った処理モデルの検討(1)

- 現在アプリケーションという概念が希薄
  - 多くのOSがファームウェアのような位置づけ
    - カーネルという明確な層が無い、もしくは希薄
    - カーネルを置き換え、書き換えることでコードを記述
  - センサ結果をノード内部で加工し、他のノードで利用することは想定していない
- アプリケーションを記述するのが困難
- アプリケーションに対する制約が無いため、OS側の考察すべき項目も多岐に

15

センサネットワークに合った処理モデルの検討(2)

- センサノードに合った処理モデルの導入
  - 単一ノードだけではなく、複数のノードを視野に
    - ノード同士の連携をスムーズに定義できるように
  - 見通しの良いモデル
    - 処理の流れが一見して分かるのが望ましい
    - 処理の流れが既知になればスケジューラにとっても処理し易い
- 現在考察中の処理モデル
  - 遷移モデル
    - データやメッセージを元に次のタスクへ遷移
    - 並列化には遷移図を複数用いることで対応
    - 自由度は低くなるが、何によってどう動くかが明確に

16

まとめ

- 発表内容
  - 協調型タスクスケジューリングについて
    - 研究背景
    - センサノード技術の概要
    - 既存スケジューリングの問題
    - 提案手法の概要
- 今後の検討課題
  - スケジューリングにおける同期手法の考察
  - スケジューリングの評価と実装

17

性能モデル学習と最適化機構を有する省電力化スケジューラ

金井 遵†
†東京農工大学大学院 工学府 電子情報工学専攻

1 はじめに

近年，計算機の複雑化に伴い，消費電力の増大や発熱が問題となる一方，計算機のユビキタス化によって，消費電力の削減に注目が集まっている．しかし，cpufreq などをはじめとした，従来の OS で利用される CPU の省電力化機能は，DVFS 機能を用いてバッテリ残量や CPU 使用率からシステム全体で一律に周波数・電圧制御を行う方法が中心であった．プロセス単位で性能要件が異なる場合や，CPU，メモリ，I/O バウンドアプリケーションの混在など，最適な動作周波数が異なるアプリケーションが並列動作する場合などにおいてはこれらの手法では不十分である．また，性能予測を行い，省電力化を行うような高度な省電力化環境は，環境に応じて性能のモデル化を手動により行わなければならない場合が多く，多くの環境への適用が困難であった．

そこで本研究では，性能予測モデル構築を自動化し，さらにこの性能予測モデルおよび実行時の性能最適化機能を用いて，性能要件の中でプロセス単位で省電力化を行うスケジューラを設計し，実際に Linux システムに実装を行った．本稿ではこれらの設計・評価の詳細について述べる．

2 統計情報に基づく性能予測

本スケジューラでは回帰分析により，定量的に性能予測式のモデル化と性能予測を行う．多くの計算機環境では，メモリアクセスや I/O アクセスが性能に大きく影響する．特に，近年のプロセッサにおいては，パフォーマンスカウンタ (PMC) が実装されていることも多く，キャッシュミス頻度などといった情報（以下，PMC 要素，PMC 値と呼ぶ）をプロセッサから取得可能であるほか，I/O アクセスの頻度は OS から取得可能である．そこで，これらの頻度と性能の関係を回帰分析により求め，定量的に性能予測式のモデル化を行う．さらに，性能予測時には周波数 u 毎にモデル化した次式から性能予測を行う．

$$Y^u = b_0^u + \sum_{q=1}^{p} b_q^u X_q \qquad (1)$$

ここで，回帰分析の説明変数 X_q としては，PMC 要素を実行命令数で正規化したものを用いる．これにより，同一コードを実行すれば，クロック数に依らず説明変数の値は等しくなる．また，目的変数 Y^u としては，対最高周波数の時間あたりのユーザランド実行命令数比 (UIPS 比) を用いる．UIPS 比は，I/O バウンドプロセスでは，周波数を下げた場合でも 1 に近く，CPU バウンドプロセスでは周波数比に近づく．学習により，回帰係数 b_q^u をあらかじめ求めておき，(1) 式に値を当てはめることで，性能予測を行うことができる．

説明変数としては，複数の説明変数を取ることができ，複数の律速要素がある計算機環境においても適用可能であるほか，回帰分析を行う際に，さまざまな PMC 要素について回帰分析を行い，相関係数を比較することで，律速要素の自動的な選択も可能となる．

3 省電力化スケジューラの設計と実装

3.1 スケジューラの全体構成

本スケジューラは，プレゼン資料 p.6，8 における図のように，性能予測式から省電力化を行う省電力モードと，性能予測式を自動的にモデル化する学習モードからなる．これらのモードおよび，後述する種々の設定は，Linux システムにおいては，システムコールまたは procfs により，プロセス単位で切替え可能である．

省電力モードでは学習モードでモデル化した性能予測式を用いて，ユーザが与えた性能制約の中でプロセス毎に省電力化を行う．スケジューラはパフォーマンスカウンタ管理機構，フィードバック機構，性能予測機構，プロセス単位 DVFS 制御機構からなる．一方，学習モードでは，省電力モードで用いる回帰方程式の学習を行う．スケジューラは性能予測式学習機構と，パフォーマンスカウンタ管理機構，プロセス単位 DVFS 制御機構からなる．次節以降では各機構の詳細について述べる．

3.2 性能予測機構

性能予測機構では，プロセス単位でパフォーマンスカウンタの集計を行い，2 章で述べた性能予測モデル (1) 式に当てはめることで各周波数に変更した場合の性能予測を行う．さらに，ユーザが与えた性能閾値の範囲内で最も低い周波数を選択し，省電力化を行う．ここで，性能閾値は，最高周波数動作時の性能に対する相対比として指定する．例えば，最高周波数である処理が 10[s] で終わるとき，処理が 20[s] で終わればよい場合には，性能閾値は 0.5 となる．(1) 式における目的変数 Y^u は最高周波数に対する UIPS 比となるが，UIPS 比と時間基準の性能比には強い相関関係があることが解っている．

Energy-Efficient Scheduler with Adaptive Performance Prediction and Performance Optimization
Jun Kanai†
Graduated School of Computer and Communication Science, Tokyo Univ. of Agri. and Tech. (†)

3.3 フィードバック機構

前項で述べた性能予測のみでは，複雑化する計算機において予測精度に限度がある．そこで，実性能を測定し，理想性能との差を補正するフィードバック機能を設計・実装した．ここで，理想性能とは最高周波数時の UIPS と性能閾値の積とする．この値と，動作周波数下での実 UIPS を比較し，理想性能＞実性能となる場合には，周波数を一段階上げることで性能要件を満たすようにし，理想性能＜実性能となる場合には，周波数を一段階下げることでさらに省電力化を行う．

なお，基準となる最高周波数時の UIPS はプログラムの処理内容によって逐次変化する．そこで，この変化を自動的に検出し，再度基準性能の計測を行う．省電力モード時のスケジューラの状態には，プレゼン資料 p.9 の図のように基準性能を測定する測定ステート，PMC の値から性能予測を行う予測ステート，実性能と理想性能の差を補正するフィードバックステートの 3 状態があり，実性能の変化，予測性能の変化の検出に応じて，ステート間を遷移する．

3.4 性能予測式学習機構

本スケジューラは，プロセス単位で動作周波数毎にパフォーマンスカウンタと性能の統計を取り，プロセス切り替え毎に回帰分析を行い，性能予測のための回帰方程式の回帰係数を算出し，性能をモデル化する．つまり，(1) 式における係数 b_q^u を最小二乗法により求める．各周波数 u について，UIPS 比 Y_i^u と，PMC 値を正規化した値 X_{n1}^u をコンテキストスイッチ毎に取得し，回帰係数を毎回更新する．例えば，単回帰分析においては次式により回帰係数を更新することができる．

$$b_1^u = \frac{\sum(X_{i1}^u - \overline{X_1^u})(Y_i^u - \overline{Y^u})}{\sum(X_{i1}^u - \overline{X_1^u})^2} = \frac{\sum X_{i1}^u Y_i^u - \overline{Y^u}\sum X_{i1}^u - \overline{X_1^u}\sum Y_i^u + n\overline{X_1^u}*\overline{Y^u}}{\sum {X_{i1}^u}^2 - 2\overline{X_1^u}\sum X_{i1}^u + n\overline{X_1^u}^2} \quad (2)$$

$$b_0^u = \overline{Y^u} - b_1^u \overline{X_1^u} \quad (3)$$

重回帰分析においても同様に，平均値などいくつかの統計データのみを保存しておけば，過去の説明変数，目的変数を保持する必要はなく，計算量およびメモリ使用量は説明変数の個数のみにより決定される．

3.5 実装

本方式自体は汎用であるが，プロトタイプ実装として Linux を選び，本スケジューラを実装した．Linux カーネルは 2.6 系をターゲットとし，カーネルパッチとローダブルカーネルモジュールとして実装した．カーネルへの変更は数行のみであるため，カーネルバージョンアップ時の追随も容易である．また，アーキテクチャ依存のコードと，非依存のコードを明確に分離し，様々なアーキテクチャへの対応も容易となっており，実際に x86(Dothan コア PentiumM)，M32R，ARM(XScale) 対応版が稼働中である．説明変数は先行研究 [1] より，L2 キャッシュミス率による単回帰分析とする．学習させるプログラム，省電力化を行うプログラムに変更の必要はなく，通常実行するのみでよい．

4 評価

本スケジューラに関して，消費電力量と性能精度に関する評価を行った．評価には，PentiumM 2GHz と，Linux 2.6.18 を利用した．あらかじめ，MiBench を用いて学習を行い，ベンチマークとしては MiBench，SPEC CPU2000，および I/O バウンドであり，性能予測が外れるディスク読み込みを行うプログラムを利用した．また比較として，学習を無効にし，周波数比がそのまま性能比になるとして性能予測を行うモード，学習を有効・フィードバックが無効となるモード，学習を有効・フィードバックが有効となるモードで消費電力量と性能精度の測定を行った．消費電力量は常時最高周波数で動作した場合の消費電力量を 100%とした相対比として，性能精度は実性能の理想性能との誤差（0%を理想性能とし，正で理想性能より性能が下回る，負で理想性能より性能が上回る）として示す．本スケジューラはフィードバックを掛ける際に，許容する誤差を設定するが，今回，± 5%としている．

性能閾値 0.9 における結果は，プレゼン資料 p.14〜16 に示したとおり，全てのモードにおいて消費電力の削減を達成している．また，殆どのベンチマークにおいて，学習効果を発揮している．特に，性能予測が当たる Matrix，QuickSort，164.gzip，176.gcc などにおいては，学習のみが最も消費電力量を削減できている．フィードバックを有効にすると消費電力が増えるのは，最高周波数での基準性能計測のためであり，学習のみと学習＋フィードバックを切替えるなどの手法が有効と考えられる．また，I/O バウンドで学習が外れるディスク読み込みや，CPU・メモリバウンドでありながら性能予測が外れる，FFT や，171.swim，183.equake ではフィードバックが有効に働く．性能閾値の範囲内で，183.equake では非学習に比べ 20%，学習のみに比べ 10%，ディスク読込みでは非学習に比べ，45%のさらなる省電力化が達成できた．181.mcf では，学習のみで消費電力を最も削減できているが，性能閾値を実性能が大幅に下回っている．ここでも，フィードバックを有効にすることで，性能閾値の範囲内で非学習に比べ省電力化を達成した．以上は，本スケジューラの学習効果とフィードバックの有効性を示す結果といえる．

5 考察および今後の課題

自動的に性能をモデル化し，性能予測とフィードバック機構による実行時最適化により，プロセス単位で省電力化を行う Linux スケジューラを設計・実装した．これにより，多くの環境に，様々なアプリケーションに対して適用可能な省電力環境を容易に実現できるようにした．今後の課題として，SMT/SMP への対応，OS から得られるデータを利用した I/O バウンドアプリケーションでも利用可能な性能予測式への拡張などを挙げることができる．

参考文献

[1] 佐々木 他: 統計情報に基づく動的電源電圧制御手法, 情報処理学会論文誌, Vol.47, No.SIG18 (ACS16), pp.80–91(2006).

性能モデル学習と最適化機構を有する省電力化スケジューラ

金井 遵

東京農工大学 工学府
電子情報工学専攻 並木研究室

2007/09/13 JSASS 2007

背景

- 計算機の複雑化に伴い，消費電力増大
- 省電力化の重要性
- 従来、OSで利用されるCPU省電力化機能（cpufreqなど）は，バッテリ残量やCPU使用率からシステム全体で一律に電圧・周波数制御
  - プロセス単位で性能要件が異なる場合，最適な周波数が異なる場合は利用に適さない
  - 周波数変更時の性能予測を行えず，性能条件のあるアプリケーションでの利用は適さない
  - 高度な省電力化環境は，多くの環境への適用が困難

目的

- さまざまな環境に容易に適応できる省電力化のための基盤ソフトウェアの提供
  - 多くのソフトウェア（CPU/メモリ/IOバウンド，リアルタイムアプリ…）に対して利用可能に
    - 性能要件や周波数・性能特性の異なるプログラムの動作が考慮されたCPU省電力化スケジューラの実現
  - 多くの環境(アーキテクチャ，SMP/SMT…)に適用可能に
    - スケジューラが利用する性能予測モデル構築の困難性解決

携帯電話・組込み機器からHPC，サーバ用途まで，
適用可能な省電力基盤を

省電力スケジューラの概要

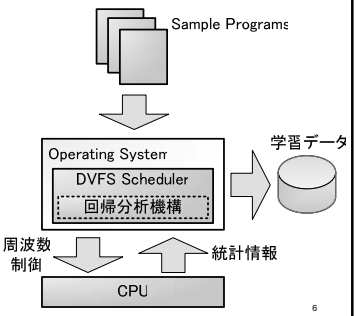
- 性能予測モデルの学習機能と，学習結果に基づいた省電力化機能
  - 性能に大きく影響するハードウェアカウンタ（PMC）の要素（ex.L2キャッシュミス回数）から各周波数時の性能をモデル化
    - 自動的に性能とカウンタの関係を定量的方法でモデル化
      → 性能予測モデル構築の容易化
    - プロセス単位で性能条件の中で周波数選択
      → プロセス単位での最適な省電力化
- 実行時に性能予測と実性能の差を補正する最適化機能
  - 単一性能予測式では，複雑化する計算機の性能予測に限界
  - さらなる省電力化と性能要件達成

用語定義

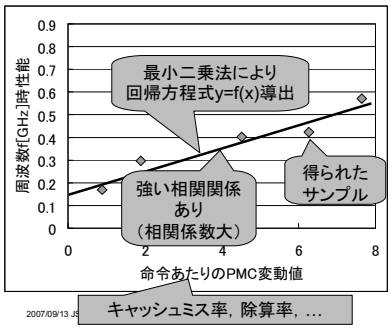
- 性能条件（性能閾値）
  - 同一プログラムが最高周波数で動作した場合と，ある周波数で動作した場合の処理終了に要する時間の比
    - ex.) プログラムAが最高周波数において10[s]で終了するとき，性能閾値を0.5と与えた場合，20[s]で終了すればよい
- UIPS
  - Usermode Instructions Per Second
    （時間当たりのユーザモードでの命令実行数）
  - 同一プログラム実行時，
    周波数変更前後のUIPS比≒時間基準比の性能

システムの概要図（学習時）

- 各周波数下でのPMCと性能の関係式を導出
  - コンテキストスイッチ毎（プロセス単位）に性能とPMCの関係を回帰分析，学習データ保存
  - 学習が容易
    - 学習させるプログラムに変更不要
    - 通常実行するのみ



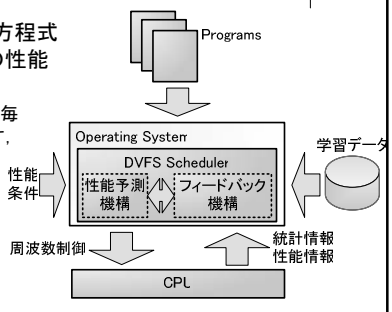
回帰分析による性能のモデル化



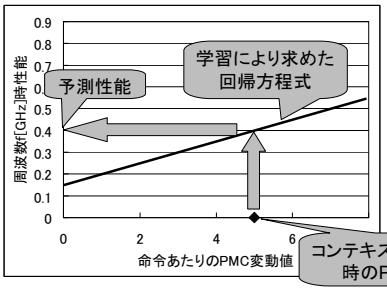
- 重回帰分析により、複数の律速要素にも対応可
- 相関係数比較による律速要素の定量的な決定可
- 目的変数（縦軸）：対最高周波数のUIPS比
- 説明変数（横軸）：命令あたりのPMC変動値
  （周波数非依存）

システムの概要図（省電力化時）

- 学習で求めた回帰方程式から，各周波数での性能予測
  - コンテキストスイッチ毎に性能条件を満たす，最低周波数を選択
  - プロセス単位で省電力化
- フィードバック機構
  - 予測性能と実性能の誤差補正



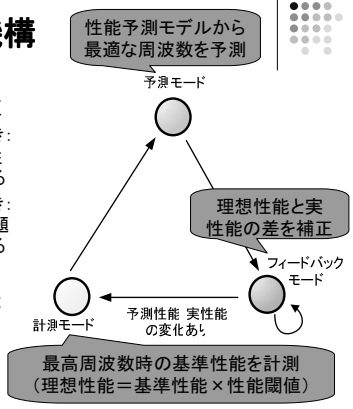
学習結果による性能予測



- コンテキストスイッチ毎(プロセス単位)で統計取得・性能予測
  - 様々なプログラムの並行動作可
  - 未知のプログラムの性能予測可

フィードバック機構

- 実性能と性能閾値(理想性能)の差を実行時補正
  - 実性能＞性能閾値のとき: さらに省電力化の可能性 →周波数を一段階下げる
  - 実性能＜性能閾値のとき: 性能条件の達成度で問題 →周波数を一段階上げる
- プログラムの処理単位(自動検出)で基準性能UIPSを予め計測．理想性能と実性能を比較



プロトタイプ実装

- OSはLinuxをターゲット
  - Linux Kernel 2.6系
    - 2.6.18, 2.6.19, 2.6.21, 2.6.22で動作
    - カーネルパッチとローダブルカーネルモジュールにより提供
    - 既存カーネルへの変更は数行，移植容易な構造
- i386，ARM，M32Rアーキテクチャ対応版稼働中
  - 方式自体は汎用
  - 他アーキテクチャへ移植容易な構造
- 性能学習・予測に用いるPMC要素は(L2)キャッシュミス率
  - 先行研究 [佐々木 et al, 2006]

実行例

- 例) 行列演算とFFTで学習，
  QuickSortで学習データを元に省電力化

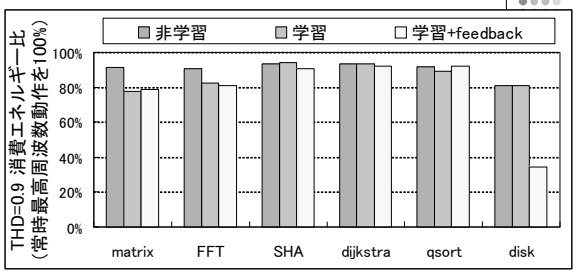
```
# echo STUDY > /proc/dvfs/mode     ・・・ 学習モードに設定
# ./matrix
# ./FFT
# echo AUTO > /proc/dvfs/mode      ・・・ 省電力モードに設定
# echo 80 > /proc/dvfs/threshold   ・・・ 性能閾値設定
# ./qsort
```

- 設定はprocfs or システムコールを介して行う
- 学習モード・省電力モード共に，プログラムへの変更不要

性能評価

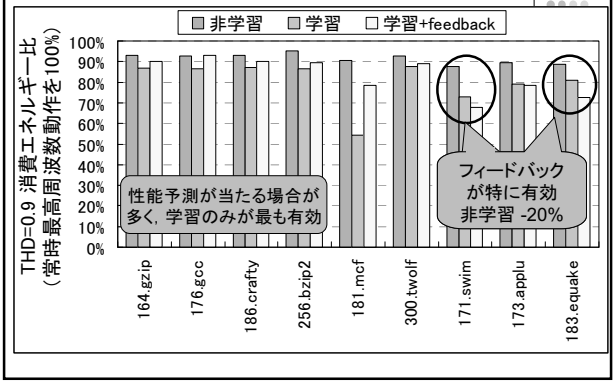
- 利用ベンチマーク
  - 行列乗算, MiBench(FFT, SHA, Dijkstra, QuickSort), SPEC CPU2000, ディスク読込み
- 以下の方式を比較
  - 非学習方式：性能予測にPMCの値を利用しない 周波数比がそのまま性能比になると仮定し周波数選択
  - 学習方式 ：学習データ(PMC)による性能予測＋フィードバック無効
  - 学習+feedback方式：学習＋フィードバック有効
- 測定環境
  - PentiumM(Dothan) 2GHz, Memory 1GB, Linux 2.6.18
  - 性能学習は予め行列乗算/MiBenchを実行
  - 消費電力は非接触型電力測定装置により実測

CPU消費電力量(MiBench・他)

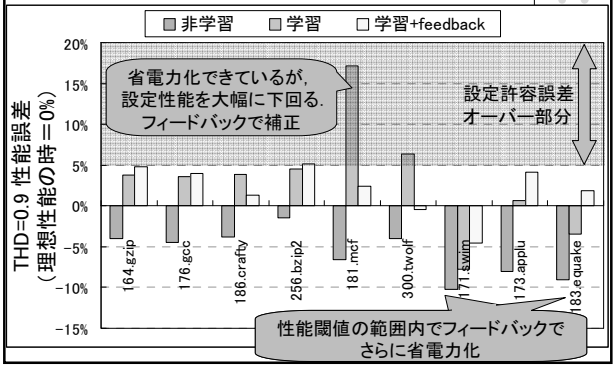


- すべてで省電力化達成
- 学習により, CPUベンチでは非学習に比べ最大12%省電力化
- 性能予測が外れるdiskで特にフィードバック機構の効果発揮

CPU消費電力量(SPEC CPU2000)



予測精度とフィードバックの有効性



まとめ

- 自動的に性能をモデル化し，性能予測と実行時最適化（フィードバック）により，プロセス単位で省電力化するLinuxスケジューラを設計・実装
  - 性能予測には定量的なモデル化手法を採用し，モデル化が容易かつ，精度の高い性能予測精度を実現
- 評価より性能予測のみで最大58%，フィードバック機構併用で最大65%の消費電力削減を達成
  - 性能予測はCPUバウンド・メモリバウンドなベンチマークで特に有効．省電力化に寄与
  - フィードバック機構は，精度向上と特に性能予測が外れるベンチマーク（特にI/Oバウンドとなるdiskベンチ）での省電力化に寄与

Joint Symposium for Advanced System Software の歴史

JSASS は，システムソフトウェアを中心として，先進的な基盤ソフトウェアに関するテーマについてオープンな議論をするためのシンポジウム. 2000 年から毎年 8～9 月頃に各地で開催している.

2000 年度（第 1 回)

2000 年 9 月 15 日(金)・16 日(土)

立命館大学 びわこ・くさつキャンパス（滋賀県草津市)

2001 年度（第 2 回)

2001 年 8 月 27 日(月)・28 日(火)

東京農工大学 小金井キャンパス（東京都小金井市)

2002 年度（第 3 回)

2002 年 9 月 9 日(月)・10 日(火)

和歌山大学（和歌山県和歌山市)

2003 年度（第 4 回)

2003 年 9 月 4 日(木)・5 日(金)

株式会社エスアールアイ（和歌山県西牟婁郡白浜町)

2004 年度（第 5 回)

2004 年 9 月 9 日(木)・10 日(金)

立命館大学 びわこ・くさつキャンパス（滋賀県草津市)

2005 年度（第 6 回)

2005 年 8 月 31 日(水)・9 月 1 日(木)

東京農工大学 小金井キャンパス（東京都小金井市)

2006 年度（第 7 回)

2006 年 9 月 14 日(木)・15 日(金)

龍谷大学 瀬田学舎（滋賀県大津市)

2007 年度（第 8 回)

2007 年 9 月 12 日(水)・13 日(木)

名古屋工業大学（愛知県名古屋市)

Millennium Joint Symposium for Advanced System Software

2000 年 9 月 15 日(金)・16 日(土)

立命館大学 びわこ・くさつキャンパス (滋賀県草津市)

コアステーション 第 3 会議室

プログラム

■ 9 月 15 日(金)

○ セッション 1: 13:00～13:45 総論・研究概要

1. 研究概要 I
   芝 公仁 (立命館大)
2. 研究概要 II
   斎藤 彰一 (和歌山大)
3. 研究概要 III
   毛利 公一 (農工大)

○ セッション 2: 14:00～15:20 マルチメディアと OS

4. 資源管理システム『堤』の PC 上への実装
   田中 大介 (立命館大)
5. 資源管理システム『堤』における QoS 制御機構の設計と実装
   水谷 敬彦 (立命館大)
6. マルチメディアコンテンツ配信専用 OS の設計
   浅見 和男 (農工大)
7. マルチメディアコンテンツ配信のためのリアルタイムスケジューラ
   帆波 幸二 (農工大)

○ セッション 3: 15:40～17:00 分散共有メモリシステム

8. コンパイラによる制御が可能な DSM システム Fagus の実現
   横手 聡 (和歌山大)
9. DSM のための UDP を用いた高速通信ライブラリ Wind の実装
   林 章仁 (和歌山大)
10. 分散共有メモリシステム Lemuria における一貫性制御プロトコル選択機構の構築
    栢 武司 (立命館大)
11. 分散共有メモリシステム Lemuria の広域分散環境への適用
    近藤 輝生 (立命館大)

○ セッション 4: 17:00～17:40 マイクロカーネル

12. マイクロカーネル Lavender における UNIX サーバの構築
    中尾 太一（立命館大）
13. マイクロカーネル Lavender における UNIX サーバの構築
    中里 豪（立命館大）

○18：00～20：00 懇親会（於 ユニオンスクエア 2 階）

■ 9 月 16 日(土)

○ セッション 5: 10:00～11:00 オペレーティングシステムのアーキテクチャと性能評価

14. リアルタイムオペレーティングシステム R2 における仮想記憶の実現
    谷出 新（立命館大）
15. 分散エージェント指向オペレーティングシステムの設計
    瀧本 栄二（立命館大）
16. 性能評価のための命令トレーサの開発
    森本 洋行（農工大）

○ セッション 6: 11:20～13:00 ネットワーク

17. 負荷分散ルータの設計
    笠井 秀一（農工大）
18. ユーザ認証を考慮した音楽配送システム MusicCast/AS に関する一考察
    川北 良一（和歌山大）
19. ホスト監視型によるスキャン防御システムの構築
    伊藤 大輔（和歌山大）
20. 電子メールによるサーバ間協調に基づく分散トランザクションの処理方式
    佐藤 友彦（立命館大）
21. Dynamic License Agreement System for Reuse of Web Contents
    安川 美智子（京都大）

Joint Symposium for Advanced System Software 2001 (JSASS2001)

2001 年 8 月 27 日(月)・28 日(火)

東京農工大学 小金井キャンパス (東京都小金井市)

7 号館 3 階 3K 室

プログラム

■ 8 月 27 日(月)

○ セッション 1: 13:00～14:15 リアルタイム OS

1. 『開聞』における組込み用デバイスドライバフレームワーク
   萱嶋 志門 (農工大)
2. 組込み用 OS 『開聞』のリアルタイム性の考察
   堀口 努 (農工大)
3. リアルタイムオペレーティングシステム Easel の設計と実装
   勝部 弘嗣 (立命館大)

○ セッション 2: 14:15～15:05 リアルタイム Java

4. リアルタイム JavaVM の設計
   駿藤 浩之 (農工大)
5. リアルタイム JavaVM の設計と実装
   奥山 玄 (立命館大)

○ セッション 3: 15:25～16:40 スケジューリング

6. 連続メディア処理のための優先度に基づく入出力制御方式
   帆波 幸二 (農工大)
7. プラグアンドプレイスケジューラの開発
   笠井 秀一 (農工大)
8. VLSI 設計における演算器選択を考慮したスケジューリング
   廣 大輔 (和歌山大)

○ セッション 4: 16:40～17:30 コンパイラ

9. ソフトウェア分散共有メモリシステム Fagus をターゲットとした自動並列化コンパイラ MIRAI の開発 ～Fagus を有効利用する目的コード生成の試作～
   峰尾 昌明 (和歌山大)
10. ソフトウェア分散共有メモリシステム Fagus をターゲットとした自動並列化コンパイラ MIRAI の開発 ～依存解析部の実現～
    北村 隆光 (和歌山大)

○ 懇親会: 18:00～20:00

■ 8 月 28 日(火)

○ セッション 5: 10:00～10:50 分散 OS

11. 分散オペレーティングシステム Solelc における通信機構の構築
    藤本 堅太 (立命館大)
12. 分散オペレーティングシステム Solelc における負荷分散機構
    永宗 宏一 (立命館大)

○ セッション 6: 10:50～11:40 ネットワーク

13. トラブルカルテシステムの構築
    亀谷 信行 (和歌山大)
14. TCP セッション管理による DoS 耐性の考察
    伊藤 大輔 (和歌山大)

Joint Symposium for Advanced System Software 2002 (JSASS2002)

2002 年 9 月 9 日(月)・10 日(火)

和歌山大学 (和歌山県和歌山市)

システム情報学センター

プログラム

■ 9 月 9 日(月)

○ セッション 1: 13:00～14:40 コンパイラと分散並列処理

1. 自動並列化コンパイラ MIRAI の概要
   信原 裕文 (和歌山大)
2. 自動並列化コンパイラ Mirai における配列データ依存解析部の実装
   北村 隆光 (和歌山大)
3. 自動並列化コンパイラ MIRAI におけるコード生成部の実装
   峰尾 昌明 (和歌山大)
4. 分散共有メモリシステムにおける WaveFront 法を用いたタンパク質のホモロジー解析の並列化
   奥野 一矢 (和歌山大)

○ セッション 2: 14:40～15:55 マルチメディア

5. マルチメディア処理に適したリアルタイム IPC の構築
   御田村 晃 (立命館大)
6. リアルタイム環境に適用可能な DMA 転送機構の設計
   勝部 弘嗣 (立命館大)

○ セッション 3: 16:10～17:25 ネットワークとライセンス

7. Web コンテンツ再利用のためのライセンス合意に対する意味論的アプローチ
   安川 美智子 (京都大)
8. インターネット有料放送システムにおける電子透かしの実現
   岩名 雄祐 (和歌山大)
9. ネットワーク型不正コンテンツフィルターの考察
   月城 史行 (和歌山大)

■ 9 月 10 日(火)

○ セッション 4: 10:00～12:05 分散 OS

10. アドホックなデバイスネットワークに向けたソフトウェア基盤に関する検討
    水口 孝夫 (立命館大)

11. オペレーティングシステムを構成するエージェントのモデルに関する考察
    瀧本 栄二（立命館大）
12. 分散 OS Solelc における計算機のグループ化
    丹羽 康裕（立命館大）
13. ユーザ単位で資源管理を行う分散 OS について
    永宗 宏一（立命館大）
14. 複数の計算にまたがる CPU 資源の管理手法
    芝 公仁（龍谷大）

Joint Symposium for Advanced System Software 2003 (JSASS2003)

2003 年 9 月 4 日(木)・5 日(金)

株式会社エスアールアイ（和歌山県西牟婁郡白浜町）

会議室

プログラム

■ 9 月 4 日(木)

○ セッション 1: 13:00～14:15 ネットワークと資源管理

1. 不正コンテンツフィルタの構築
   月城 史行（和歌山大）
2. 分散オペレーティングシステム AG における資源管理手法
   瀧本 栄二（立命館大）
3. マルチスレッドアーキテクチャにおけるシステムソフトウェア
   笹田 耕一（農工大）

○ セッション 2: 14:30～15:45 分散共有メモリとプロセス間通信

4. DIMMnet を用いた分散共有メモリシステム Fagus の設計
   奥野 一矢（和歌山大）
5. IP を用いた多対多通信対応 DIMMnet エミュレータの実装
   平石 高之（和歌山大）
6. 分散オペレーティングシステム AG におけるプロセス間通信手法
   西村 昌俊（立命館大）

○ セッション 3: 16:00～17:15 アプリケーションプラットフォーム

7. コンテキストの変動に適応するアプリケーションプラットフォーム
   永宗 宏一（立命館大）
8. アドホックデバイスネットワークに向けたソフトウェア基盤
   水口 孝夫（立命館大）
9. RT-Java VM における GC パラメータの調整手法
   平田 敬一（立命館大）

■ 9 月 5 日(金)

○ セッション 4: 10:00～10:50 アドホックネットワーク

10. コンテキストを考慮したアドホックネットワークのルーティング方式
    市岡 怜也（立命館大）
11. アドホックデバイスネットワークにおけるデバイスの発見手法
    中沖 治彦（立命館大）

○ セッション 5: 10:50～11:40 コンパイラと並列処理

12. 線形計画法に基づく高速かつ厳密な配列データ依存解析
    峰尾 昌明（和歌山大）
13. 自動並列化コンパイラ MIRAI におけるループ再構築部の設計と実装
    信原 裕文（和歌山大）

○ 昼食休憩: 11:40～13:00

○ セッション 6: 13:00～14:15 データ管理

14. 分散オペレーティングシステム AG における分散ファイルシステム
    藤田 耕作（立命館大）
15. ユビキタス環境におけるトランザクション処理方式に関する検討
    西山 正浩（立命館大）
16. データの多様な制御を実現する手法に関する検討
    鈴来 和久（立命館大）

○ セッション 7: 14:30～16:10 適応的処理

17. ATR 適応研の研究紹介
    滝沢 泰久（ATR）
18. アドホックネットワークにおけるルーティングプロトコルの研究動向
    昌山 一成（ATR）
19. マルチメディアチャットシステムにおけるユーザの関心の類似性に基づいた適応的 QoS 制御
    谷口 典之（ATR）
20. ホットスポットを利用した地域密着型情報ステーション
    益崎 将一（富士通）

Joint Symposium for Advanced System Software 2004 (JSASS2004)

2004 年 9 月 9 日(木)・10 日(金)

立命館大学 びわこ・くさつキャンパス (滋賀県草津市)

エポック立命 21 K309・クリエーションコア CC203

プログラム

■ 9 月 9 日(木) (エポック立命 21 K309)

○ オープニング: 12:20～12:30

○ セッション 1: 12:30～14:00 適応処理

1. プライバシ保護 OS のためのコンテキスト管理手法
   鈴来 和久 (立命館大)
2. 適応的メモリ管理手法に関する考察
   瀧本 栄二 (ATR)
3. 適応的通信デバイス切り替え手法
   元濱 努 (立命館大)

○ セッション 2: 14:10～15:10 ネットワークとシステムソフトウェア

4. ネットワークインタフェース DIMMnet におけるスレッドライブラリ
   森 拓郎 (農工大)
5. 分散共有メモリシステム Fagus における DIMMnet への移植に関連する改善案
   笠松 領介 (和歌山大)

○ セッション 3: 15:20～16:50 言語処理系(1)

6. PC クラスタを対象とする自動並列化コンパイラ MIRAI におけるループ再構築部の発展
   佐川 靖彰 (和歌山大)
7. PC クラスタを対象とする自動並列化コンパイラ MIRAI におけるコード生成部の発展
   善 隆裕 (和歌山大)
8. インタプリタ処理の分割による Java 仮想マシンの高速化
   芝 公仁 (龍谷大)

○ 施設見学（クリエーションコア）：17:10～17:50

○ 懇親会: 18:00～20:00 (C3)

■ 9 月 10 日(金) (クリエーションコア CC203)

○ セッション 4: 10:00～11:00 言語処理系(2)

9. マルチスレッドアーキテクチャにおける言語処理系の検討
   笹田 耕一 (農工大)
10. 並列化可能性判定のための配列データ依存解析問題のモデル化とシンプレックス法を基とする解法の提案
    峰尾 昌明 (和歌山大)

○ セッション 5: 11:15～12:45 ユビキタスコンピューティングとデータ管理

11. ユビキタス環境におけるトランザクション処理方式
    西山 正浩 (立命館大)
12. 携帯電話向け XML ライブラリについての考察
    小高 健二 (農工大)
13. センサネットワークにおけるストリーミングデータベース
    市岡 怜也 (立命館大)

○ クロージング: 12:45～12:55

Joint Symposium for Advanced System Software 2005 (JSASS2005)

2005 年 8 月 31 日(水)・9 月 1 日(木)

東京農工大学 小金井キャンパス（東京都小金井市)

7 号館 3 階 3K 室

プログラム

■ 8 月 31 日(水)

○ オープニング: 13:20～13:30

○ セッション 1: 13:30～14:30 言語処理系

1. オブジェクト指向スクリプト言語 Ruby の処理系の刷新
   笹田 耕一（農工大）
2. Java リアルタイム環境におけるプリエンプション可能な JIT コンパイル処理に関する検討
   西岡 隆司（立命館大）

○ セッション 2: 14:40～16:10 センサネットワーク・モバイル環境

3. センサネットワークにおけるゲートウェイサービスの構成
   市岡 怜也（立命館大）
4. TDoA による位置測位手法 multilateration における問題点と解決手法
   寺嶋 邦浩（立命館大）
5. 携帯電話の Java で稼働する XML データベースコンポーネント『cXMLDB』
   小高 健二（農工大）

○ デモセッション: 16:30～17:30

6. Privacy-Aware OS Salvia におけるコンテキストに適応したファイルアクセス制御
   鈴来 和久（立命館大）
7. 複数のセンサノードを管理するゲートウェイサービスにおけるセンサデータの取得
   市岡 怜也（立命館大）
8. YARV: Yet Another Ruby VM のデモンストレーション
   笹田 耕一（農工大）
9. 携帯電話の Java で稼働する XML 処理コンポーネント
   小高 健二（農工大）
10. SMT プロセッサの FPGA への実装と評価
    小笠原 嘉泰（農工大）

○ 懇親会: 18:00～21:00

■ 9 月 1 日(木)

○ セッション 3: 10:00～10:40 コンピュータと教育

11. 高校生相手にオブジェクト指向プログラミングを教えたら…
    並木 美太郎（農工大）

○ セッション 4: 10:50～12:10 DIMMnet (1)

12. DIMMnet プロジェクトとその後
    中條 拓伯（農工大）
13. DIMMnet-2 におけるプリフェッチ機能による大規模計算の可能性
    羅 徹哲（農工大）
14. DIMMnet-2 における MPI 派生データタイプ通信の構想
    荒木 健志（農工大）

○ 昼食休憩: 12:10～13:00

○ セッション 5: 13:00～14:00 DIMMnet (2)

15. DIMMnet による分散共有メモリシステムの同期変数管理機構の開発と性能評価
    笠松 領介（和歌山大）
16. DIMM スロット搭載型 NIC DIMMnet-2 におけるネットワークドライバ
    森 拓郎（農工大）

○ セッション 6: 14:10～15:10 システムソフトウェア

17. SMT プロセッサにおけるスレッドスケジューラの開発
    内倉 要（農工大）
18. PADOC: Privacy-Aware Data Object Container の着想と考察
    鈴来 和久（立命館大）

○ クロージング: 15:10～15:20

Joint Symposium for Advanced System Software 2006 (JSASS2006)

2006 年 9 月 14 日(木)・15 日(金)

龍谷大学 瀬田学舎（滋賀県大津市)

4 号館 201 室

プログラム

■ 9 月 14 日(木)

○ オープニング: 14:00～14:10

○ セッション 1: 14:10～15:40 システムソフトウェア・システムアーキテクチャ(1)

1. センサノード向けオペレーティングシステムに関するサーベイと考察
   松尾 英治（立命館大）
2. 移動センシング環境を実現するセンサプラットフォーム技術の考察
   松井 雄一（立命館大）
3. HTTP-FUSE-KNOPPIX のモバイル化と高速化・耐故障性の実現
   金井 遵（農工大）

○ セッション 2: 15:55～17:25 システムソフトウェア・システムアーキテクチャ(2)

4. DIMMnet-1 による分散共有メモリシステムにおけるページ転送方式の改良
   中平 勝人（和歌山大）
5. ソフトコア CPU を用いたマルチコアプロセッサシステムの実装と評価
   渡邉 聡（農工大）

○ 懇親会: 18:00～20:00

■ 9 月 15 日(金)

○ デモンストレーション: 10:00～12:00

6. 移動センシング環境におけるセンサノードとマイクロストレージシステムの連携に関するデモンストレーション（立命館大）
7. Privacy-Aware OS Salvia におけるコンテキストに適応したファイルアクセス制御方式に関するデモンストレーション（立命館大）
8. 複数チャネルを用いた無線 LAN の通信品質管理手法に関するデモンストレーション（立命館大）
9. ソフトコア CPU を用いたマルチコアプロセッサシステムの実装と評価（農工大）
10. マルチ SMT アーキテクチャのための要素プロセッサの構成と開発（農工大）

○ 昼食休憩: 12:00～13:20

○ セッション 3: 13:20～14:50 セキュリティ・プライバシ

11. 安全性と設定の簡易化を両立させたファイルアクセスパーミッション
    山口 拓人（岡山大）
12. プライバシ情報の伝播範囲を制御するためのソケット通信制御手法
    西村 和憲（立命館大）
13. 情報漏洩事故の事例を対象とする各種データ保護方式の検証
    鈴来 和久（立命館大）

○ クロージング: 14:50～15:00

編集後記

JSASS には第 2 回目の 2001 年から参加している。毛利先生が東京農工大におられた頃に始まり、7 回参加したことになる。普段は各大学の研究室の中で指導教員および学生同士で議論を行っており、その中で知識だけでなく研究の方法論を学ぶが、ともすると閉鎖的になり、身内だけの議論に陥りやすい。私としては、学生さんは学外やその組織外で通用する研究者・開発者になってほしい、と言う願いもあり、学会発表や他組織との共同研究を通じて、広い視野を身につけてくれればと思っている。しかし、学会は発表を通じて様々な意見をいただける利点もあるが、公の場であること、そして、時間も限られていると言うこともあり、なかなか本音をいただくのが難しい場合もある。本音は懇親会で、と言うことも少なくなかろう。

その点、私的な研究会はざっくばらんな議論ができるので面白い。例えば、PTT と呼ばれるプログラミングに関する私的な研究会をもう 20 年近く参加している。慶応大・電通大・東大・東工大・農工大・早稲田大で持ち回りで月 1 回の研究発表を開催し、ざっくばらんな意見交換を行ってきた。また、近年、プ会なる情報教育の私的な研究会を筑波大・一橋大・農工大の有志数名で立上げ、大学・高校・中学・民間の方々から情報教育の意見交換を行っている他、仮想化技術についても私的な会に参加している。これらの私的な会は、通常の学会と異なり、普段聞けない内容を話してもらえるので、非常に楽しい会となっている。

同様に、毎年 JSASS も楽しみにしている。研究会などでお会いするメンバが研究会では聞けないことをお話いただける他、研究会でお会いできないメンバからもお話を聞けることがとても興味深い。また、学生さんにとっては、各大学での研究内容の他、研究の進め方、何より人脈を広げることができる。実は、若い頃にいろいろな会で会ったメンバは、高い確率でどこかで再会することが多いのである。それは学会であったり、会社であったり、海外であったりするが、「あのときは...」実は大きな財産なのである。実際、まだ 20 代の頃に先の会で知り会ったメンバは今も公私に渡り、どこかで顔をつないでいたりするのである。JSASS もしかり、である。人こそ最大の財産である。

さて、JSASS も 8 回を迎えた。今までは発表するだけで、その内容が残っていなかった。それはそれでいいのだが、やはり、そのときの内容を何らかの形で残し、振り返ることができることは大事であろう。また、毎回ご尽力をいただいている発表者、それからローカルアレンジメントの努力に報いることができないか、と今回は post proceedings を ISSN 番号付の雑誌として企画し、立命館大学のご協力により発刊できた。深く感謝する。冊子体になっていれば、口頭発表として残せるのではなかろうか。そして、今回の各発表資料、それから巻末にある 8 回のリストを見て、「あの頃はこのようなことをやっていた」「あの人はこのようなことをやっていたのか」など、知り合った人々、何より自分自身の足跡を考えることが、発表された方々の将来につながれば幸いである。

東京農工大学　並木 美太郎

JSASS が始まってもう 8 年になった。早いものである。懇親会で大久保先生からご紹介頂いたように、最初は軽い気持ちで、研究交流をしようと話していたところから始まった。とは言え、その背景には、互いに率直な意見をぶつけての議論ができる場が欲しい・必要だという思いや、システムソフトウェア分野が発展するために我々は何をなすべきか、社会へさらに打ち出しをして確固たる地位を築かないといけないといった、ある種の危機感もあったように記憶している。JSASS がこうやって続いているのも、その思いに支えられているからなのだと思う。

毎回、アレンジをして頂いている方々のご尽力によって新鮮味を維持しているところも長続きのコツなのだろうとも感じている。2003 年の白浜で開催された回では、上原先生を中心としてアレンジをして頂いた。開催場所、宿泊、懇親会の随所に工夫を凝らして頂き、さらには白浜町長にもいらして頂いたりと大変有意義だった。2005 年の回では、並木先生にプログラミング教育に関する話をして頂いたり、中條先生に研究プロジェクトの紹介をして頂いたりと、普通の研究発表の形にとらわれない話も聞くことができ、知見を広めることができた。また、JSASS の開催地となったことをきっかけに、それまで JSASS に参加されていなかった方々にも参加して頂いていることも重要な点であろう。今回は、名古屋工業大学の松尾先生、津邑先生、松井先生と、先生方の研究室の学生諸君に参加して頂いた。さらに、卒業生として NTT から一柳さんが参加してくれた。こうやって、新たな刺激を受けることは我々皆の原動力になる。今後とも続けて行きたいところである。

もちろん、継続的に参加し、普段の研究活動の成果を発表してくれる学生諸君の存在は欠かせない。また、表には出てこないものの、開催時期を決めるためにスケジュールを調整したり、発表を取りまとめたりしてくれている、各大学のインタフェース役を担って頂いている方々の協力も欠かせない。この場を借りて、全ての人々に感謝したい。

さて、今回は、並木先生のご提案でポストプロシーディングを発行することとなった。これも JSASS にとっては新たな試みであり、良い刺激になるであろう。ISSN 番号を有する冊子・オンラインジャーナルという形となって残り、さらに国会図書館にも納本されると聞いている。ものづくりを指向する我々であれば、資料としてのものづくりについても、その価値と喜びを共有できるであろう。なお、今回は発表件数が多かったこともあり、全体としては 300 ページを超え、立派なポストプロシーディングを作成することができた。発表者各位にはアブストラクトと電子ファイルの作成でご協力を頂いた。また、各大学・研究室では、資料収集の取りまとめでご協力頂いた。ご協力頂いた方々に再度感謝の意を表したい。

最後に、今後に向けての期待についても記したい。学生諸君の発表は毎回楽しみに聞かせてもらっているが、学生諸君がもっと議論へも参加してくれることを切に期待している。より熱い、意義高い場が得られるであろう。また、卒業後もぜひ顔を出して頂けると幸いである。JSASS が我々皆の将来の基盤となることを願っている。

立命館大学　毛利 公一

先進的基盤ソフトウェア
Joint Symposium for Advanced System Software 2007 (JSASS2007)

1巻1号（通号1号）オンライン版　2007年11月15日発行

編　　集　毛利 公一, 並木 美太郎
委 員 長　大久保 英嗣
発 行 者　JSASS 実行委員会
〒525-8577 滋賀県草津市野路東１－１－１
立命館大学情報理工学部 毛利研究室内
電話 077-561-5061
発 行 所　〒184-8588 東京都小金井市中町２－２４－１６
東京農工大学工学部 並木研究室
電話 042-388-7139