先進的基盤ソフトウェア 3巻1号（通号3号）オンライン版 2009年11月30日発行
ISSN 1882-4196

先進的基盤ソフトウェア

Joint Symposium for Advanced System Software 2009
(JSASS2009)

2009年8月24日(月)・25日(火)

於 岡山大学 津島キャンパス (岡山県岡山市)

JSASS実行委員会

巻頭言

立命館大学　毛利 公一

JSASSは今回で第10回を迎えました. 2000年に第1回が開催されたときは，実は「Millennium Joint Symposium for Advanced System Software」と，名前にMillenniumの冠がついていました．当時は，「世界中が大混乱に陥るに違いない」と言われた2000年問題が騒がれていたことを思い出します．それから10年を経ましたが，この時間はとても重みのあるものだと思います．近頃は，国際会議も多数開催されるようになってきましたが，歴史と権威のあるものを除けば，その開催回数が一桁であることも珍しくはありません．そう思うと，JSASSが刻んだ10年という時間に価値を見いだすことができます．しかも，JSASSでの発表の多くは，情報分野の中核を成す基盤技術に関するテーマであること，システム設計や理論構築だけでなく，実装を伴った工学指向のアプローチを採っていることも特徴的です．さらに，参加者が工学的マインドを持って，互いに切磋琢磨していることも，特筆すべきことと思います．

一方で，周りを見渡してみると，この10年間も情報の研究分野はどんどん広がってきました．特にアプリケーション系の広がりは急速で，それ故に情報分野とは何をする分野なのか，見失われつつあるような気がしています．

一つは，我々研究者・学生とは直接関係しないのですが，「作ること」と「使うこと」の混同が激しいと感じています．例えば，自動車を作ることと運転することは異なることを気づかない人はいないと思います．しかし，情報分野はどうでしょうか．一般の人は，情報分野とは，アプリケーションを使うことだと勘違いしている人が多いとは思いませんか．これほど誤解の多い分野も珍しいのではないでしょうか．

もう一つ，近年の情報系学部・学科の受験生の減少が激しいのもこれと関係しませんでしょうか．高校の教員や受験生の両親が情報分野を理解しておらず進路指導ができないことはないでしょうか．高校の教科情報で，情報Aや情報Cを学んだ人は，情報分野を正しく理解して自分の進路選択に活かせているでしょうか．情報リテラシーやコンピュータリテラシー，すなわち，使うことを情報分野だと勘違いしていないでしょうか．

さらに，大学や企業もどうでしょうか．アプリケーション，すなわちコンピュータのおもしろおかしい使い方ばかりがもてはやされ，そこにある基盤技術や構築技術が軽視される流れが一部にあるようで，近年危機感を感じています．日本の企業では，コンピュータやOSを作らなくなって久しいですし，大学

も間違いなくその流れを進もうとしているように感じます．大学のパンフレットを見ると，見栄えの良いアプリケーションが前面に押し出され，学生を集めようとしている現実がそこにあります．アプリケーションづくりを志す学生ばかりを集めることが，情報分野の未来に本当に寄与するのでしょうか．

情報を専門としていない一般の人々に情報分野について知ってもらい，初等・中等教育においても正しく情報分野を扱ってもらうこと，それも我々専門家の一つの役割ではないでしょうか．これをなくして，将来情報分野を背負って立つ人材は得られません．また，大学や企業も，近視眼的なメリットを追い求めるのではなく，地道であっても，情報分野を醸成する役割を担わないといけないのではないでしょうか．いつまでも，アーキテクチャと OS を一部の外国の企業に握られ，振り回されつづけるのでいいのでしょうか．さらに企業について言えば，ソフトウェアが知識と経験の集大成であるにもかかわらず，それを平気でアウトソーシングしたり，オフショア開発したり，平気で社外に投げるのでいいのでしょうか．今，真剣に考える時期に来ていると思います．

このような中，JSASS では，重要性が変わることがない基盤ソフトウェアを 10 年もの間継続的に取り扱い，そこに基盤技術の研究に従事する先生方や学生が集っていることについてはほっとします．先生方が今後も継続して基盤ソフトウェアの研究をされ，その重要性と魅力を後輩に伝えていただけることを期待しています．また，ここから巣立っていく学生諸君にも，そこで学んだ知識と技術を活かして活躍してほしいと思いますし，アプリケーション作りに飲み込まれていくだけではなく，将来また日本でコンピュータや OS を作り，コンピュータ分野の先進国として名を馳せるような，そういった復活を狙っていってもらいたいと，切に願っています．また，ソフトウェア開発を自社で行えるような体制作りにも取り組んでほしいと考えています．

最後になりましたが，JSASS の開催に幹事としてご尽力いただきました立命館大学の横田裕介先生と龍谷大学の芝公仁先生，ローカルアレンジとしてご尽力いただきました岡山大学の田端利宏先生に，この場をお借りして感謝の気持ちを表したいと思います．また，発表された皆さん，ご質疑ご討論に参加してくださった皆さんにも，お礼申し上げるとともに，今後のご活躍を期待しております．

2009 年 11 月 1 日記

Joint Symposium for Advanced System Software 2009 (JSASS2009)

2009 年 8 月 24 日(月)・25 日(火)

岡山大学 津島キャンパス (岡山県岡山市)

プログラム

■ 8 月 24 日(月)

○ オープニング: 13:20～13:30

○ セッション 1: 13:30～15:00 OS 構成法

○ セッション 2: 15:15～16:45 資源管理

○ セッション 3: 17:00～18:30 セキュリティ(1)

○ 懇親会: 19:00～21:00

■ 8 月 25 日(火)

Cell/B.E.の SPE 向け軽量カーネルの設計と試作

太田 篤志 †

† 東京農工大学大学院工学府

1 はじめに

Cell Broadband Engine（以降「Cell/B.E.」）[1][2]は，汎用的なプロセッサコアである PPE（PowerPC Processor Element）と演算用のプロセッサコアである複数の SPE（Synergistic Processor Element）とで構成されたヘテロジニアスマルチコア CPU であり，SPE を用いた並列演算性能の高さが特徴である．それゆえに SPE をどれだけ効率的に利用できるかがポイントとなるが，従来の SPE の利用方法では，アプリケーションプログラムに課される手間や無駄なオーバーヘッドが存在する．本研究ではこの問題を解決するため，SPE 上で動作する軽量 OS を設計，試作し，アプリケーションに対する SPE 割り当ての効率化，高速化を行う．

2 従来の SPE の利用方法

2.1 spufs+libspe

Cell/B.E.のプラットフォームの一つである家庭用ゲーム機 PLAYSTATION 3 上で動作する Linux には，SPE を仮想化して管理するファイルシステム spufs[3]とそれを利用するためのライブラリ libspe[4]が用意されている（以降この二つを「spufs+libspe」と表記）．この方法ではアプリケーションが必要とする分の SPE コンテキストを管理する．SPE コンテキストと 1 対 1 で対応する論理 SPE は spufs 内でスケジューリングされる．

2.2 MARS

MARS（Multicore Application Runtime System）は 1 個以上の MPU と，それを管理，制御するホストプロセッサで構成されるマルチコアアーキテクチャ上で動作するプログラムの開発のためのライブラリである．MARS は MPU に軽量のカーネルをロードし，実行待ちタスクのチェックやその読み込み，実行，さらにタスクが同期をとる際のコンテキストスイッチ処理などはこのカーネルが担当する．

2.3 従来の利用方法の問題点

spufs+libspe においては，タスクを SPE に割り当てる処理はアプリケーションに任されている．つまり処理を終えた SPE に，次はどのタスクを割り振ればよいか決定する手間をアプリケーションが負わなければならないという問題がある．MARS において，その問題はタスクをキューに保存してスケジューリングすることによって解決しているが，MARS 自体が spufs+libspe を利用して実装しているため，タスクと論理 SPE の二重のスケジューリングになっており，ここに無駄なオーバーヘッドが発生している．

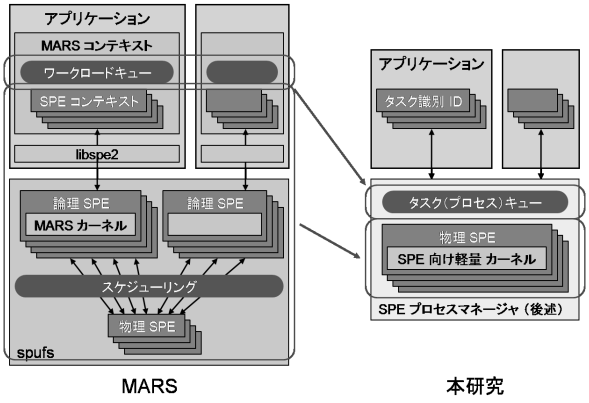


図 1 MARS と本研究の比較

3 目標と方針

以上を踏まえて，本研究ではアプリケーションに対して SPE を効率的に割り当て，割り当てにかかる時間の高速化を目標とする．またその方針を以下に示す．さらに MARS と本研究を比較した図を図 1 に示す．

- アプリケーションごとに管理していた SPE のタスクをまとめ，一本化して管理することにより，SPE をどのアプリケーションにも効率的に割り当てられるようにする．
- SPE に軽量カーネルを実装する．軽量カーネルが実行待ちのタスクのロードと実行を行うことで，SPE は自律的に動作する．
- spufs+libspe を使わず直接ハードウェアを制御し，物理 SPE を直接管理することによって，SPE 割り当てを高速に行えるようにする．

4 設計と実装

SPE で実行するタスクとなる SPE 用プログラムを，以降「SPE プロセス」と呼称する．本研究ではまず PPE 側で動作するプログラムとして，SPE プ

Designing of Lightweight Kernel for SPE of Cell/B.E. and Development of Its Prototype

Atsushi OHTA†

†Graduate School for Engineering, Tokyo University of Agriculture and Technology.

表 1　システムコール一覧

open	SPE プロセスの登録準備
write	SPE プロセスの登録（必要な情報を書き込む）
ioctl	SPE プロセスに対する各種操作 （現在はプロセスの実行開始のみサポート）
read	SPE プロセスの状態の読み出し
close	SPE プロセスの破棄

ロセスに対する操作を受け付け，後述する SPE 向け軽量カーネルに対して実行の指示を行う「SPE プロセスマネージャ」を実装する. SPE プロセスマネージャは Linux のキャラクタデバイスとして実装され，SPE プロセスに対する操作は表 1 のようにデバイスに対して発行するシステムコールにそれぞれ対応している．アプリケーションはこれらシステムコールを介して SPE プロセスを登録し，実行に必要な情報を与え，実行を開始させ，終了を待つことができる．

また SPE 側で動作するプログラムとして，本研究では「SPE 向け軽量カーネル」を実装する．この軽量カーネルは SPE プロセスマネージャが動作を開始した際に SPE にロードされる．その後 SPE プロセスマネージャからの指示によって SPE 用プログラムをロードし，それを実行する．実行が終了すると，SPE 向け軽量カーネルに制御が戻り，SPE プロセスマネージャに対して実行終了を通知する．通知は PPE への割り込みによって実現され，SPE プロセスマネージャ内の割り込みハンドラ内で次に実行するプロセスが再び指示される．

5　評価

SPE プロセスマネージャおよび SPE 向け軽量カーネルの性能を評価するため，何も処理を行わない空のタスクを SPE で動作させたときの実行時間と，ピボットを設定し分割を行うまでの一連の処理を SPE プロセスとしたクイックソートの実行時間を計測し，spufs+libspe および MARS を利用した際の実行時間を比較した．開発・実行環境は PLAYSTATION 3（Cell/B.E. 3.2GHz, 7SPEs），OS は Yellow Dog Linux 6.1（2009/02/01 リリース，カーネルバージョン 2.6.28）である．

空のタスクを SPE で動作させたときの実行時間を図 2 に示す．この評価では，MARS に比べて最大 1.82 倍，spufs+libspe に比べて最大 4.32 倍の高速化を実現することができた．spufs+libspe や MARS よりもオーバーヘッドが小さくなり，SPE 割り当ての効率化，軽量化に一定の成果があるといえる．

次にクイックソートの実行時間を図 3 に示す．この評価では spufs+libspe との比較のみだったが，1,024 要素のソートで 1.80 倍，128 要素のソートでは最大 6.43 倍の高速化を実現することができた．空のタスクだけではなく負荷のかかる一般的なアプリケーションでも本研究によって高速化できることが分かった．

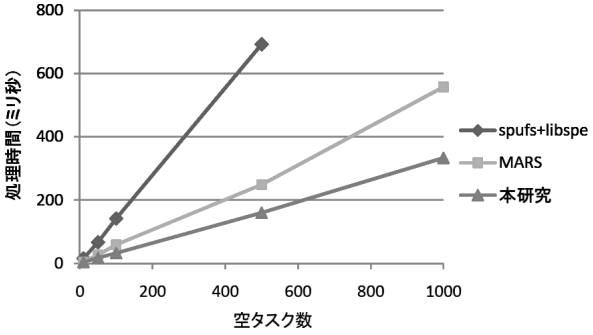


図 2　空タスクの実行時間

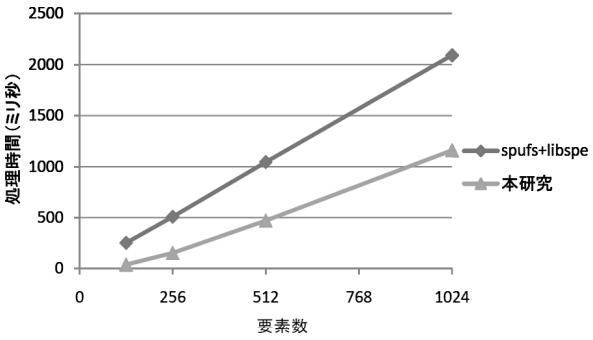


図 3　クイックソートの実行時間

6　おわりに

本研究では SPE プロセスマネージャと SPE 向け軽量カーネルとの連携により SPE 割り当ての効率化を，また物理 SPE を直接制御することにより割り当ての高速化を実現することができた．今後は SPE プロセスに対する操作の追加や，SPE プロセスマネージャ, SPE 向け軽量カーネル双方の機能拡張，また SPE プロセスマネージャの安定性向上などが課題となる．

参考文献

[1] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer and D. Shippy: Introduction to the Cell multiprocessor. *IBM Journal of Research and Development 49 (4)*, pp. 589-604, 2005.

[2] T. Chen, R. Raghavan, J. N. Dale and E. Iwata: Cell Broadband Engine Architecture and its first implementation. *IBM Journal of Research and Development 51 (5)*, pp. 559-572, 2007.

[3] Arnd Bergmann: Spufs: The Cell Synergistic Processing Unit as a virtual file system. *IBM DeveloperWorks*, 2005.

[4] International Business Machines Corporation: SPE Runtime Management Library Version 2.2. *Cell Broadband Engine Architecture Joint Software Reference Environment Series*, 2007.

Cell/B.E. の SPE 向け 軽量カーネルの設計と試作

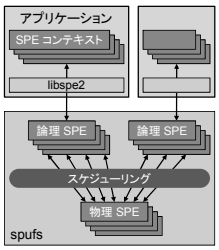
Joint Symposium for Advanced System Software 2009 / No.1
August 24th, 2009 13:30～15:00 : OS 構成法セッション
太田　篤志（東京農工大学 工学府情報工学専攻）

背景

- ヘテロジニアスマルチコアの登場
  - そもそもなぜヘテロジニアス?
    - CPU コア単体で性能を追求するのが難しくなってきた
    - その他, コアの高機能化によるダイ面積増大や発熱問題
    - ある作業に特化したコアを複数種用意することで, コアの構造を簡単にしてこれらの問題を解決する
  - Cell/B.E.（Cell Broadband Engine）
    - OS・制御向けの PPE と計算処理向けの SPE を搭載
      - どちらのコアも単体での性能はあまりよくない
      - いかに SPE を効率的に利用できるかがポイントとなる

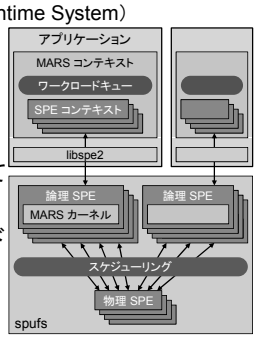
従来の SPE の利用方法（1）

- spufs+libspe2
  （以降は「libspe2」と表記）
  - PS3 Linux 上での開発環境
  - PPE で動作するアプリケーションが必要とする分の SPE のコンテキストを生成
    - コンテキストと論理 SPE は 1 対 1 で対応
    - 論理 SPE のスケジューリング
    - タスクを SPE（コンテキスト）に割り当てる処理はアプリケーションが手動で行う



従来の SPE の利用方法（2）

- MARS（Multicore Application Runtime System）
  - MPU とホストプロセッサからなるアーキテクチャのプログラム用ライブラリ
    - ホスト=PPE, MPU=SPE
  - タスクは SPE に直接割り当てずキューイング
  - MPU に軽量カーネルをロード
    - タスクのロード, 実行などを行う
  - libspe2 を利用して実装



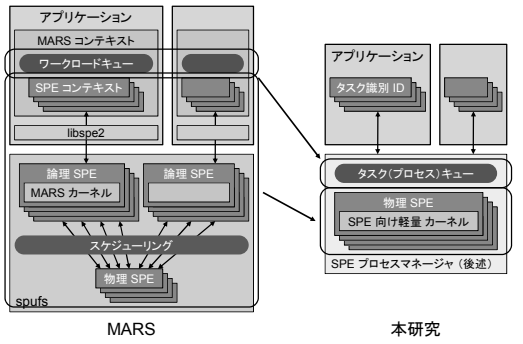
従来の SPE の利用方法（3）

- SPE 割り当て時の問題点
  - libspe2
    - タスクを SPE に割り当てる処理はアプリケーションに任される
      - 次のタスクをどの SPE に割り振ればよいか決定する手間がある
      - タスクの終了タイミングが各 SPE ばらばらの時は特に困る
  - MARS
    - libspe2 の上記の問題をキューイングによって解決
    - MARS と spufs で二重にスケジューリング
      - ワークロードのスケジューリングと論理 SPE のスケジューリング
      - 無駄なオーバーヘッドを出している

目標と方針（1）

- SPE 割り当ての効率化と軽量化
  - アプリケーションごとに管理していた SPE のタスクをまとめ, 一本化して管理
    - SPE を効率的に割り当てられるようにする
    - タスクの操作を行うためのインターフェースを提供
  - SPE に軽量カーネルを実装する
    - 指示されたタスクのロードとその実行を行う
  - 物理 SPE を直接管理する
    - SPE 割り当ての軽量化を図る
    - libspe2 を用いない

目標と方針（2）



設計

SPE プロセス : SPE 用プログラムの実行単位

PPE 側 : SPE プロセスマネージャ
- SPE プロセスの制御（登録, 実行）
- SPE 軽量カーネルとの通信

SPE 側 : SPE 軽量カーネル
- SPE プロセスマネージャとの通信
  - 受信 : 次のプロセスのコンテキスト情報
  - 送信 : 実行終了の通知
- プロセスコンテキストのロード

アプリケーションとのインターフェース

システムコール	説明
open	SPE プロセスの登録準備
write	SPE プロセス登録 （必要な情報を書き込む）
ioctl	プロセス実行開始・中断・再開・強制終了 （現在は実行開始のみサポート）
read	SPE プロセスの状態の読み出し （実行状況, 完了している場合は戻り値）
close	プロセスの破棄をマネージャに許可

プログラム例

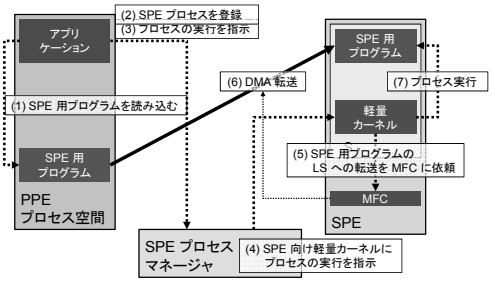
PPE 側

```
int main(int argc, char *argv[])
{
    char spe_program[] = { ... };
    int fd;
    spe_process_info process_info;
    int data[] = { ... };
    fd = open("/dev/spe_mng", O_RDWR);
    process_info.program_start = spe_program;
    process_info.program_size = ...;
    process_info.arg = data;

    write(fd, process_info, sizeof(process_info));
    ioctl(fd, SPE_PROCESS_START, 0);
    do {
        read(fd, &process_info, sizeof(process_info));
    } while (process_info.status != PROCESS_END);
}
```

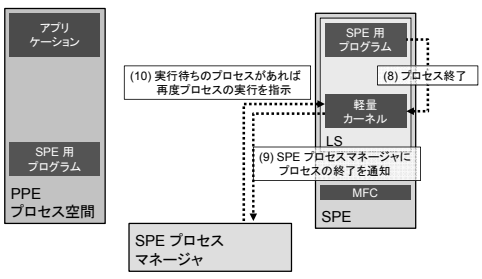
SPE 側

```
int main(...)
{
    int data[128 * 1024] = { 0 };
    dma_get(data, addr, size);
    /* 何か処理する */
    dma_put(data, addr, size);
    return 0;
}
```

SPE プロセスの実行手順（1）



SPE プロセスの実行手順（2）



評価（1）

- 開発・実行環境
  - PLAYSTATION 3
    - Cell/B.E. 3.2GHz, 7 SPEs
      - Linux で使用できる SPE は最大 6 個
  - Yellow Dog Linux 6.1（2009/2/1, カーネル 2.6.26）
- 評価内容
  - SPE での空タスクの実行時間
    - タスク生成から終了までを PPE アプリケーション側で計測
    - libspe2, MARS で行った場合と比較

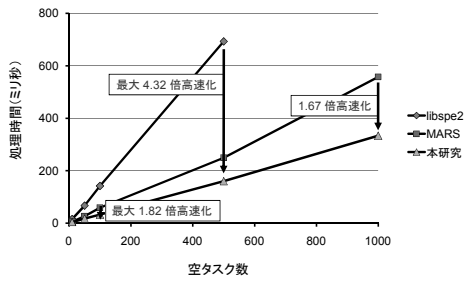
```
int main(...)
{
    return 0;
}
```

評価（2）

- 評価内容
  - クイックソートの実行時間（128～1024要素）
    - ピボット設定→分割 を 1 つの SPE プロセスで実行
    - 分割されたデータは DMA でいったん書き出し，各々また新しい SPE プロセスでピボット→分割処理
    - プログラム自体は DMA ダブルバッファリングや -O3, SIMD による最適化は行っていない
    - libspe2 で行った場合と比較
      - どちらもまったく同じ乱数列をソート

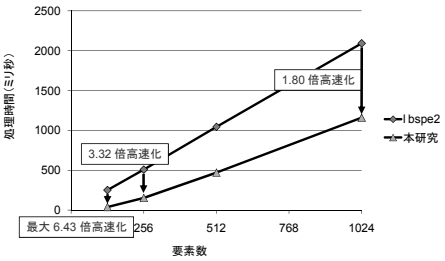
評価（3）

- SPE での空タスクの実行時間



評価（4）

- クイックソートの実行時間



考察

- SPE での空タスクの実行時間
  - MARS との比較では最大 1.82 倍，libspe2 との比較では最大 4.32 倍高速化
  - libspe2 や MARS に比べオーバーヘッドが小さく割り当ての効率化・軽量化に一定の成果
- クイックソートの実行時間
  - libspe2 と比較して 1,024 要素では 1.80 倍，128 要素では最大 6.43 倍高速化
    - 細粒度のタスクには libspe2 に比べて特に有効
    - 粒度が粗くなっても本研究の方が高速

まとめ（1）

- SPE の割り当ての効率化・軽量化を実現
  - 効率化
    - SPE プロセスマネージャと SPE 向け軽量カーネルの連携
      - SPE プロセス管理の一本化
      - プロセス操作のためのインターフェース
      - SPE 割り当て方法の改良
  - 軽量化
    - spufs+libspe2 を使わず物理 SPE を直接制御することで SPE を高速に利用することが可能

まとめ (2)

- SPE の割り当ての効率化・軽量化を実現
  - 空タスクの実行速度
    - MARS に比べて 1.82 倍,
      libspe2 に比べて 4.32 倍高速化
    - 実行オーバーヘッドは二者に比べて非常に小さい
  - クイックソートの実行速度
    - libspe2 に比べて最大 6.43 倍高速化
    - 細粒度のタスクへの有効性を確認

今後の課題

- プロセスマネージャ, 軽量カーネルの機能拡張
  - SPE プロセスに対するアクションの追加
    - 中断, 再開, 終了など
  - 同期, プロセス間通信, 明示的な制御移譲
  - SPE が備える mailbox, シグナル機構への対応
    - 現時点ではプロセスマネージャ側が未対応
  - プロセスマネージャに対するシステムコールの追加
    - 現在の実装ではまだ使いにくい
- そのほか
  - プロセスマネージャの安定性向上

DAVfs と Aufs を用いた組込み Linux による シンクライアントシステムの試作

竹川知孝†

東京農工大学大学院工学府情報工学専攻†

1. はじめに

近年，ソフトウェア，ハードウェアの両面で組込み機器の高機能化が進んでいる．ソフトウェアにおいて，以前は組込み機器に OS を搭載する事は少なかった．しかし，最近は組込み機器で使用する OS として Linux を採用するケースが増加している．この結果，組込み機器のアプリケーションが直接，ハードウェアを制御する必要がなくなり，Linux 用に開発された豊富なミドルウェアも利用できるようになった．

ハードウェアの高機能化の例として，ネットワーク機能が挙げられる．組込み機器をネットワークで繋いだセンサーネットワークやホームネットワークが普及しユーザの利便性は増した．しかし，ネットワークで機器を繋ぐ事でクライアントが使用するユーザデータや管理情報が分散し，クライアントの管理が難しくなっている．そこで本研究では組込み機器を対象にした Linux シンクライアントシステムの作成を試みる．

2. 問題分析

クライアントが使用するカーネル，initrd の配置場所と実行場所によって既存のシンクライアントシステムを分類し，それぞれの問題点について述べる．

● 起動前にクライアントに配置

クライアントの中に ROM を用意し，ROM にクライアントが使用するカーネル，initrd を保存する．そのため大量の ROM が必要になり，これらのファイルの更新をする場合は，クライアント個別に更新作業を施さなければならない．

● 画面転送方式

サーバで処理を実行し，クライアントには入出力機器のみを配置する．このため入出力機器のあるクライアントしか使用できない．組込み機器のように必要最低限のハードウェアしか搭載しない機器では入出力機器が接続されるとは限らないため，組込み機器を対象としたシンクライアントシステムには適さない．

● ネットワークブート方式

クライアントが電源投入後にカーネルと initrd を取得してブートする．しかし，使用の有無に関わらず全てのファイルを取得する事は，機器のメモリを圧迫している状況と言える．

3. 目的

本研究はネットワークブート方式のシンクライアントシステムにおいて，クライアントに要求されるメモリ（ROM）の削減とクライアントの管理を容易にする事を目的とする．

クライアントに要求されるメモリを削減するために本システムはクライアントが使用するファイルをサーバとクライアントに分散配置する．この時，クライアントに配置するファイルを最小限にすることでクライアントに要求されるメモリの削減を目指す．また，クライアントの管理を容易にするため，クライアントが使用するファイルはサーバで一元管理をする．

4. 設計

ネットワークブート方式のクライアントは通常ストレージを搭載していない．クライアントはサーバにデータを保存するため，ネットワークファイルシステムを利用する事になる．既存のシンクライアントシステムは Unix 系であれば NFS,Windows 系であれば SMB を利用する事が多かったが，本研究では WebDAV と Aufs を組み合わせて使用する．WebDAV を利用する事で 80 番ポートのみを用いた通信が可能になり，容易にファイアウォール介した通信ができる．また，Aufs を用いる事で二つのディレクトリを統合できる．クライアントはサーバとクライアントに分散したファイルを統合できるので，これらをシームレスに扱えるようになる．WebDAV と Aufs を組み合わせる事で必要最小限のファイルをクライアントに配置し，目的に応じた機能を Web サーバから取得するシンクライアントシステムが構築

Prototyping of a Thin Client System Using the DAVfs and Aufs on Embedded Linux

† Tomotaka TAKEKAWA

† Department of Computer and Information Sciences, Tokyo University of Agriculture and Technology

できる.

図 1に構築したシンクライアントシステムの全体図を示す.

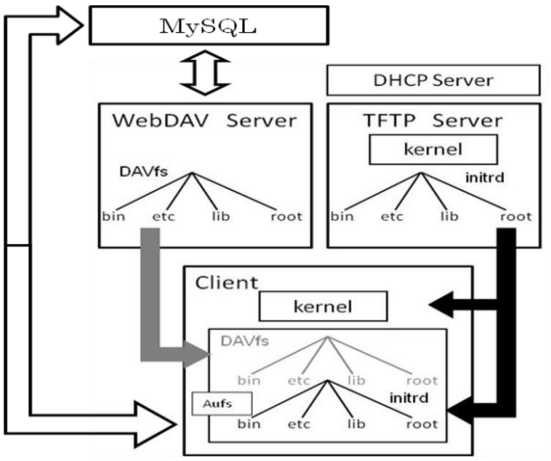


図 1　システム全体図

本システムは大きく分けて４つのパートで構成されている．それぞれの名称と機能を以下に示す.

● WebDAV サーバ

DAVfs (WebDAV で接続されるファイルシステム) をクライアントに提供する

● TFTP サーバ

クライアントにカーネルと initrd を提供する

● データベースサーバ

クライアントが使用する DAVfs を管理するデータベースである

● クライアント

DAVfs と initrd を Aufs で統合して利用する

本システムのクライアントはブート時に DAVfs と initrd を Aufs で統合しルートファイルシステムとして使用する．以下にクライアントのブートプロセスの詳細を示す.

① 電源投入後，クライアントは TFTP サーバからカーネルと，WebDAV サーバに接続するためのファイルが入った initrd を取得する.

② initrd のファイルを用いて WebDAV サーバに接続し，SQL_DAVfs(データベースサーバに接続するためのファイルが入った DAVfs)を取得する．その後，SQL_DAVfs のファイルを用いてデータベースサーバに接続する.

③ データベースでクライアントが取得する User_DAVfs(クライアント個別の設定ファイルやユーザデータを含む DAVfs)を確認する．確認後，SQL_DAVfs を解放して，確認した User_DAVfs を WebDAV サーバから取得する.

④ クライアントのローカルにある initrd と取得した User_DAVfs を Aufs で統合しクライアントのブートが完了する.

クライアントの IP アドレスなどを指定する設定ファイルを User_DAVfs に置く事で，設定ファイルは常に WebDAV サーバの管理下に置かれる．結果，本システムによってクライアントの管理が容易になる.

5. 実装と評価

本システムはサーバとクライアントを PC と玄箱 PRO でそれぞれ実装し，4 通りのシステムを構築する事ができる．また，User_DAVfs に配置したプログラムを実行し，クライアントに接続したセンサーキットの値をクライアントで表示する事も可能である.

本システムを評価するため PC と玄箱をクライアントにした時、それぞれの機器が使用する非圧縮時の initrd のサイズを計測した．表 1に計測結果を示す.

表 1　initrd のサイズ

PC の initrd(KB)	玄箱の initrd(KB)	Fedora の initrd(KB)
9204	6038	4877

玄箱の initrd に WebDAV サーバに接続するためのファイルだけを配置する事でサイズを約 6MB まで削減できた．これは PC で起動する Linux のルートファイルシステムと比較して大幅に削減している．また，PC の initrd は Fedora の initrd を基盤に作成したためサイズが大きくなっている．それぞれの initrd はライブラリファイルが約 5MB を占めている.

6. おわりに

本研究では，組込み機器を対象にした Linux シンクライアントシステムの作成を試みた．結果，クライアントに必要な initrd を 6MB まで削減し，WebDAV と Aufs を用いた組込み機器による Linux シンクライアントシステムを構築した．また，クライアントが使用するユーザデータ，設定ファイルを WebDAV サーバで一元管理しクライアントの管理を容易にした.

参考文献

1) 高橋竜男 , 高橋修 , 水野忠則 : モバイル向けシンクライアントシステムの検討 , 情報処理学会論文誌 Vol.45,No.5, pp.1417-1431(2004).

2) WebDAV Linux File System (davfs2). http://dav.sourceforge.net/.

DAVfsとAufsを用いた組込みLinuxによるシンクライアントシステムの試作

JSASS
2009/08/24
東京農工大学　工学府　情報工学専攻
竹川知孝

背景

- Linuxベースの組込み機器の増加
  - 携帯電話，情報家電においてOSとしてLinuxを採用するケースが増加
- ネットワーク機能を搭載した組込み機器の普及
  - 組込み機器のネットーワーク（センサーネットワーク，ホームネットワーク）が普及し、ユーザデータや管理情報が分散

Linuxベースの組込み機器向け
シンクライアントシステム

クライアントが使用するファイル

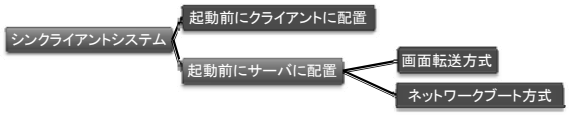
- Linuxカーネル
- initrd・・・・・・・・・・Linuxが使用するファイルの集合
  - 実行ファイル・・・・・・・ユーザが実行できるコマンド
  - ライブラリファイル・・・共有ライブラリ
  - 設定ファイル・・・・・・・IPアドレスの指定など
  - ユーザデータ・・・センサーの計測結果などのユーザファイル

カーネル，ユーザデータ以外のファイルをシステムファイルとする

これらのファイルの配置場所、実行場所によって
シンクライアントシステムを分類できる

既存のシンクライアントシステムの分類

クライアントが使用するカーネルとinitrdの配置場所、実行場所による分類



- 起動前にクライアントに配置
  - クライアントに保存するためのROMが必要になる
  - カーネルとinitrdの更新はクライアント個別に対応しなければならない
- 画面転送方式
  - 入出力機器のあるクライアントしか使用できない
  - ex. Citrix Presentation Server, VNC

ネットワークブート方式の問題点

- 組込み機器の資源
  - ネットワークを介してカーネル、initrdを取得し、クライアントが起動
    ⇒メモリを圧迫している
- 管理の手間
  - クライアントが設定ファイルを取得
    ⇒クライアントごとに設定ファイルを用意しなければならない

目的

クライアントに要求されるメモリ（ROM）の削減

- クライアントに配置するファイルとサーバに配置するファイルに分ける
- クライアントに配置するファイルは最小限にする

クライアントの管理を容易にする

- 電源投入前のクライアントには何もファイルを置かない
- クライアントが使用するファイルはサーバで一元管理する

取り組むべき課題

- クライアントに配置すべき機能の選定
  - メモリを削減するためにはどんな機能をクライアントに残し、どんな機能を削減するか
- どうやってクライアントの管理を容易にするか
  - クライアントの管理を容易にする具体的な方法を検討しなければならない

サーバとクライアントに分散したファイルをシームレスに扱い、容易にネットワークの構築が出来るようにする

7

ネットワークファイルシステム

- サーバに配置したシステムファイル、ユーザデータをクライアントが使用するために、ネットワークファイルシステムを利用する
  ex. NFS, SMB

WebDAVとAufsを使用

- クライアントのルートファイルシステムはinitrdだけでなく、WebDAVとAufsを組み合わせて使用する

8

WebDAVとAufsの特徴

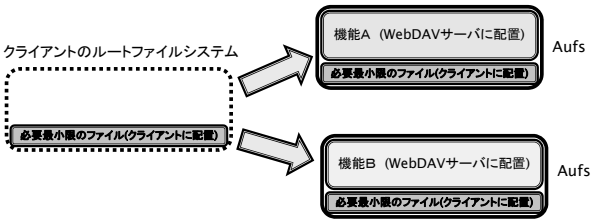
- WebDAV
  - httpを拡張し，ダウンロードだけでなくアップロードにも対応したプロトコル
  - SSLの利用などWebサーバのセキュリティを確保・向上させるときの考え方が流用可能
  - httpを用いた通信であるため、WANにおいて容易にファイアウォールを介した通信ができる

  ⇒面倒な設定をしなくても接続できるので管理が容易になる
- Aufs
  - オーバーライドによって二つのディレクトリを統合する
  - クライアントのディレクトリとサーバのディレクトリを統合できるので、クライアントに配置するファイルを最小限にできる

  ⇒クライアントに必要なメモリを削減できる

9

WebDAVとAufsを組み合わせる利点

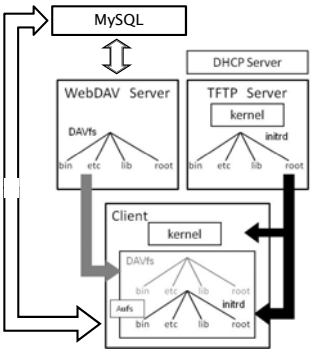


- 必要最小限のファイルをクライアントに配置
- 目的の機能に応じたファイルをWebサーバから取得してクライアントが使用する

10

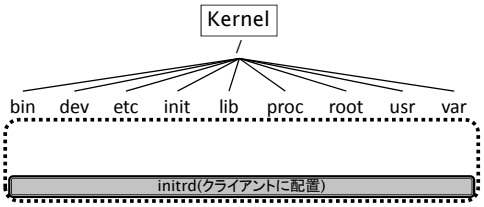
システム全体図

- WebDAVサーバ
  - DAVfs(WebDAVで接続されるファイルシステム)を提供
- TFTPサーバ
  - カーネルとinitrdを提供
- MySQL
  - クライアントが使用するDAVfsを管理するデータベース
- クライアント
  - ブート時にDAVfsとinitrdをAufsで統合し、ルートファイルシステムとして使用



11

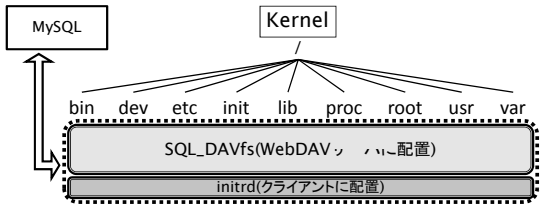
クライアントのブートプロセス



- 起動前にクライアントには何も配置しない
- TFTPサーバからカーネルを取得
- WebDAVサーバに接続するためのシステムファイルを含むinitrdをTFTPサーバから取得しクライアントに配置

12

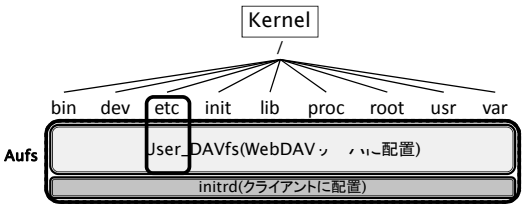
クライアントのブートプロセス



- initrdのファイルを用いてSQL_DAVfsを取得
- データベースに接続してクライアントが使用するUser_DAVfsを確認する
- 確認後、SQL_DAVfsを解放

13

クライアントのブートプロセス



- 確認したUser_DAVfsを取得
- Aufsを用いて、クライアントのローカルに置いたinitrdと統合する
- /etcの設定ファイルをDAVfsに置き、クライアントの管理を容易にする

14

initrdとSQL_DAVfs

	initrd	SQL_DAVfs
bin	**mount.davfs** busybox	mysql
dev	デバイスノード	×
etc	mtab init.d/rcS	×
lib	mount.davfsに必要なライブラリ	mysqlに必要なライブラリ libmysqlclient.so.15
mnt	server/bin,etc,lib,root	×
proc	procファイルシステムをマウント	×
root	何も保存しない	×
sbin, usr	busyboxへのシンボリックリンク	/usr/libにmysqlに必要なライブラリを保存
var	cache, run	×

（×：WebDAVサーバで提供されない）

- initrdは mount.davfsの実行に必要なシステムファイルから構成される
- SQL_DAVfsにはデータベースサーバに接続するためのファイルが格納されている

15

initrdとUser_DAVfs

	initrd	User_DAVfs
bin	**mount.davfs** busybox	クライアント個別の実行ファイル InterfaceKit-simple
dev	デバイスノード	×
etc	mtab init.d/rcS	クライアント個別の設定ファイル sysconfig/network
lib	mount.davfsに必要なライブラリ	クライアント個別のライブラリ libphidget21.so
mnt	server/bin,etc,lib,root	×
proc	procファイルシステムをマウント	×
root	何も保存しない	ユーザデータ
sbin, usr	busyboxへのシンボリックリンク	×
var	cache, run	×

（×：WebDAVサーバで提供されない）

- クライアントごとのシステムファイル、ユーザデータ はUser_DAVfsで提供されAufsによってオーバーライドされる

16

実現

- 本シンクライアントシステムは以下の機器を使用した
  - PC(サーバ)
    - Celeron 2.5GHz,メモリ512MB,1GbpsEthernet
    - Fedora6でApache2.2, DHCP Server V3.0.5, tftp0.42を起動
  - PC(クライアント)
    - Core2Duo1.66GHz,メモリ1GB,1GbpsEthernet
    - PXEによるネットワークブートが可能
  - 玄箱PRO
    - ARM9互換Marvell製88F5182 400MHz,メモリ128MB, 1GbpsEthernet
    - サーバはdebian4.0でApache2.2, DHCP Server2.0p15 tftp0.17を起動
    - クライアントはローカルのブートローダによるネットワークブート

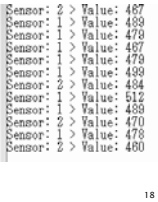
17

構築したシンクライアントシステムの機能

- 現在は4通りの組み合わせでシンクライアントシステムの構築が可能
- Webサーバに配置したアプリケーションを実行し、クライアントに接続したセンサーキットの値をクライアントで表示できる

シンクライアントシステムの組み合わせ

サーバ	クライアント
PC	PC
	玄箱
玄箱	PC
	玄箱



18

評価方法

- システムの性能を評価するため以下の項目について計測した
  - PCと玄箱をクライアントにした時、それぞれの機器が使用するinitrdのサイズ
    - 非圧縮時のinitrdのサイズを計測
  - サーバとクライアントをPCと玄箱にした4通りの組み合わせでWebDAVとNFSのファイル転送速度
    - 「time dd if=/root/testfile of=/result」によってユーザデータを転送した場合の転送時間から転送速度を算出

19

評価結果

- initrdのサイズの比較

PCのinitrd(KB)	玄箱のinitrd(KB)	Fedoraのinitrd(KB)
9204	6038	4877

- 玄箱が使用するinitrdを約6MBまで小さくした
- PCのinitrdはFedoraのinitrdを基盤に作成したので大きくなった
- それぞれのinitrdの内/libが約5MBを占めている

20

評価結果

- 転送速度の比較

サーバ	クライアント	WebDAVの転送速度(MB/秒)	NFSの転送速度(MB/秒)
PC	PC	17.9	12.5
	玄箱	2.98	3.25
玄箱	PC	13.1	8.89
	玄箱	3.72	2.85

- PC(サーバ)－PC(クライアント)で43%
  玄箱(サーバ)－PC(クライアント)で47%
  玄箱(サーバ)－玄箱(クライアント)で30%
  NFSよりWebDAVの方が速かった

21

まとめ

- 組込み機器でのシンクライアントシステムを実現
  - クライアントに必要なファイルを取捨選択する事で、クライアントに必要なメモリを削減
  - ⇒組込み機器でもシンクライアントシステムに加える事が可能
- クライアントの管理を容易にした
  - クライアントが利用するファイルは、クライアント共通のinitrdと、個別のDAVfsに分けて提供
  - クライアントが使用するUser_DAVfsをデータベースで管理
  - ⇒クライアントが使用するファイルをサーバで一元管理し、クライアントにはサーバ接続に必要な共通のファイルのみ配置

22

今後の目標

- 他の組込み機器への移植
  - 玄箱以外の機器も本システムに加えられるようにする
- データベースサーバとの連帯強化
  - ディレクトリまたはファイルの単位でUser_davfsが管理できるようにする
- 応用システムの作成
  - センサーネットワーク

23

ご清聴ありがとうございました

24

AnT における OS サーバ入れ替え機能の設計と基本評価

藤原 康行　田端 利宏　乃村 能成　谷口 秀夫
岡山大学大学院自然科学研究科

1　はじめに

高い適応性と堅牢性の実現を目指し，OS 機能の大半を OS サーバとして実現するマイクロカーネル OS である ***AnT*** オペレーティングシステム [1] において，OS サーバ入れ替え機能を設計した．ここでは，本機能の設計と基本評価を述べる．

2　*AnT* オペレーティングシステム

AnT オペレーティングシステムは，マイクロカーネル構造を持ち，OS 機能の大半を OS サーバとして実現している．

AnT の OS サーバ間通信 [2] は，コア間通信データ域（ICA）[3] を利用する．ICA は，仮想空間のマッピング表の書き換えによりプロセス間でのゼロコピー通信を可能にしている．OS サーバ間通信は，OS サーバへ渡す引数や通信制御の情報と OS サーバで扱うデータを別々の ICA（制御用 ICA/ データ用 ICA）に格納することで，高速な通信を可能にしている．

3　OS サーバ入れ替え機能の設計 [4]

OS サーバ入れ替えにおいては，OS サーバが提供するサービスを利用する応用プログラム（AP）への影響を最小限にすることが非常に重要である．このため，OS サーバ入れ替え機能の設計方針として，以下を定める．

（**方針 1**）OS サーバが保有する情報の最小化

（**方針 2**）新旧 OS サーバ間における処理の継続

（**方針 1**）への対処として，OS サーバ処理に原子性を持たせる（OS サーバ設計規約）．（**方針 2**）への対処として，新旧 OS サーバ間での情報の引き継ぎを行う．引き継ぐ情報には，依頼キューや結果キューに繋がれている情報（要求情報）と OS サーバの内部状態に関する情報（状態情報）がある．

4　評価

OS サーバ入れ替え機能を実装した ***AnT*** オペレーティングシステムを Intel® Celeron（2.0GHz）プロセッサの計算機上で走行させ，時間を測定した．なお，OS サーバを入れ替えない場合の応答時間（AP プロセスの OS サーバに対する処理依頼発行から結果返却が得られるまでの時間）は 250 ミリ秒程度であった．

（**1**）入れ替え時間は最大 110 ミリ秒程度

（**2**）応答時間への影響は最大 20 ミリ秒程度

入れ替え時間の大半は，新 OS サーバのプロセス生成時間であった．また，OS サーバやドライバの CPU 処理中に OS サーバ入れ替え機能が呼び出された場合，入れ替え時間が増加している．これは，OS サーバやドライバの CPU 処理完了まで入れ替えが待たされるためである．

多くの場合，応答時間は 250 ミリ秒程度であり，OS サーバを入れ替えない場合の応答時間と同等である．これは，AP プロセスやドライバの Sleep 処理中に入れ替えを行うためである．また，AP プロセスからの処理依頼時，またはドライバからの結果返却時に入れ替えを行っている場合，応答時間が増加している．これは，入れ替え完了まで新 OS サーバの CPU 処理が待たされるためである．

5　まとめ

AnT オペレーティングシステムにおける OS サーバ入れ替え機能の設計について述べた．また，基本評価として，OS サーバ入れ替え時間は最大 110 ミリ秒程度であり，サービスを利用する応用プログラムに与える影響は最大 20 ミリ秒程度と小さいことを示した．

参考文献

[1] 谷口秀夫, 乃村能成, 田端利宏, 安達俊光, 野村裕佑, 梅本昌典, 仁科匡人: “適応性と堅牢性をあわせ持つ ***AnT*** オペレーティングシステム,” 情報処理学会研究会報告, 2006-OS-103, Vol.2006, No.86, pp.71-78, 2006.

[2] 岡本幸大, 谷口秀夫: “***AnT*** におけるサーバ間の高速なプログラム間通信機構,” マルチメディア通信と分散処理ワークショップ論文集, Vol.2007, No.9, pp.61-66, 2007.

[3] 梅本昌典, 田端利宏, 乃村能成, 谷口秀夫: “***AnT*** オペレーティングシステムにおけるメモリ領域管理の設計と実現,” 情報処理学会研究報告, 2007-OS-104, Vol.2007, No.10, pp.33-40, 2007.

[4] 藤原康行, 岡本幸大, 田端利宏, 乃村能成, 谷口秀夫: “***AnT*** における OS サーバ入れ替え機能,” マルチメディア通信と分散処理ワークショップ論文集, Vol.2008, No.14, pp.201-206, 2008.

Design and Basic Evaluation of Dynamic OS Server Replacement Mechanism for *AnT*

Yasuyuki FUJIWARA, Toshihiro TABATA, Yoshinari NOMURA and Hideo TANIGUCHI
Graduate School of Natural Science and Technology, Okayama University

AnT におけるOSサーバ 入れ替え機能の設計と基本評価

藤原 康行　田端 利宏
乃村 能成　谷口 秀夫

岡山大学大学院自然科学研究科

研究背景

(1) 計算機の提供するサービスの高度化
　ー　24時間365日稼動システムの普及
(2) 基盤ソフトウェア(OS)に対する高い適応性と堅牢性の要求
　ー　機能拡張や不具合発生によるサービスの停止を抑制

＜マイクロカーネル構造＞
(1) 必要最低限のOS機能をカーネルとして実現
(2) その他のOS機能をプロセス(OSサーバ)として実現

OSの機能拡張時や不具合発生時

⇒ 対象OSサーバを新たなOSサーバに入れ替え

⇓

システムを停止させることなくサービスの継続が可能

No.2

AnT オペレーティングシステム

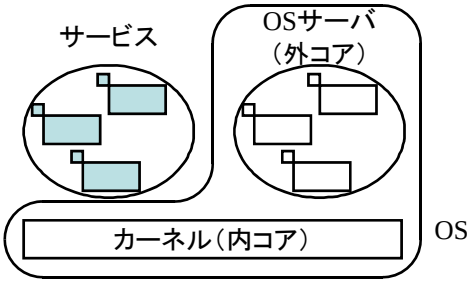
（An operating system with Adaptability and Toughness）

＜マイクロカーネル構造＞

（1）必要最低限の機能のみをカーネル（内コア）として実現
　　－　実行権管理，領域管理，割り込み管理等

（2）その他の機能をOSサーバ（外コア）として実現
　　－　ファイル管理，ネットワーク管理，デバイスドライバ等

（3）種々のサービスをプロセスとして実現
　　－　コマンドインタプリタ，各種コマンド等



No.3

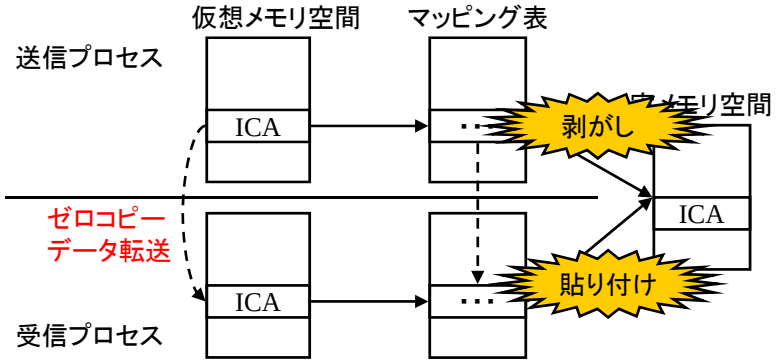
コア間通信データ域

（Inter-core Communication Area）

＜ゼロコピーデータ転送＞

（1）コア間通信データ域（ICA）と呼ぶ領域を仮想空間のメモリマッピングの書き換えによりゼロコピーでデータ授受

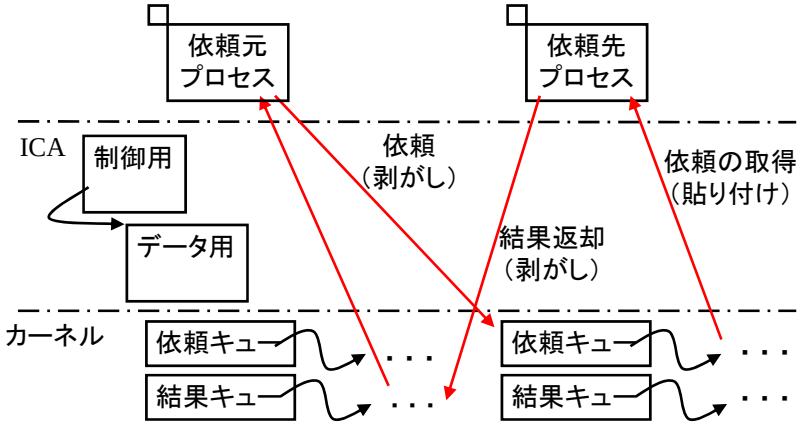
（2）ページ単位による領域の貼り付けと剥がし（貼り替え）



No.4

サーバ間通信制御機構

＜高速なサーバ間通信＞
(1) 依頼元プロセスは依頼先プロセスの依頼キューに依頼を登録
(2) 依頼先プロセスは登録された情報を取得し処理を実行
(3) 依頼先プロセスは依頼元プロセスの結果キューに結果を登録



No.5

OSサーバ入れ替えの設計方針

(方針1) OSサーバが保有する情報の最小化
　(目的1) OSサーバ不具合発生時の影響を最小化
　(目的2) OSサーバ入れ替え処理の時間を短縮

⇒ OSサーバ処理に原子性を持たせる

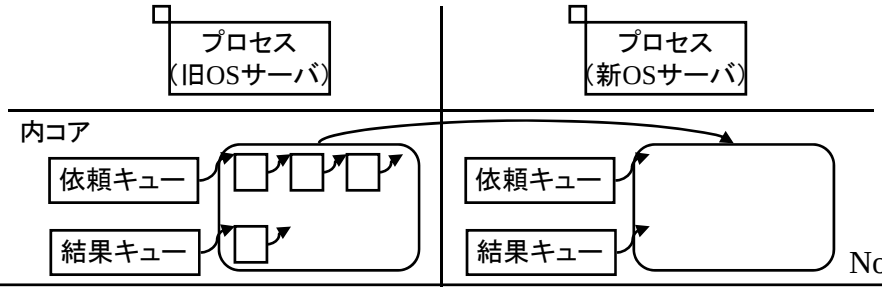
(方針2) 新旧OSサーバ間における処理の継続
　(目的) OSサーバ入れ替えによるAPへの影響を最小化

⇒ 新旧OSサーバ間における情報の引き継ぎが必要
(1) 依頼や結果に関する情報(要求情報)
(2) OSサーバ内部状態に関する情報(状態情報)

No.6

要求情報の扱い

(1) OSサーバ入れ替えは不可分操作非処理時に実行
　(機能拡張時)　不可分操作が終了するまで待機
　(不具合発生時) 現在処理中の要求をエラーとし，不可分操作を終了
(2) 要求情報はICAの貼り替えとキューの繋ぎ替えにより移行
　(A) 保有するすべてのICAの貼り替え
　(B) 依頼キューと結果キューの繋ぎ替え



No.7

状態情報の扱い

(例1) 各APが利用しているファイル情報(ファイル管理)
(例2) 現在キャッシュ中のブロック情報(ブロック管理)

(1) 状態情報の保存
　ー　各OSサーバ毎に状態情報保存用のICA(状態情報ICA)を設け，状態情報を保存

⇒ 要求情報の扱いと同様にICAの貼り替えを利用可能

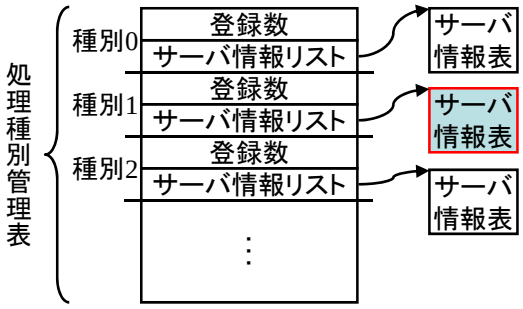
(2) 状態情報の移行
　ー　OSサーバ入れ替え時に状態情報ICAのアドレスを通知

⇒ ゼロコピーによる状態情報の移行が可能

No.8

登録情報の移行

＜登録情報の管理構造＞
（1）OSサーバの登録情報はサーバ情報表に格納
（2）OSサーバ1つに対してサーバ情報表1つを割り当て
（3）他のプログラムからOSサーバへの通信は，処理種別に対応するサーバ情報表から対象のOSサーバを特定して実行



旧OSサーバのサーバ情報表を再利用

OSサーバ入れ替え途中に発行される要求に対応

No.9

評価

＜評価観点＞
（1）OSサーバ入れ替えにかかる時間
（2）OSサーバ入れ替えがAPに与える影響

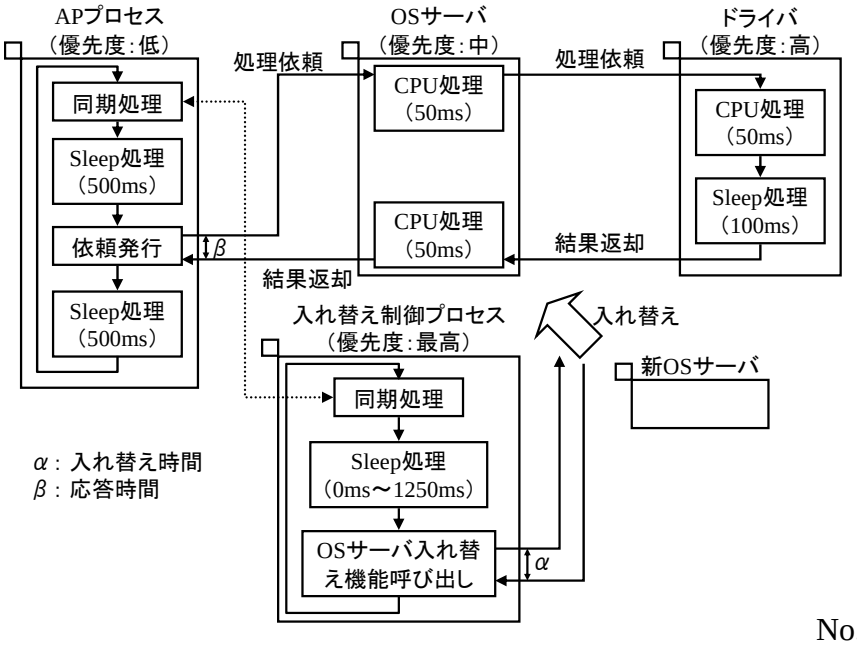
＜測定項目＞
（1）OSサーバ入れ替え機能呼び出し ～ 完了までの時間（入れ替え時間）
（2）APがOSサーバに処理依頼を発行 ～ 結果受取までの時間（応答時間）

測定環境

OS	***AnT***
CPU	Intel Celeron 2.0GHz
メインメモリ	768MB（266MHz）
ハードディスクドライブ	IDE, 40GB, 5400rpm
OSサーバプログラムサイズ	8621B

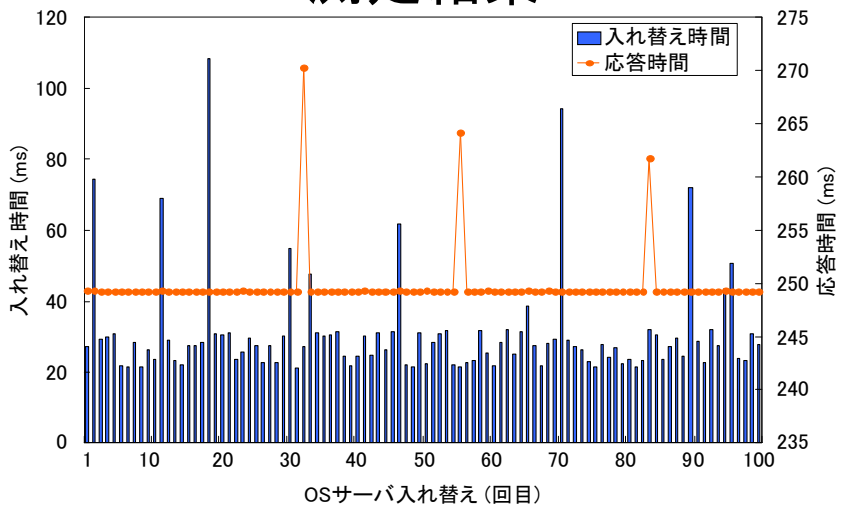
No.10

測定の様子



No.11

測定結果



（1）入れ替え時間は最大で110ms程度

（2）応答時間への影響は最大で20ms程度

入れ替えが応答時間に与える影響は18%と小さい

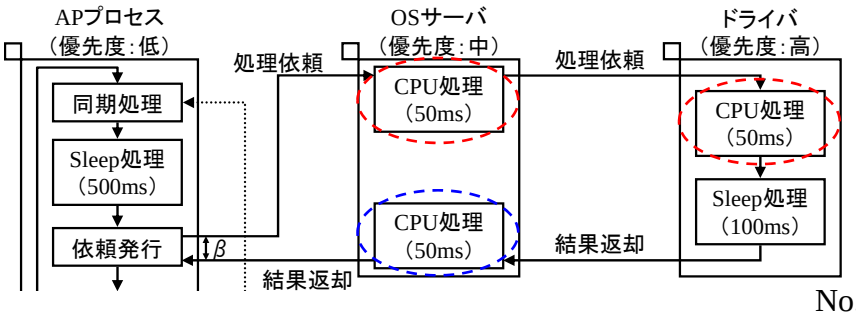
No.12

結果分析
（入れ替え時間）

（1）入れ替え時間の大半は新OSサーバのプロセス生成時間

（2）OSサーバやドライバのCPU処理中にOSサーバ入れ替え機能が呼び出された場合，入れ替え時間が増加

⇒ OSサーバやドライバのCPU処理完了まで入れ替えが待たされるため



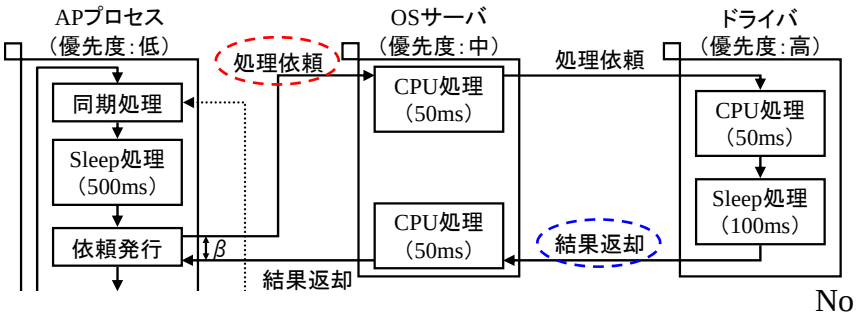
No.13

結果分析
（応答時間）

（1）APプロセスやドライバのSleep処理中に入れ替えを行うため影響小

（2）APからの処理依頼時，またはドライバからの結果返却時に入れ替えを行っている場合，応答時間が増加

⇒ 入れ替え完了まで新OSサーバのCPU処理が待たされるため



No.14

まとめ

(1) OSサーバ入れ替えの設計方針
　(A) OSサーバが保有する情報の最小化
　(B) 新旧OSサーバ間における処理の継続
(2) 引き継ぎ情報の処理方式
　(A) 要求情報の扱い
　(B) 状態情報の扱い
(3) 評価
　－　入れ替え時間が応答時間に与える影響は小さい

＜残された課題＞
－　詳細な評価

No.15

静的解析によるPCクラスタの省電力化に向けた最適処理台数の見積もり

山本　克也†　　桑原 寛明‡　　國枝 義敏‡
†立命館大学大学院理工学研究科 ‡立命館大学情報理工学部

1　はじめに

タンパク質構造解析などの生物・化学分野や，気候のシミュレーションなど多岐の分野において，大規模科学計算は多く用いられている．これらの分野では今後も計算機の性能向上に対する需要が高いと考えられる．

しかし一方，高性能計算機システムにおける消費電力の増加が問題となりつつある．これは，プロセッサの性能向上に伴い計算機単体の電力消費が増加していることが大きな原因である．特に複数台の計算機をネットワーク接続したPCクラスタであれば，計算機の数に比例して電力が増加するのは明らかである．消費電力の増大に伴って発熱量が増加し，熱暴走などによるシステム全体の信頼性が低下するので，冷却等の問題から単位面積あたりの実装密度を抑えざるをえないなどの問題が生じている．

そこで本研究では，PCクラスタの省電力化を目的とした，静的解析よる処理台数の制御を提案する．プログラム中のデータ量や通信量から異なる台数によるプラグラムの大まかな挙動の違いを見積もり，省電力の観点から最も効率の良い並列処理が行うことができる台数を導出する．

2　PCクラスタシステムによる並列処理

2.1　消費電力

図1は，PC内における消費電力の内訳である[1]. CPUが消費する電力がPC全体の3割程度であることがわかる．PCクラスタにおいて，台数を増やすことの利点はCPU数の増加による計算能力の向上である．しかしCPUを増やすためにPC1台を増設すると，CPUの省電力の3倍程度の消費電力が増加することとなる．

CPUの消費電力の割合が比較的少ないことは，DVFSなどの手法[2]を用いてCPUの消費電力を抑えたとしても，PC全体としての省電力効果が薄いことを示す．たとえCPUの消費電力が3割に抑えられたとしても，PC全体としての省電力効果は1割にも満たない．

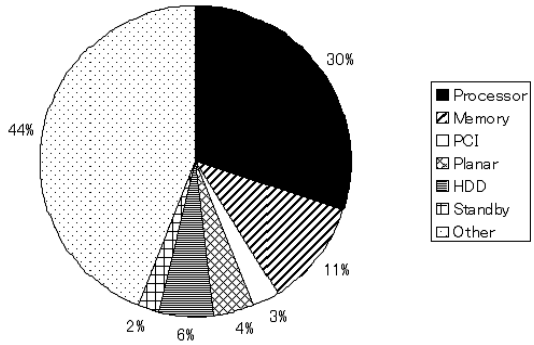


図1: 典型的なPC本体の消費電力の内訳[1]

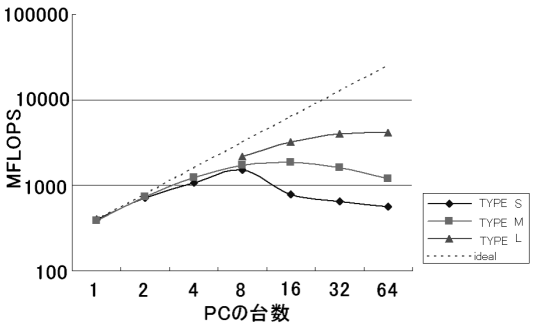


図2: 典型的な並列プログラムの並列実行結果[4]

2.2　並列処理における台数効果

図2は，姫野ベンチマーク[3]の実行結果である[4]. 速度向上の飽和，速度の逆転，などの現象がみられることがわかる．これらの現象が発生する主な原因として，分散メモリ型並列処理システムでは避けられないデータ通信の存在が挙げられる．一般に並列プログラムでは処理台数を増やせばれて並列処理部分の実行時間は短くなるが，逆に通信部分の実行時間が多くなるからである．

つまり，PCクラスタにおいては必ずしも台数を増やすほどに速度が向上するわけではない．そのため，与えられたプログラム毎に最適な台数で並列処理を行うことが重要となる．

Estimation of optimal number of executors in a PC cluster to cut down electronic power consumption.
Katsuya Yamamoto†, Hiroaki Kuwabara‡and Yoshitoshi Kunieda‡
†Graduate School of Science and Engineering, Ritsumeikan University
‡College of Information Science and Engineering Ritsumeikan University

3 提案手法

本研究では，あらかじめ MPI で並列記述されたプログラムを PC クラスタで並列処理させる前提で，PC クラスタの省電力化に向けた処理台数の最適制御を提案する．与えられたプログラムを解析し，通信量増加などによる性能低下点を見極め，最も効率の良い台数で処理を行わせることを目標とする．本方式では CPU のみならず PC そのものの台数を制御するため，大きな省電力効果が見込めると考えられる．

そのための手法として，まずソースコードから配列のサイズやループの回数，通信関数の数などを静的に解析し，それらから与えられたプログラムの演算量やデータ量，通信量などを大まかな求める．

次に，それらの解析結果をパラメータとして予測モデルに適用する．ここで用いる予測モデルとは，解析したプログラムを何台で実行すればどれのくらいの実行時間になるのか，その中での処理時間と通信時間の割合はどの程度か，などの見積もりを行うものである．

なお，ここで解析対象とするのは MPI によって並列化済みのプログラムとする．

3.1 提案モデル

プログラムの挙動を予測するため，図 3 に単純化されたモデルを示す．

式 (1) は総実行時間を見積もる式である．計算時間と通信時間の合計で求めることができる．

式 (2) では，演算回数と台数から計算時間を見積もる．演算回数はプログラム全体でどの程度の計算を行うのかを示し，それを何台で分担するかによって理論的な計算時間が決定する．

式 (3) では，通信量と台数から通信時間を見積もる．通信量は，さらに通信回数，データ量，台数より見積もることができる．全体でやりとりするデータを何台で分割し，どのような通信関数でやりとりするかによって通信量はほぼ決定し，さらにそれを何台の間で通信するかによって理想的な通信時間を予測することができる．

$$(\text{実行時間}) = \alpha(\text{計算時間}) + \beta(\text{通信時間}) + \lambda_1 \quad (1)$$

$$(\text{計算時間}) = f_1(\text{演算回数，台数}) = \gamma\left(\frac{\text{演算回数}}{\text{台数}}\right) + \lambda_2 \quad (2)$$

$$(\text{通信時間}) = f_2(\text{通信量，台数}) = \delta(\text{通信回数} \times \text{データ量}) + \lambda_3 \quad (3)$$

図 3: 提案モデル

3.2 モデルの具体化

前項で述べたモデルの係数は，実測値から回帰分析によって求める．実測値を得るには，一般に使用されているベンチマークプログラムや，通信部分や処理部分がはっきり分かれた自作のプログラムなどを用いる．

経験則に基づくモデルとなるため高い精度とはならないことが予測されるが，本研究では省電力化に向けた大まかな制御を目標としているため，精度の低い予測モデルでも十分省電力が達成できると考えられる．

4 おわりに

本稿では，PC クラスタの省電力化を目的とした，静的解析よる処理台数の制御を提案した．プログラム中のデータ量や通信量から処理台数を変化させたときの消費電力を大まかに見積もり，その観点で最も効率の良い並列処理が行うことができる台数を導出する．

そのための手法として，静的解析で得たパラメータを予測モデルに適用し，プログラムの実行時間，処理時間と通信時間などを見積もる．これらからどの台数で行えばどの程度の処理時間，電力消費が必要なのかを大まかに把握することで，効率の良い台数を選択することが可能となる．

今後の課題として，現在は解析の対象を MPI で既に並列化されたプログラムに限定しているが，提案方式を並列化コンパイラに実装し，逐次のプログラムを並列化すると同時に最適台数を導出できるようにすることが挙げられる．また，現在はモデルの性質上，回帰分析を行った並列処理環境に予測値が強く依存するため，環境パラメータ等を用いることによって複数環境に適応できるようにすることも必要となる．

参考文献

[1] 根本雅樹．日本 HP の HPC 向け PC クラスタへの取り組み,2008，PC クラスタワークショップ in 京都, 2008. http://www.pccluster.org/event/workshop/pcc2009osaka/

[2] 高柳圭孝:PC クラスタにおける DVFS を用いた並列プログラムの電力量削減のための基礎研究， 平成 19 年度立命館大学情報理工学部修士論文, 2008.

[3] 姫野ベンチマーク. http://accc.riken.jp/hpc/HimenoBMT/index.html.

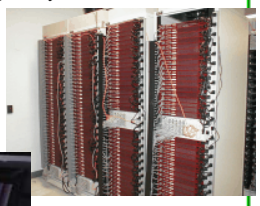
[4] 広島国泰寺高校科学部物理班:事例紹介 1，KNOPPIX クラスタ情報交換会, 2004. http://www.is.doshisha.ac.jp/SMPP/report/2004/041228.html

静的解析による PCクラスタの省電力化に向けた 最適処理台数の見積もり

立命館大学　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
山本克也　桑原寛明　國枝義敏

クラスタシステム

- クラスタシステムとは？
  - 複数の計算機をネットワークで接続して構成
  - 単一計算機では得られない処理性能を発揮
    - HPC(High Performance Computing)クラスタ
      - 並列計算、科学計算
    - HA(High Availability)クラスタ
      - 負荷分散、高可用性

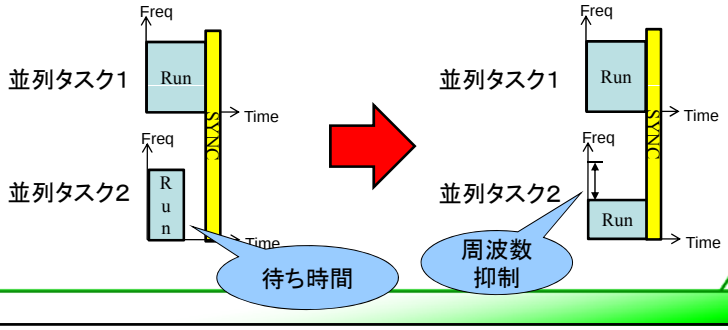


研究背景・目的

- PCクラスタシステムの消費電力問題
  - コスト増加
  - データセンターにおける深刻な問題に
- 目的
  - PCクラスタを対象とした省電力化
    - データセンター等，大規模環境向け
  - 効率の良い資源運用を目指す
    - 速度だけでなく，電力面を考慮

関連研究

- DVFSによる周波数制御
  - DVFS(Dynamic Voltage and Frequency Scaling)
    - 動作周波数および動作電圧を動的に変更する機能
  - 負荷の不均衡があるタスクにDVFSを適用

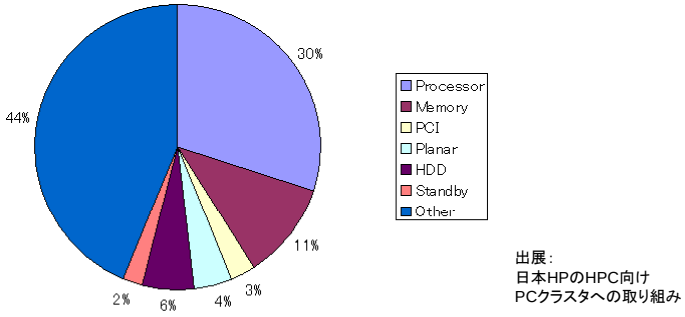


PCクラスタシステムの特徴

- プロセッサだけが電力を消費するわけではない
- 必ずしも台数効果が得られるわけではない

消費電力の内訳

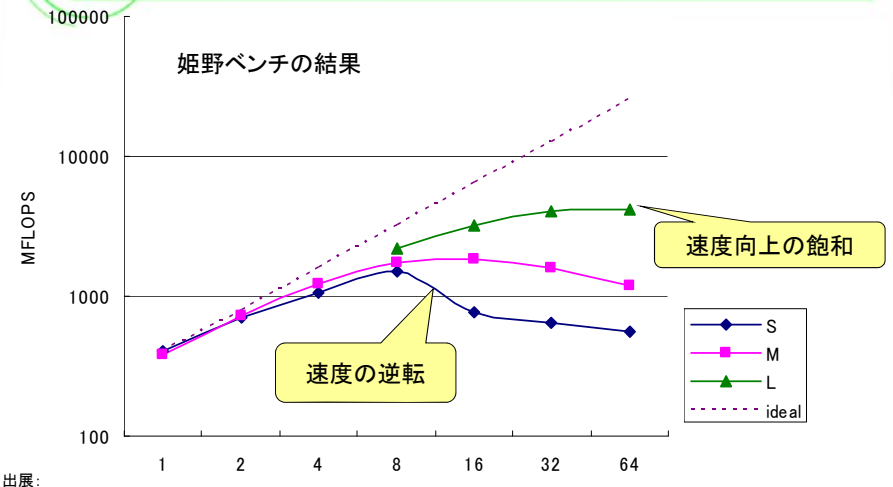
- プロセッサの消費電力は全体の3分の1程度
- 台数が増えるほど無駄な電力も増える



出展:
日本HPのHPC向け
PCクラスタへの取り組み

CPUの消費電力だけを抑えても効果薄

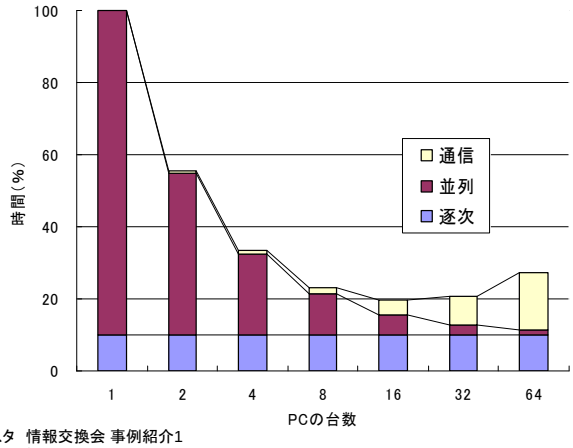
速度向上の限界



出展:
KNOPPIX クラスタ 情報交換会 事例紹介1
広島県立広島国泰寺高校 科学部物理班

速度向上を遮る原因

- 台数を増加することで通信回数が増加



出展:
KNOPPIX クラスタ 情報交換会 事例紹介1
広島県立広島国泰寺高校 科学部物理班

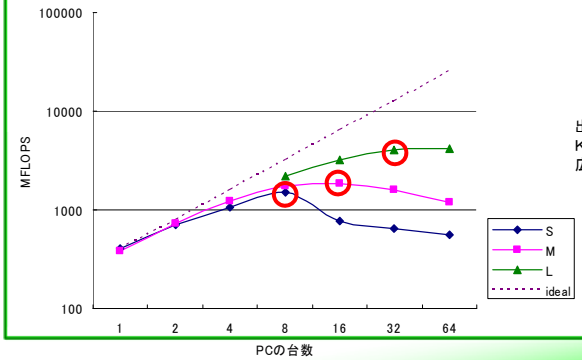
PCクラスタシステムの特徴

- プロセッサだけが電力を消費するわけではない
  - 全体の3分の1程度
    - 台数が増えるほどCPU以外の電力も増える
    - CPUの消費電力だけを抑えても効果が薄い
- 必ずしも台数効果が得られるわけではない
  - 速度向上の限界・速度の逆転
    - 通信による弊害

最適な台数は場合により異なる

研究概要

- 台数制御を行うことで省電力化
  - CPUだけではなく実行台数そのものを制御
    - 使わなくなったマシンは別処理に
  - 最も効率よく並列処理を行うことができる台数で実行



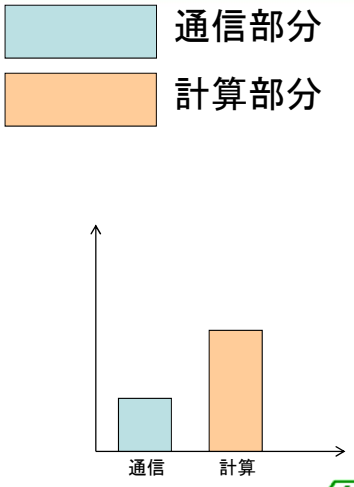
出展:
KNOPPIX クラスタ 情報交換会 事例紹介1
広島県立広島国泰寺高校 科学部物理班

研究概要

- 静的解析を用いて最適台数を導出
  - プログラムの挙動を予測
  - プログラムの構造から先の予測が可能
    - 一部プロファイリング情報も用いる
- 『入力』は並列化済みプログラム
  - 現在の対象はMPI
    - Message Passing Interface
    - 分散メモリ型の標準的な並列プログラミング規格
    - メッセージパッシングを用いた並列処理

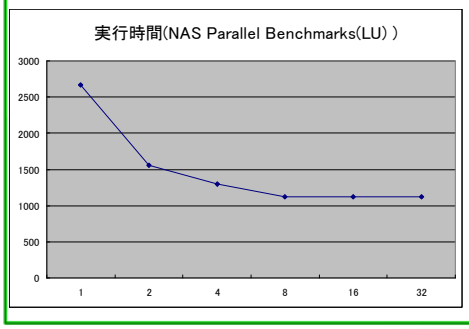
研究概要

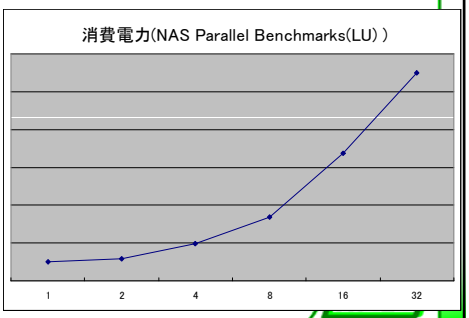
```
MPI_Init (&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD,...);

if(my_rank==0){
   fp = fopen(FILENAME,"r");

   fscanf(fp,"%d",&data_num);

   for(i=0;i<data_num;i++){
      fscanf(fp,"%d",data);
      data++;
      ……
   }

    MPI_Send (&local_result, 1,...);
}
else{
   MPI_Recv (&local_result, 1,...);
   for(i=0;i<data_num;i++){
      data=data*num;
      ……
   }
}
```



研究概要

- 静的解析を用いて最適台数を導出
  - プログラムの挙動を予測
  - 実行時間や消費電力を予測





提案手法

- 判断基準
  - 演算量
  - データ量
    - 明らかに少ない場合は少ない台数で
  - 通信量との比
    - 通信量が少なければ多い台数を指定
- 予測モデルを生成
  - 実行時間の早さではなく実行効率を尺度に

モデル案

- (実行時間)＝α(計算時間)＋β(通信時間)＋λ_1
  - (計算時間)＝f_1 (演算回数，台数)
    ＝γ(演算回数 / 台数) ＋ λ_2
  - (通信時間)＝f_2 (通信量，台数)
    ＝f_2 (f_3 (通信関数実行回数，データ量，台数)，台数)
    ＝δ(通信実行回数×(データ量 / 台数)) ×(台数) ＋ λ_3
- 回帰分析で係数を決定

モデル案

- (処理量)＝α(計算量)＋β(通信量)＋λ_1
  - (計算量)＝f_1 (演算回数，台数)
  - (通信量)＝f_2 (部分通信量，台数)
    ＝f_2 (f_3 (通信関数実行回数，データ量，台数)，台数)
- 回帰分析で係数を決定

モデル指針

- 経験則に基づくモデル
  - ベンチマークプログラム＋自作の『綺麗な』プログラム
- 大雑把な予測を目標に
  - 最終目標は，台数制御という大雑把な制御
  - 細かな挙動の予測までは必要としない

今後の課題

- 複数環境への適応
  - 経験則は経験を積んだ環境に依存
    - 環境パラメータを設置
      - 計算や通信の最小単位を予め設定
      - 最小単位での実行にかかる時間をパラメータに
- マルチコアへの適応
  - 通信の違いなどを考慮
- 並列化コンパイラに実装
  - 並列化と同時に最適台数を提言

まとめ

- PCクラスタ向け省電力手法の提案
  - 静的解析を用いた最適台数の導出
    - 演算量やデータ量，通信量を判断基準に
  - 効率の良い並列化を
- 今後
  - コンパイラへの組み込み
  - 複数環境，マルチコアへの適応

複数のサービスが稼働するセンサアクチュエータノード上の資源管理機構

金丸 達雄† 横田 裕介†† 大久保 英嗣††
† 立命館大学大学院理工学研究科 †† 立命館大学情報理工学部

1 はじめに

無線センサアクチュエータネットワーク (以下, WSAN) で用いるノードは，コスト効率の良い動作，および資源利用 (センサ，アクチュエータ，無線通信装置) の要求を満たす動作が重要である．前者は，ノード数に比例した，配備コストの増加を防ぐためである．後者は，資源利用によって，WSAN 上サービスの要求を実現するためである．すなわち，適切な資源利用によって，要求通りに実世界へ対応するためである (図 1).

コスト効率を実現するため，複数のサービスにより単一のノードを共有する方法がある．これにより，全サービスに対するノードの数を削減できる．

しかし，複数のサービスが稼働する環境では，資源利用の競合による影響で，資源利用の要求が満たせない場合がある．また，その結果として，サービスが破たんする可能性がある．これは，競合により以下の影響が発生するためである．

タイミングへの影響 ブロックにより，資源利用が遅延して資源利用が無意味になる場合がある．また，締切りベースのスケジューリングの影響により，資源利用の開始が前倒しされすぎる場合がある．前者の例として，隣接ノードのスリープ後に通信してしまう場合がある．後者の例として，超音波センサの競合を予期したサービスが，物体通過前にトラッキングを行うといった場合がある．

資源利用の機会の減少 他サービスの資源利用の期間によっては，資源利用ができずサービスが実行できない場合がある．これは，あるサービスがイベントを待つ場合に発生する．例えば，常に温度センサを利用するサービスが稼働している場合，他のサービスは温度センサを利用できない．

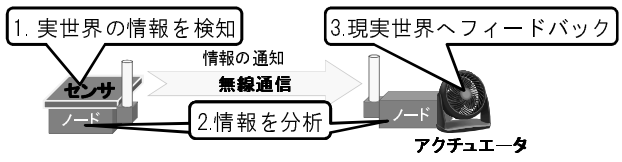


図 1 無線センサアクチュエータネットワークの動作概要

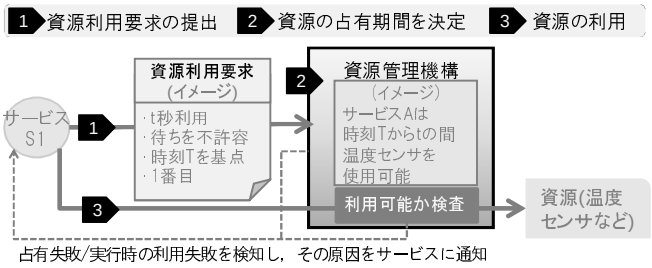


図 2 資源管理機構の概要

失敗の波及 他サービスの影響で資源利用に失敗した場合，失敗を検知できずに後の資源利用を実行する場合がある．例えば，物体通過前のトラッキング結果を正しいデータとして外部に通知する場合がある．これは，締切ベースでエラー検知した場合，資源利用が前倒しされすぎて発生する．

また，従来の WSAN ノード用 OS の機能 (ロック，スケジューラ) では，競合を予期できないため，サービスは他サービスからの競合をふまえた動作ができない．結果，他サービスからの競合による影響を回避できず，他ノードとの連携，および実世界への対応に失敗する場合がある．

以上を踏まえ，本研究では，予約による資源利用を用い，競合による資源利用失敗を防止する資源管理機構を提案する．結果，本機構を用いることで，複数のサービスによる実世界への対応を保証できる．

2 提案する資源管理機構

本機構は，占有期間の予約，および資源利用の失敗検知を用いて 1 章の問題を解決する．提案する資源管理機構の概要を図 2 に示す.

表 1　資源利用要求の主な項目

項目名		指定する内容
利用時間	開始時刻	時刻
	期間	時間
利用する処理		資源を利用するスレッド
使用する資源		センサ，アクチュエータ，無線通信装置など
順序		何番目に利用するか
占有方法		排他占有，共有占有 (共有する状態)

2.1　資源管理機構による資源利用要求の保証

タイミングの影響の回避は，本機構の予約利用を用いた，資源の占有期間の予約で実現される．予約利用は，資源利用要求 (表 1) を提出し，占有期間を決定した後，資源利用を行う方法である．資源の占有期間は，他サービスによる資源利用のブロックを受けないことが保証される．以下に予約利用の手順を示す．

1. サービスは，複数の資源利用要求を提出する．
2. 資源管理機構は，資源利用要求に基づき，そのサービスが資源を占有する期間を決定する (以下，予約と示す)．資源の占有期間は，資源利用要求の [利用時間] によって決定される．
3. サービスは，資源管理機構が決めた占有期間に資源を利用する．資源利用は，資源利用要求の [利用する処理] によって実行される．

他サービスの影響を受けない期間に資源利用が行われるため，タイミングは保証される．

また，利用機会の減少を回避するため，本機構は，複数のサービスによる資源の共有を実現する．これは，共有占有を用いて実現される．共有占有は，サービス間である状態の資源を共有する方法である．資源の状態を固定することで，資源利用の結果をキャッシュしてサービス間で共有できる．なお，共有ロックは，資源利用要求を用い，予約利用時に指定される．

さらに，失敗波及を防止するため，本機構は資源利用の順序管理を行う．順序管理により，ある資源利用が失敗した場合において，以降の資源利用を中止できる．そのため失敗が波及しない．なお，資源利用の順序は，資源利用要求を用いて指定される．

さいごに，競合の影響をふまえたサービスの動作は，本機構による資源利用のエラーの検知，および原因の通知によって実現される．サービスは，エラー原因を把握し，次回の資源利用の失敗を回避することができる．なお，エラーの定義は，予約利用の際に資源利用要求から与えられる．エラー検知とその対策の例を 2 つ挙げる．[利用する処理] がオーバランした期間をもとに，次回の資源利用の期間を正しく設定することができる．また，競合により占有期間が予約できない場合，資源管理機構は競合期間の先頭と終端をサービスに通知する．これを利用することで，タイミングへの影響を防ぐことができる．具体的には，競合期間の先頭より前に占有期間を予約することで，後の資源利用に対する遅延を防止できる．

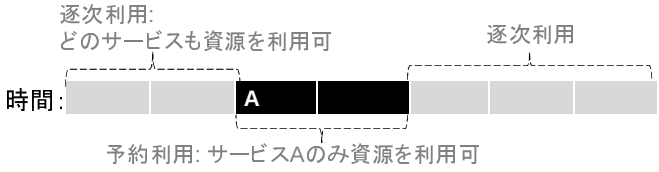


図 3　予約利用と逐次利用の関係

2.2　資源管理機構の利用例

警備サービス (予約利用)，および測位サービス (逐次利用，後述) における資源利用の例を示す．なお，逐次利と用は，予約利用がされていない期間にサービスが資源利用する手法である．これは，サービスの事前処理 (資源利用期間の決定のためのテストなど)，および資源利用に関する保証が不要なサービスを想定している．それぞれの資源利用の関係を図 3 に示す．

警備サービスは，予約利用を用いて以下の挙動を行う．

1. ノード上の超音波センサ S で物体を検知する (以後，この処理を [検] と示す)．
2. 1 で物体が検知された場合，S を 2 回利用し，物体の移動方向を確認する (以後，[析] と示す)．
3. で物体の挙動を把握した場合，無線通信装置 C により，物体の挙動を基地局へ通知する (以後，[通] と示す)．

また，警備サービスは，表 2 の資源利用要求を提出する．

一方，測位サービスは，以下の動作を行う．

1. 移動ノードから測位要求を C で受信する (以後，この処理を｛要｝と示す)．

表 2　警備サービスを実現する資源利用要求

	資源利用要求 1	資源利用要求 2	資源利用要求 3	資源利用要求 4
利用開始	T	$T+t_s$	$T+2t_s$	$T+3t_s$
期間	t_s	t_s	t_s	t_c
資源	S	S	S	C
処理	検	析	析	通
順番	1	2	3	4
占有方法	排他	排他	排他	排他

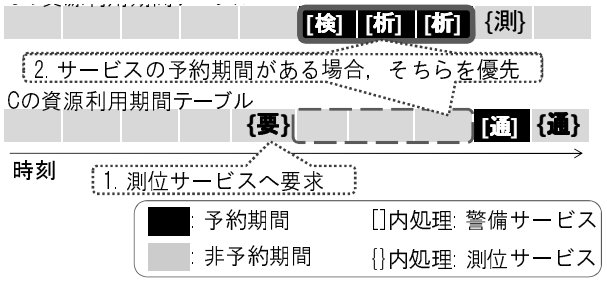


図 4　警備サービスと測位サービスによる資源利用の例

2. S で隣接ノードの位置を測距する (以後，{測} と示す).

3. 測距データを基地局に通知し，測位を要求する (以後, {通} と示す).

以上の 2 サービスが稼働した場合，資源利用の結果は図 4 のようになる．このように，警備サービスの資源利用は保証される．これは，資源利用要求によって配置された占有期間中は，他サービスによって資源の内部状態を変更できないためである．

3　資源管理機構の実装予定計画

LiteOS [1] を用いて MicaZ 上に実装を行う．なお，サービスは，LiteOS 上のプロセスを利用する．資源管理機構のモジュール構成とモジュール間の関連を図 5 に示す．また，各モジュールの説明を以下に示す．

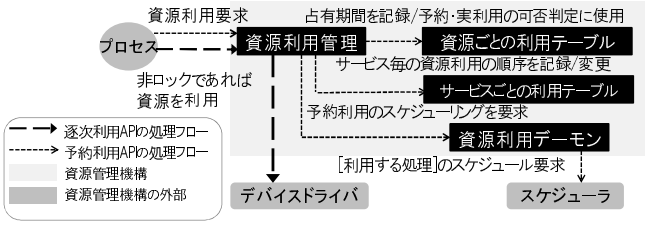


図 5　資源管理機構のモジュール構成

資源利用管理　予約利用，逐次利用の API を提供する．予約の可否の判定と予約，および資源利用の可否の判定を行う．

資源ごとの利用テーブル　資源利用がいつ予約されているかを管理するテーブルである．資源ごとにテーブルが作成される．

サービス毎の利用テーブル　サービスから見た資源利用の順序を管理するテーブルである．

資源利用デーモン　資源管理機構に関連する処理をスケジューリングする．具体的には，資源利用要求の [利用する処理]，エラー検知，エラーハンドル処理である．

評価は，ノード上で 2 サービスを稼働させ，競合が回避できるか確認する．稼働させるサービスは，火災検知サービス (予約利用)，および空調サービス (逐次利用) を模したものを考えている．

4　おわりに

本稿では，複数のサービスからの資源利用要求を保証する資源管理機構について述べた．今後，実装を行う．

参考文献

[1] Cao, Q., Abdelzaher, T., Stankovic, J. and He, T.: The LiteOS Operating System: Towards Unix-Like Abstractions for Wireless Sensor Networks, *Information Processing in Sensor Networks, 2008. IPSN'08. International Conference on*, pp. 233–244 (2008).

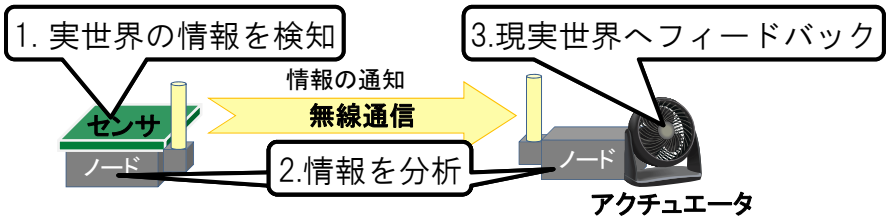
複数のサービスが稼働する センサアクチュエータノード のための資源管理機構

立命館大学大学院
大久保･横田研究室
金丸 達雄

発表内容

- 研究背景
  - 無線センサアクチュエータネットワーク
  - 複数のサービスが稼働するノードの特徴，要求
- 提案する資源管理機構
  - 占有期間の予約による，資源利用の要求を保証
  - タイミングの保証，資源利用の機会の増加，失敗検知と対策
- 資源管理機構を介した資源利用の例
  - 複数サービスによる資源利用
- 資源管理機構の実装(予定)
- 関連研究との比較
- 今後の予定
- まとめ

WSAN:無線センサ・アクチュエータ ネットワーク



- 資源利用を中心とした，ノード上の動作が重要
  - 資源：センサ，アクチュエータ，無線通信機
  - 実世界への対処≒資源利用
- コスト効率のよいノードの動作が重要
  - ノード数が多く，配備のコストが高い

複数のサービスが稼働するノード

- 複数のサービスで単一のノードを共有
  - 全サービスに対し，必要となるノード数を削減
- 稼働するサービスの特徴
  - サービスは省電力化のため，実世界に対応するときのみ資源利用
    - WSANの一般的な要求(長寿命化による再配備コスト削減)を実現
  - サービスは無線通信でノード上に配布
    - サービス配備コスト削減を実現，ただし，ノード上の環境は予測できない
- 各サービスが，実世界への対応を成功できる必要
  - 資源利用の競合による問題(後述)を回避する必要

複数のサービスが稼働するノード

- 複数のサービスで単一のノードを共有
  - 全サービスに対し，必要となるノード数を削減
- 稼働するサービスの特徴
  - *競合による問題を解決し，複数のサービスからの資源利用の要求を保証するノード上の資源管理機構が必要*
  - サービスは無線通信でノード上に配布
    - サービス配備コスト削減を実現，ただし，ノード上の環境は予測できない
- 各サービスが，実世界への対応を成功できる必要
  - 資源利用の競合による問題(後述)を回避する必要

他サービスからの競合による資源利用への影響

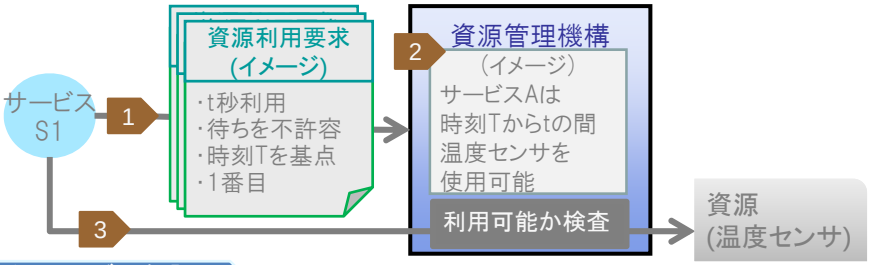
- タイミングへの影響
  - ブロックにより，資源利用が遅延し，利用が無意味に
    - 隣接ノードのスリープ後の通信，物体通過後のトラッキング
- 資源利用の機会が減少
  - 資源利用しているサービス以外，資源利用ができない
    - 複数のサービスが隣接ノードからの通信を待つ場合，通信の到着を検知できない期間が発生
- 失敗の波及
  - 失敗した資源利用の結果を用いて，後の資源利用を実行
    - 物が通る前にトラッキング
      → トラッキングの実施は締切内であるため，データを送信(ただし無意味)

他サービスからの競合による資源利用への影響

- タイミングへの影響
  - ➜タイミングを保証：
    予約により，資源の占有期間をあらかじめ決定
- 資源利用の機会が減少
  - ➜資源を共有：
    資源が共有できる状態のとき，資源を共有して利用機会を増加
- 失敗の波及
  - ➜失敗を検知し波及防止：
    - 前回の資源利用が成功した場合のみ，次回を実施
    - 予約・資源利用の失敗原因(競合など)をサービスに通知し対策

提案する資源管理機構

1 資源利用要求の提出　2 資源の占有期間を決定　3 資源の利用



タイミングの保証

- サービスからの資源利用要求に従い，資源の占有期間を割当て

資源の共有

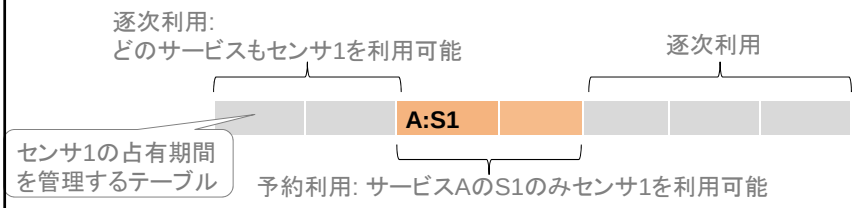
- 資源の内部状態のみを保証し，資源の同時利用を可能にする占有方法
  - 一貫性のため，通常の資源単位の占有方法も利用

失敗の検知・対策

- 資源利用の順序を管理し，失敗が発生した場合，以降の資源利用を停止
- エラー検知とエラーハンドル処理による，資源利用へのフィードバック

想定する資源の利用方法

- 予約利用
  - 資源利用要求を用い，予約された期間に資源を利用
    - タイミング，失敗の波及回避を保証したいサービスのための方式
- 逐次利用
  - 予約利用がされていない期間に自由に資源を利用
    - サービスの事前処理(資源利用期間の決定など)を想定
    - 資源利用に関する保証が不要なサービス

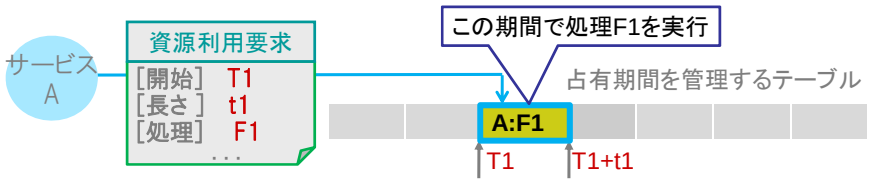


主な資源利用要求

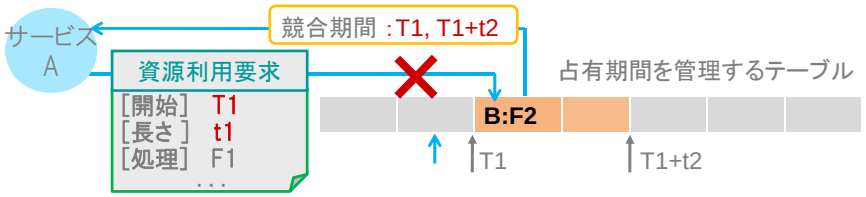
- 使用する資源
  - センサ，無線通信機，アクチュエータなど
- 利用時間
  - 開始時刻，長さ
- 占有方法(後述)
- 利用する処理
  - 資源を利用する処理(スレッド)を指定
    - 短期に終了する処理を想定
- 順序
  - 資源の利用順
    - 同時利用する場合，同じ番号を指定
- エラーハンドル処理(後述)

資源利用のタイミングを満たす占有期間の決定

- 占有期間は，資源利用要求の[利用時間]に従って決定



- 占有できない場合，競合期間の[開始時刻][終了時刻]を返却
  - 資源利用を前倒しすることで，後の資源利用の遅延を回避可能
    - 遅延が許容できる場合は，逐次利用

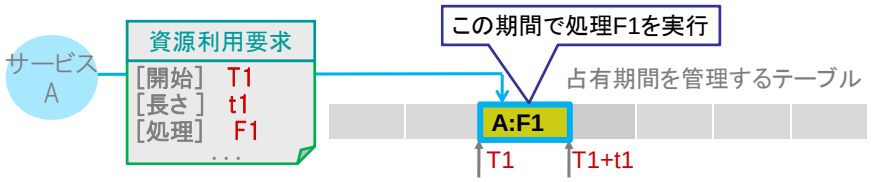


11

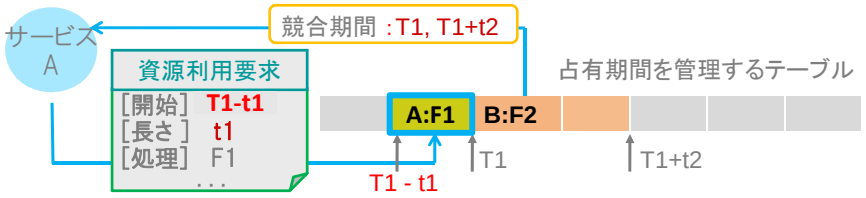
立命館大学大学院 大久保・横田研究室

資源利用のタイミングを満たす占有期間の決定

- 占有期間は，資源利用要求の[利用時間]に従って決定



- 占有できない場合，競合期間の[開始時刻][終了時刻]を返却
  - 資源利用を前倒しすることで，後の資源利用の遅延を回避可能
    - 遅延が許容できる場合は，逐次利用

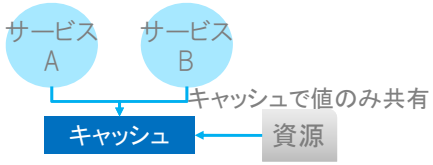


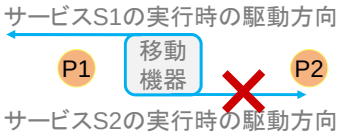
12

立命館大学大学院 大久保・横田研究室

2種類の占有方法による資源利用の共有と一貫性の実現

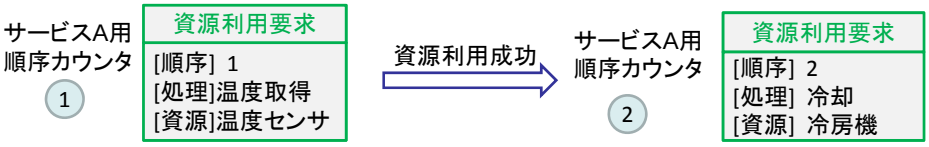
- 共有占有
  - サービス間で，ある内部状態の資源を共有
    - 資源利用の結果をキャッシュしてサービス間で共有
    - 例：単純なセンサの読出し，無線通信機の受信状態
- 排他占有
  - 単一のサービスのみが，資源の内部状態を変更
    - 他サービスの資源利用による，影響防止(誤作動の防止)
    - 例：移動機器の方向決定，複雑なセンサ，無線通信の送受信





失敗の波及を回避するための順序管理

- 現在の資源利用の順序を，順序カウンタで管理
  - 順序カウンタ＝資源利用要求の[順序]
    →その資源利用が実行
- 資源利用の結果に基づき，順序カウンタは以下の挙動
  - 資源利用が成功→ 順序カウンタを増し，次の[順序]の資源利用
  - 資源利用が失敗→ 以降の[順序]の資源利用を中止
    （失敗の波及を防止）
- 例：1ノードにおける空調



次回の資源利用にフィードバックするための エラーハンドル処理

- 資源利用が失敗した場合，エラーハンドル処理が実行
  - 資源利用要求の提出時に，登録された任意の処理
  - 失敗の監視は，バックグラウンドで実行
- 失敗の原因・失敗時の情報がエラーハンドル処理に通知
  - 失敗：[利用する処理]のオーバラン，[利用する処理]からの失敗の通知，高優先度の予約利用による占有期間の横取り
  - 情報：オーバラン時間，競合期間，競合相手の優先度，任意の情報
    - 情報を基に占有期間を再度決定(利用期間の増加，競合期間の回避)

複数サービスによる資源管理機構を介した 資源利用の例(1/2)

- 警備サービス(予約利用)
  1. 超音波センサSで物体を検知 (以後，[検]と表記)
  2. 物体を検知した場合，Sを2回利用し移動方向を分析([析])
  3. 方向を把握したのち，無線通信機Cにより基地局へ通知([通])
- 位置情報サービス(逐次利用)
  1. 移動ノードから測位要求を受信([要])
  2. Sで隣接ノードの位置を測距([測])
  3. 基地局に測位を要求([通])

複数サービスによる資源管理機構を介した資源利用の例(2/2)

■ 警備サービスの資源利用要求

利用開始	期間	資源	処理	順番	占有方法
T	ts	S	検	1	排他
T+ts	ts	S	析	2	排他
T+2ts	ts	S	析	3	排他
T+3ts	tc	C	通	4	排他

■ 資源利用の結果

Sの資源利用期間テーブル

Cの資源利用期間テーブル

•警備サービス
•位置情報サービス
：非予約期間
：予約期間

17

複数サービスによる資源管理機構を介した資源利用の例(2/2)

■ 警備サービスの資源利用要求

利用開始	期間	資源	処理	順番	占有方法
T	ts	S	検	1	排他
T+ts	ts	S	析	2	排他
T+2ts	ts	S	析	3	排他
T+3ts	tc	C	通	4	排他

■ 資源利用の結果

Sの資源利用期間テーブル

Cの資源利用期間テーブル

他サービスの予約期間がある場合，そちらを優先

•警備サービス
•位置情報サービス
：非予約期間
：予約期間

18

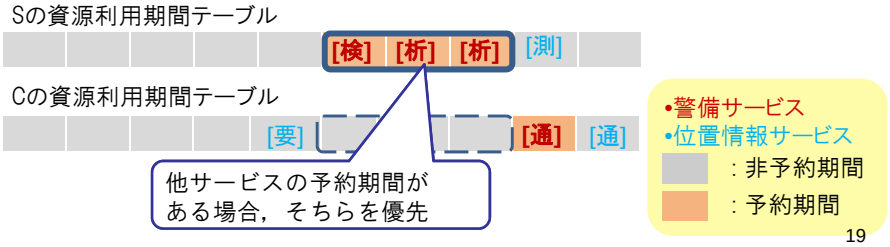
立命館大学大学院 大久保・横田研究室

複数サービスによる資源管理機構を介した資源利用の例(2/2)

- 警備サービスの資源利用要求

利用開始	期間	資源	処理	順番	占有方法
T	ts	S	検	1	排他
T+ts	ts	S	析	2	排他

- 予約利用により資源利用の要求を保証
  - 警備サービス以外は，資源の内部状態を変更不能
- 資源利用の結果

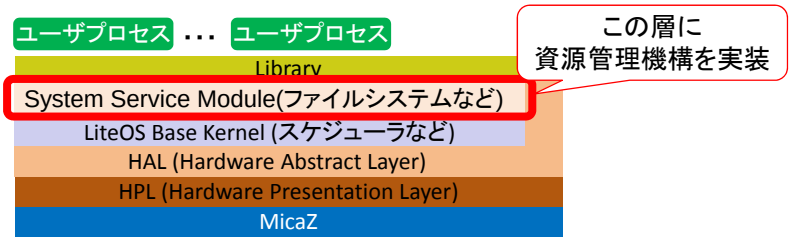


19

立命館大学大学院 大久保・横田研究室

資源管理機構の実装(予定)

- センサノードMicaZ上でLiteOSを利用
  - LiteOS上のSystem Service Moduleとして，資源管理機構を実現
  - サービスはLiteOS上プロセスで実現
- LiteOS
  - イリノイ大学のQing(チン) Cao(チャオ)らが開発
  - Unix-likeなOS：C言語で記述，マルチプロセス，外部Flash上ファイルシステム



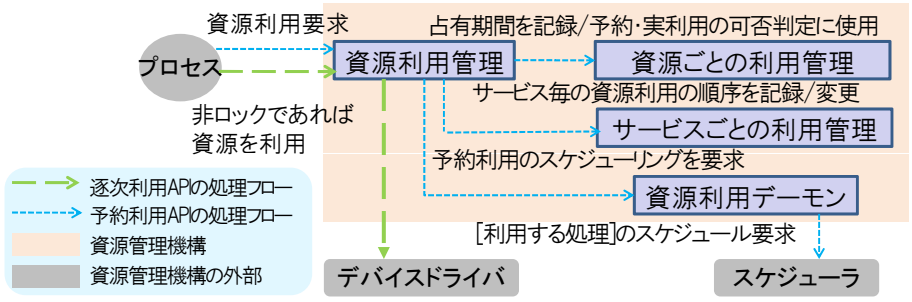
LiteOS+MicaZの構成

20

立命館大学大学院 大久保・横田研究室

資源管理機構のモジュール構成

- 資源利用管理：予約利用API，逐次利用API
- 資源ごとの利用管理：資源ごとに，資源利用のタイムテーブルを管理
- サービスごとの利用管理：サービスの資源利用の順序を管理
- 資源利用デーモン：
  [利用する処理]，エラーハンドラのスケジューリング，オーバラン監視



使用方法:コードレベル

- エラーハンドラを記述
  - 例: void errorHandler(erroCause *e) //エラー原因と情報が渡される
    - 渡された情報に基づき，資源利用要求を修正
- 資源[利用する処理]を記述
  - 例：void useFunc(void)
    - 内部で実際に資源利用する関数を呼び，呼出し元にIPCで結果を通知
- 資源利用要求の記述，および提出
  - ResorceRequest rr;
    rr.useResource = TEMP; //温度センサを利用
    rr.useFunc=func; //資源利用する関数
    ･･･(略)･･･
  - setSequenceCounter(0) //順序カウンタの設定
  - reserveResource(rr) //資源利用要求の提出
- 資源利用後，呼出し元が起床して，利用結果を確認

使用方法:方針

- サービス内で最後の資源利用から，資源利用要求を提出
  - 先に締切りを決め，後の資源利用を調整することで，後戻りの手間を回避
- 2周期以上，先まで予約
  - 他プロセスに周期を明示するため
- オーバラン時間を[利用期間]にフィードバック
  - エラーハンドラ内でオーバラン時間は取得可能
- 応答時間を向上させたい場合，逐次利用
  - [利用期間]より前に処理が終了した場合，次の資源利用まで待ちが発生
- 複数の資源を同時に利用する場合，同時に複数の資源利用要求を提出
  - デッドロックを回避(2資源に対し，各々の資源利用要求の逆順での提出を回避)
    - 同じ[順序]の資源利用は，両方満たされなければ，両方とも破棄

関連研究との比較

- OSEKTime(OSEK/VDX):指定した時刻にタスクを起動
  - 本機構と異なり，資源利用が可能かが実行前に不明
- t-kernel(Lin Guら，Virginia大学):
  ページング時チェックによる不正アクセス防止
  - 本機構と異なり，アプリケーションによるエラーハンドリングができない
- Agilla(Chien-Liang Fokら，Saint Louis大学):
  センサノード上のエージェント
  - 本機構と異なり，各プロセスで，プロセス毎の交渉プロトコルに対処する必要

今後の予定

- 実装
  - 課題：メモリ領域が少ない
    - 残り：RAM約1.3KB，EEPROM 0.9KB，プログラムフラッシュ83KB
    - 一部のデータ，プログラムを外部Flash(512KB)へ退避することで対応
- 評価
  - プログラムのサイズ
  - 複数のサービスを稼働させ，競合しないか確認
    - 火災検知サービス(予約)，空調サービス(逐次)を模したものを作成

まとめ

- 資源管理機構について発表
  - 占有期間の予約による，資源利用のタイミング保証
  - 共有占有ロックによる，資源利用の機会の増加
  - 順序管理，エラーハンドル処理失敗の検知・対策
  - LiteOSのSystem Service Moduleとして実装
- 今後の予定
  - 実装
  - 評価
    - 2サービスによる評価
    - プログラムのサイズ

センサネットワークOSにおける協調型処理代替機構

李 冉 †
† 立命館大学大学院理工学研究科

1 はじめに

近年, 半導体技術や無線通信技術の発達により, 無線通信機能とセンシング機能を持つセンサノードが開発されている. それに伴い, センサノードにより構成される無線センサネットワークへの関心が高まっている. 無線センサネットワークは斜面防災システムや交通状況監視システムなどのさまざまな場面に応用される. また, 将来のセンサネットワークでは, 周囲の環境情報の取得に使用されるだけではなく, さまざまなアクチュエータを搭載することによって, マルチロボットコントロールのようなアプリケーションを実現することも可能となる [1]. また, 無線センサネットワークは低コスト, 低電力消費, 配置しやすいという特徴を持っている. そのため, ビルや商業施設などに無線センサネットワークを設置し, 従来のネットワークに代わり, 防犯や火災検知のようなさまざまなサービスを 1 つのセンサネットワークで実行することが可能である. しかし, センサネットワークは省電力性を実現するため, センサノードには低クロックの MPU を採用することが多い. そのため, センサネットワークに対する複数のサービスの要求がある場合, 単一のノードでは決まった時間内に処理を完了できず, 応答性の低下やサービスの破たんが発生する可能性がある.

この問題を解決するために, 本研究では, 複数のノード間の協調代替処理を用いて応答性を確保する. 具体的には, あるノードがビジー状態であるとき, クエリなどのサービス要求をグループ内のあるノードに転送し, 代理応答・処理を行わせる.

2 協調型処理代替機構の概要

本研究では 1 つのセンサネットワークで複数のサービスを同時に実行する際, MPU や無線モジュールの処理速度による応答性の低下やサービスの破たんなどの問題を解決することを目的としている. ここでは, 提案手法の概要および前提条件について述べる.

2.1 提案手法

提案手法は, センサネットワークをいくつかのグループに分割し, 各グループのノード 1 つをマスタノード, ほかのノードをスレーブノードとし, スレーブノードはスリープさせる. 1 つのグループを仮想的な 1 つのセンサノードと見なし, 多数のグループ化されたノードを用いてセンサネットワークを構成する. マスタノードはデータの集約処理などで応答できない場合, スレーブノードのタスクを遠隔で呼び出し, 代替応答処理要求および処理データを送信して, 代替応答処理をさせる. 本研究の最終目的は図 1 に示すように, ユーザにフレームワークを提供することである.

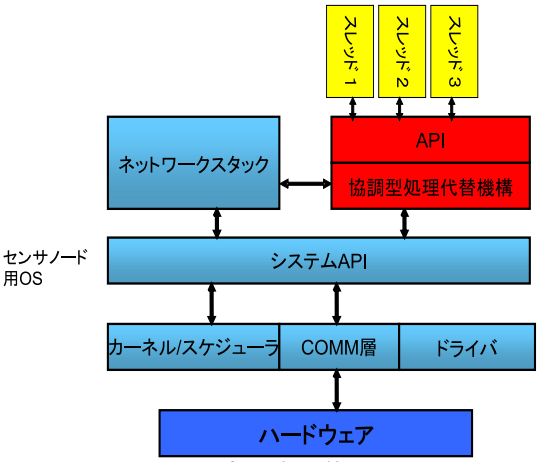


図 1 本研究の位置づけ

2.2 提案手法の前提条件

本手法では, 次の条件を前提として, 機構の設計を行う.

- すべてのノードが同じ能力・タスクを持つ.
- ノードの位置情報は既知である.
- 無線モジュールは常にオンとする.
- グループ分けは最初のみ行う

3 仮想グループの構成

この章では, 仮想グループの分け方について述べる. 本研究では, GAF(Geographical Adaptive Fidelity)[2] というアルゴリズムを使用して仮想グループを決定する. この手法は次の 2 段階に分けて行われる.

3.1 第 1 段階：仮想グループの大きさの決定

第 1 段階は仮想グループの大きさの決定である. 代替応答処理を行う際, グループにあるすべてのノード間及び隣接グループ間での通信が必要となるため, グループの大きさは隣接グループ内のすべてのノードが互いに通信可能という条件を満たさなければならない. したがって, 図 2 に示すように, ノードの最大通信距離を R とすると, 隣接グループのもっとも遠い 2 点間の距離は R 以下である必要がある. ピタゴラスの定理で計算できるように, グループの 1 辺の長さ r は $r \leq R/\sqrt{5}$ を満たす必要がある. したがって, 1 つのグループの最大面積は $R^2/5$ となる.

3.2 第 2 段階：マスタノードの選定

仮想グループを決定する第 2 のステップでは, 仮想グループにあるすべてのセンサノードを管理するマスタ

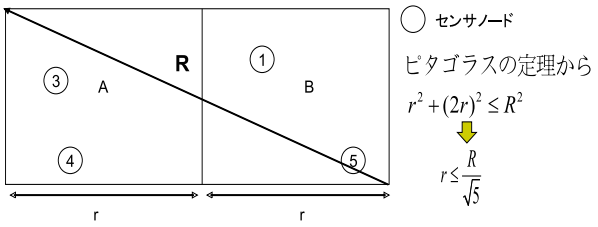


図 2　仮想グループの大きさ

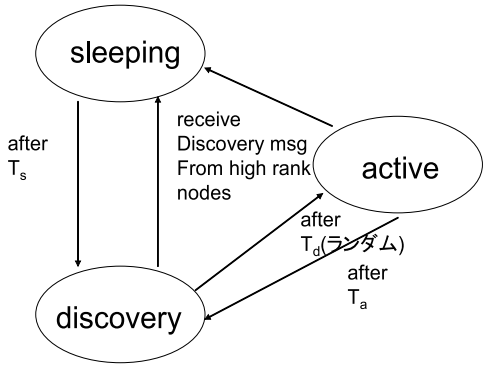


図 3　仮想グループにあるノードの状態遷移

ノードを選定する. すべてのノードの電力を均等に消費させるため, ノードは周期的にスリープ状態とアクティブ状態に入らなければならない. 各ノードはスリープ状態からアクティブ状態に切り替える際, 3 つの状態から遷移する. 図 3 に示すように, それぞれは discovery, active, sleeping である.

仮想グループは最初に形成する際, すべてのノードは自身の位置や ID などの情報をグループ内にあるほかのノードにブロードキャストする. これにより, 各ノードはグループ内にあるほかのノードの情報を知ることが可能となる. それから, 自身のタイマにランダムな値 T_d を設定する. T_d は自分自身が discovery 状態となる時間を意味する. ノードは自身のタイムが T_d を超えるとき, 自分がマスタノードとなり, その情報をほかのノードにブロードキャストする. ノードはタイマが T_d になる前にほかのノードがマスタノードになる情報を受信する場合, 自分自身をスリープ状態にし, タイマに値 T_s を設定する. T_s は自分自身のスリープ時間を意味する. マスタノードは自身のタイマに値 T_a に設定する. T_a はマスタノードの活動時間である. タイマが T_a になる前に, マスタノードは定期的にほかのノードにブロードキャストし, ほかのノードがマスタノードになることを防ぐ. スリープ状態のノードのタイマが T_s を超える場合, 再び discovery 状態になる. マスタノードあるいは discovery 状態になるノードはグループ内に自身よりマスタノードにふさわしいノードを発見した場合, 自身をスリープ状態にする.

4　システム構成

本システムでは, 4 つの機構から構成される。それぞれはシステム API, RPC 機構, ノード遷移管理機構, タスク管理機構である. 以下, それぞれについて述べる.

- API:提案システムを簡単に制御するためのインタフェースである.
- RPC 機構:マスタノードからスレーブノード上に存在するタスクを呼び出すために使用される. 情報はマスタノードからスレーブノードへパラメータとして転送され, 手続きの結果として戻る.
- ノード遷移管理機構：この機構は, マスタノードが遷移する際, データの統合やマスタノードの選定を行う. また, ノード遷移中にほかのグループからの要求に応答できないため, 遷移の開始と終了後にほかのグループに情報を送信する.
- タスク管理機構：この機構は, タスクの優先度に基づいて代替応答処理を行うかどうかを決定する. 優先度を決める際, 相対デットラインが小さいタスクを先に実行させるため, 相対デットラインが小さいものにより高い優先度をつける. そのため, 本手法では, デッドラインモノトニックスケジューリングを使用する.

5　おわりに

本稿では, 複数のセンサノードを 1 つのグループにまとめ, 互いに代替応答処理を行うことによって, 全体処理時間の短縮および安定性を向上させる方法について述べた. 今後は MANTIS OS[3] を用いてこれらの機能の実装し, 評価方法を検討する予定である.

参考文献

[1] Murat Demirbas, Onur Soysal, Muzammil Hussain, ”TRANSACT: A Transactional Framework for Programming Wireless Sensor/Actor Networks ”, pp. 295-306, IPSN 2008.

[2] Ya Xu, et al:”Geograhy-informed Energy Conservation for Ad Hoc Routing,” in proc. of MobiCom’01, pp.70-84, 2001

[3] S. Bhatti, et al, ”MANTIS OS: An Embedded Multithreaded Operating System for Wireless Micro Sensor Platforms, ” ACM/Kluwer Mobile Networks & Applications, Special Issue on Wireless Sensor Networks, vol. 10, no. 4, pp. 563-579, August 2005.

センサネットワーク向けOSにおける協調型処理代替機構

立命館大学　理工学研究科
基本ソフトウェア研究室
李 冉

1

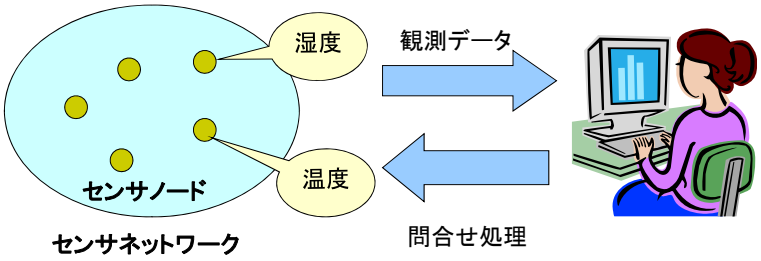
発表内容

- はじめに
  - 研究背景
  - センサネットワークとは
  - 既存のセンサネットワークの問題点
- 研究概要
- 提案システムの構成
  - API
  - ノード遷移管理機構
  - RPC機構
  - タスク管理機構
- おわりに

2

はじめに(1)

- 無線センサネットワーク(WSN)とは
  - センサノードと呼ばれる無線通信機能を持つデバイスに構成される自律的なネットワーク
  - 周囲の環境情報などを取得し、マルチホップ通信によって基地局にデータを送信
  - 低コスト、低電力消費、配置しやすいという特徴を持つ



3

はじめに(2)

- WSNの特徴を利用し、普通のネットワークに代わり、様々なサービスを同時に実行可能
  - 商品検索、火災検知、マルチロボット制御、etc
- 複数のサービスの実行における問題点
  - 単一ノード上に複数のタスクが存在
    - センサノードには低クロックのMPUを採用することが多い
    - MPUと無線モジュールが常にビジー状態になり、応答性が低下
    - サービスが破たん
  - WSNはマルチホップで通信するため、センサノードの故障によりシステム全体が通信不可能になる可能性がある
    - システムの安定性が低下

4

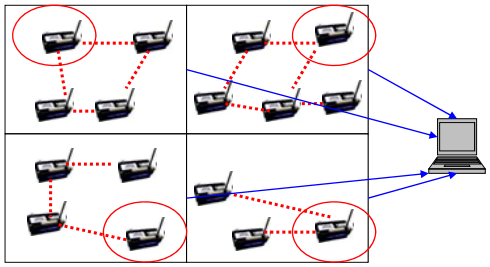
研究目的、ゴール及び実現方法

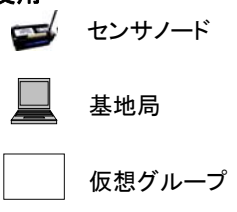
- 研究目的
  - 1つのセンサネットワークで複数のサービスを同時に実行する際、MPUや無線モジュールの処理速度による応答性の低下やサービスの破たん問題を解決すること
- 研究ゴール
  - 上記の目的を果たすフレームワークをユーザに提供する
- 実現方法
  - 複数のノードを使用して仮想グループを構築する
  - 1つのグループを1つのセンサノードと見なし、センサネットワークを構築する
  - グループ内にあるノードは互いに協調し、代理応答処理による負荷分散を行う

5

提案手法の概要

- **複数のセンサノードを用い、グループにまとめて仮想のネットワークを構成する**
- **グループ内にアクティブ状態となるノード1つをマスタノードとし、ほかのノードをスレーブノードとし、スリープさせる**
- **マスタノードはグループ内にあるすべてのノード情報を管理する**
- **同一グループにあるセンサノードを用いて、サービスの代替応答処理を行う**
- **マスタノードはある周期で動的に切り替える**
- **仮想グループを構築するには、GAFアルゴリズムを使用**





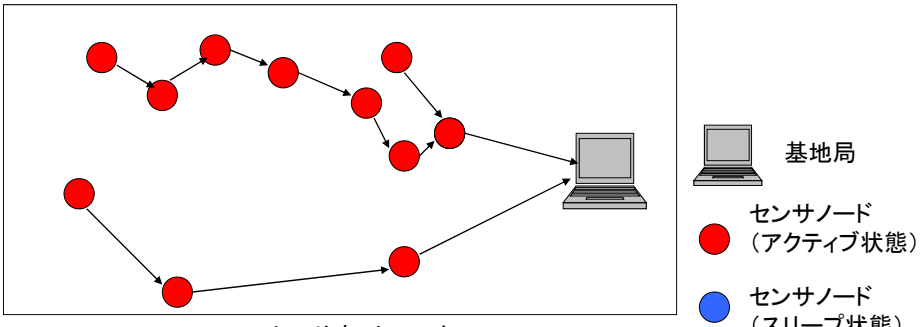
6

提案手法の前提条件

- すべてのノードが同じ電力・タスクを持つ
- ノードの位置情報は既知である
- 無線モジュールは常にON
- グループ分けは最初のみ行う
  - ノードは電力が使い切るまで1つのグループに属する

GAFの概要

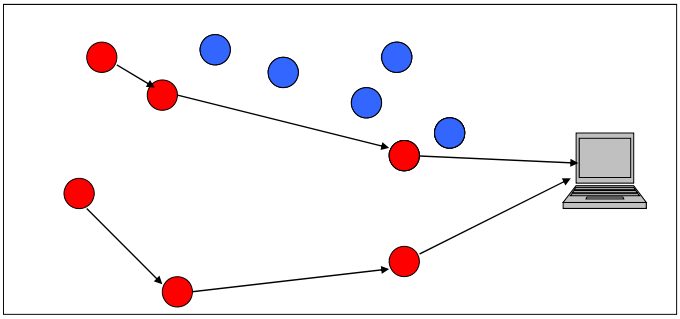
- GAF(Geographical Adaptive Fidelity)とはノードの位置をベースにし、不要なノードを停止による省電力ルーティングアルゴリズム
- 本研究では、GAFのグループ分け部分のみを使用



センサネットワーク

GAFの概要

- GAF(Geographical Adaptive Fidelity)とはノードの位置をベースにし、不要なノードを停止による省電力ルーティングアルゴリズム
- 本研究では、GAFのグループ分け部分のみを使用



センサネットワーク

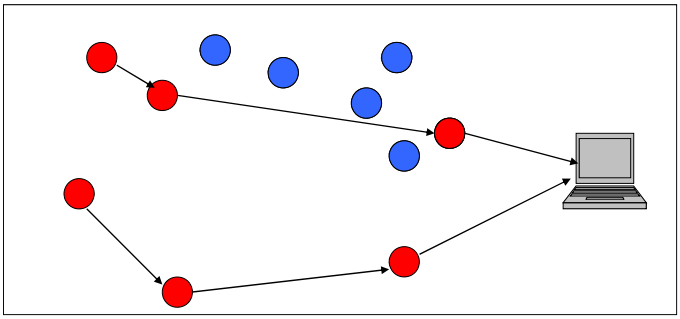
基地局

センサノード
（アクティブ状態）

センサノード
（スリープ状態）

GAFの概要

- GAF(Geographical Adaptive Fidelity)とはノードの位置をベースにし、不要なノードを停止による省電力ルーティングアルゴリズム
- 本研究では、GAFのグループ分け部分のみを使用



センサネットワーク

基地局

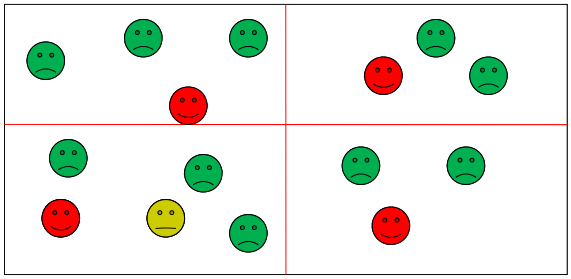
センサノード
（アクティブ状態）

センサノード
（スリープ状態）

GAFの概要(2)

- センサネットワーク内のセンサノードを動的にグループ化することが可能
- ノードを高密度に配置することにより，システム全体の動作時間を延ばす
- アクティブ状態となるノードはグループ内に遷移できる
- 本研究では、GAFのグループ分け部分のみを使用

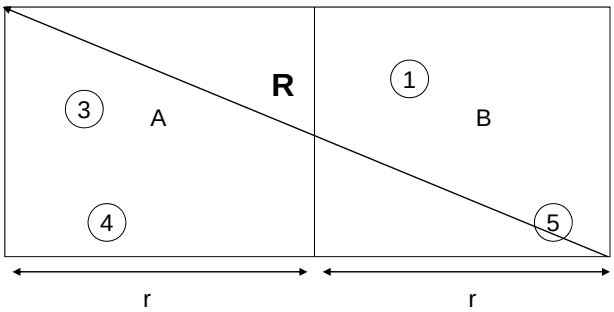
センサネットワーク



マスタノード
スレーブノード
使えないノード

仮想グループの構築（GAF）の第一段階：仮想グループの大きさの決定

- 1つのグループを仮想的に1つのセンサノードと見なすため、隣接のグループにあるノードが互いに通信できる必要がある



センサノード

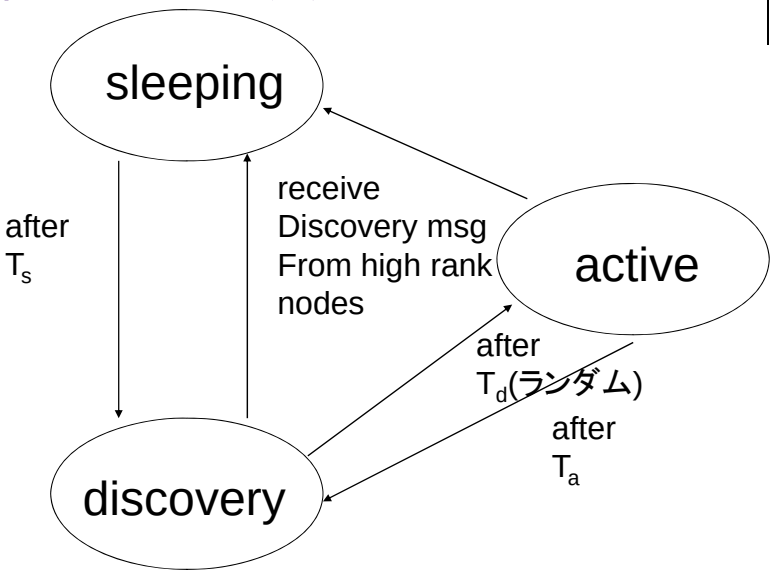
ピタゴラスの定理から

$$r^2 + (2r)^2 \le R^2$$

$$r \le \frac{R}{\sqrt{5}}$$

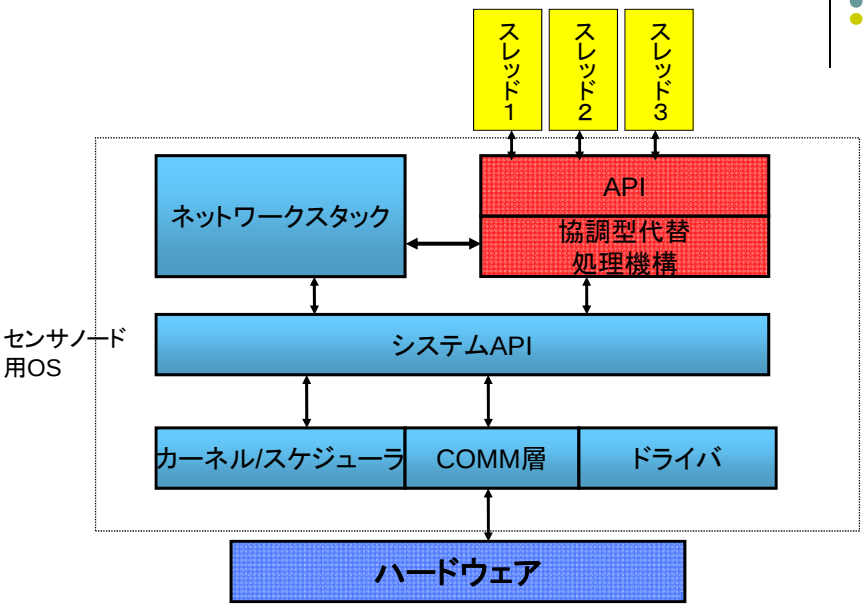
12

仮想グループの構築(GAF)の第二段階:マスタノードの選定



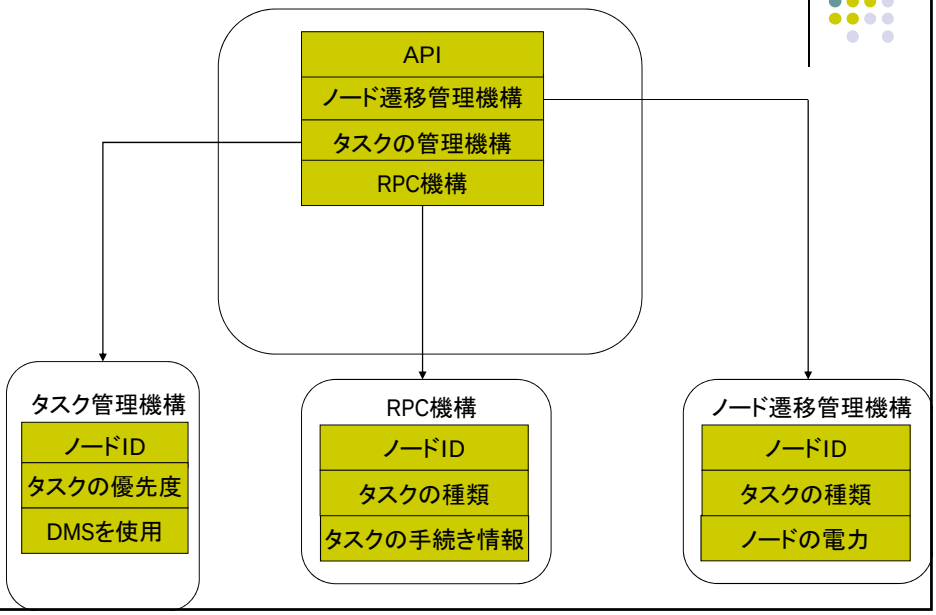
13

提案する機構の位置づけ



14

システム構成(1)



システム構成(2)

- API
  - 提案するシステムを簡単に制御するためのインタフェース
  - 例:uint8 get_master_info(unit16 G_NUMBER, struct node_i node_info)
    uint8 change_priority(unit16 G_NUMBER, unit8 priority)
- RPC機構
  - マスタノードからスレーブノード上に存在するタスクを呼び出すために使用される
  - 情報はマスタノードからスレーブノードへパラメータで転送され、手続きの結果として戻る

16

システム構成(3)

- ノード遷移管理機構
  - 仮想グループの分け
  - ノードの加入と離脱
  - マスタノードの遷移による データの統合
  - グループ内にあるすべての ノード情報を管理
    - 情報管理テーブルを作成
    - 情報管理テーブルに基いて マスタノードを選定
  - マスタノード遷移の開始と終了後にほかのクループに情報を送信

ノード番号	電圧	タスクの種類	そのほか
1	2.7	データの送信	
2	3.0	モニタリング	
3	2.5	データの圧縮	

17

システム構成(4)

- タスク管理機構
  - タスクの優先度に基づいて代替応答処理を行うかどうかを決定する
  - デッドラインモノトニックスケジューリング(DMS)を使用して優先度を決定する

実装

- ハードウェア
  - Crossbow社のMICAzを使用
    - 7.37MHzのMPU
    - 4kbのSRAM
- OS
  - MANTIS OS(MOS)
    - カーネルサイズ＜500B
    - C言語で記述可能
- 現在の実装状況
  - ノード遷移管理機構をほぼ実装した
  - RPC機構は実装中

19

おわりに

- 協調型処理代替機構の概要と構成
  - 複数のセンサノードを使って仮想グループを構成する
  - 同一グループにあるノードの協調代替応答処理によってシステムの全体の待ち時間を短縮し、安定性を向上する
- 今後の予定
  - 提案するフレームワークの実装を継続
  - 評価方法を検討

20

匿名通信路のノード離脱に対する通信路継続方式

石黒　聖久 † 田中　寛之 † 近藤　正基 † 齋藤　彰一 † 松尾　啓志 †
† 名古屋工業大学

1　はじめに

現在, 通信内容を暗号化によって保護することは広く行われている. しかし通信の匿名性は完全ではない. この状況下で匿名性が重要視される通信, 例えば重要なビジネス情報の通信や電子投票の実装を行うのは危険である. この様な通信を行う為には匿名通信を行うための通信路が必要となる.

現在, 代表的な匿名通信手法の一つに Onion Routing[1] がある.Onion Routing では複数の中継ノードを介し, データを多重暗号化することで匿名通信を行う. しかしこの手法には, 通信路を構成する中継ノードが離脱した場合, 中継ノード同士による通信路の部分修復が不可能という欠点を伴う. 本研究ではこの Onion Routing のための新しい匿名通信路修復方式を提案する.

以下第 2 章では既存の匿名通信手法である Onion Routing と Tor[2] について説明し, その手法の持つ問題点を挙げる. 第 3 章では本稿の提案方式を説明する. 第 4 章では Tor と提案方式を比較し, 第 5 章でまとめる.

2　既存手法

2.1　Onion Routing

既存の匿名通信手法である Onion Routing は, ネットワーク上に複数配置された中継ノードで構成される. 送信者が受信者に対してデータを送信する際, 送信データを中継ノードの公開鍵を用いて多重に暗号化する. 各中継ノードは受信した暗号メッセージを復号し, 次ノードの情報を得る. これを受信者までの全中継ノードで繰り返し行う. 以上の中継処理によって, 送信者と受信者が中継ノードと区別出来なくなり, 匿名性を得る. また中継ノードは自身の前後の経路上ノードしか把握出来無いため, 中継ノードが一つでも信頼出来れば, 匿名性は維持される.

Onion Routing の問題点として, 各中継ノードの所持する経路情報が少ないために中継ノード同士による経路の部分修復が困難という欠点がある. 送信者が離脱を検知出来ない限り, 経路の切断に対応することが出来ない. なぜなら送信者以外で修復に必要な経路情報を持つノードは存在しないからである.

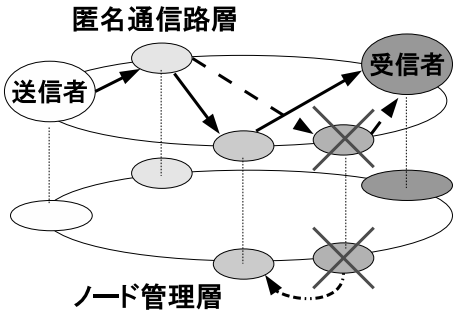


図 1: 全体構成

2.2　Tor

Tor は Onion Routing を用いて実装された匿名通信システムである. Tor は送信者は受信者との間に複数の匿名通信路を定期的に生成する. 送信者はその経路の内, 最も新しく生成された経路を用いて通信を行う. 送信者は定期的に使用する経路を切り替える. 切り替えの間隔はデフォルトで 1 分である.

ノードの離脱が発生した経路では, 前述した Onion Routing と同様に以後の通信を行うことは出来ない. しかし経路の切断が発生した後に, 送信者が経路の切り替えることで, 再び通信可能となる.Tor による経路の切り替えは定期的に行われ, 送信者がノードの離脱を検知できない場合でも通信の再開が可能である.

しかしこの手法では送信者は定期的に経路の生成を行う必要がある. 経路の生成にはメッセージの多重暗号化, 多段中継と複数の復号処理を伴うため, 生成に必要なメッセージ数と, 送信者および中継ノードに掛かる負荷が大きい.

3　提案手法

提案方式では通信路生成時に経路を構成する各中継ノードが, 自身の離脱に備えたノードを選出する. このノードを**代理ノード**と呼ぶ. 代理ノードは選出元の中継ノードと同じ経路情報を保有し, 経路の切断時に離脱した元の中継ノードに代わり通信を復旧する.

しかし, 代理ノードの選出や代理ノードへの経路情

A Method for Recovery of Anonymous Communication from Node Secession

† Kiyohisa Ishiguro
† Hiroyuki Tanaka
† Masaki Kondo
† Shoichi Saito
† Hiroshi Matsuo
Nagoya Institute of Technology (†)
Gokiso-cho, Showa-ku, Nagoya, Aichi, 466-8555 Japan

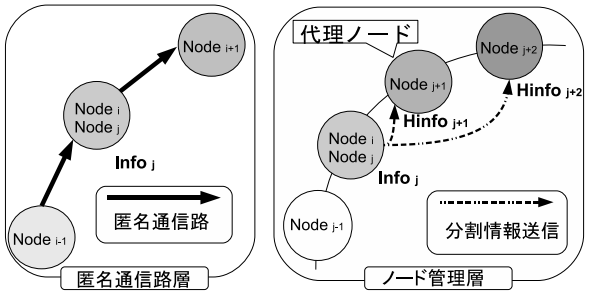


図 2: 代理ノード選出

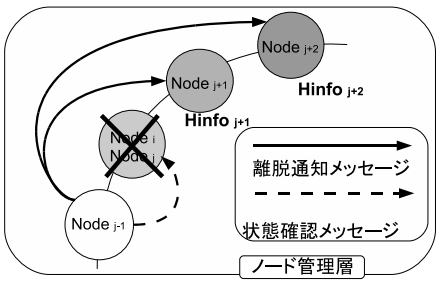


図 3: ノード離脱検知

報の保管, 離脱検知にはノード間で通信を行う必要がある. このためには, 各ノードは他の参加ノードの状態を保持する必要がある. しかし, 各ノードが独立してノード状態の保持を行うことは一般に困難である.

そのため提案方式では匿名通信を行う層とノード管理を行う層の二つの層に分離する. 以下, それぞれを匿名通信路層, ノード管理層と呼ぶ. 全体図を図 1 に示す. 匿名通信路層の実線矢印が Onion Routing による通信可能な匿名通信路を表し, 破線矢印がノード離脱により使用不可能な匿名通信路を表す. ノード管理層における破線矢印は, 離脱した中継ノードの代わりに代理ノードが中継ノードとなる様子を表す.

匿名通信路層では Onion Routing による匿名通信を行う. この層では匿名通信以外の処理を行わない. ノード管理層では Chord[3] を用いてノードの管理を行う. ノード管理層では Chord による検索機能, 安定化処理による離脱ノードの検知機能を用いて, 代理ノードの選出とノードの離脱検知を実現する.

提案方式では Tor のように送信者が定期的に経路を生成する必要も, 経路の切り替えを行う必要もない. このため Tor に比べて低コストでの経路の維持ができる.

3.1 匿名通信路生成時の処理

匿名通信路生成の際, 送信者は任意の数の中継者を選択し, 匿名通信路生成メッセージを生成する. このメッセージの生成手順は Onion Routing に準ずる. 送信者は生成したメッセージを中継者ノードを経由して受信者に送信する.

このときメッセージを受信した中継ノード $Node_j$ は,Onion Routing の手順に従って以後の通信に使用する経路情報 $Info_j$ を取得し, 次のノードにメッセージの送信を行う (図 2 左参照).

ノード管理層 (図 2 右参照) では, 自身の代理ノードの選出処理と経路情報の保管処理を行う. ノード $Node_j$ は自身の Successor である $Node_{j+1}$ を代理ノードとして選出する. 代理ノードには修復に必要な経路情報

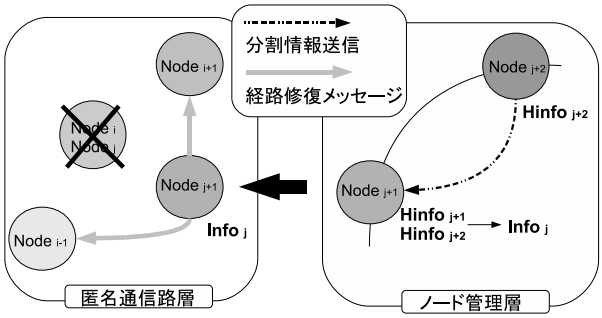


図 4: 通信路修復処理

$Info_j$ を保管する必要があるが, 安全性確保のために分割して別々のノードに保管する. 具体的には $Info_j$ を分割経路情報 $Hinfo_{j+1}$ と $Hinfo_{j+2}$ に分割する. 分割経路情報は $Hinfo_{j+1}$ と $Hinfo_{j+2}$ を結合しない限り, 経路情報として利用できない.$Node_j$ は $Hinfo_{j+1}$ を代理ノードへ,$Hinfo_{j+2}$ を代理ノードの Successor である $Node_{j+2}$（以下**保管ノード**とする）に送信する.

3.2 通信路修復処理

Chord は定期的にノードの状態を確認する安定化処理を行う. 安定化処理では各ノードは自身の Successor に状態確認メッセージを送信する. このとき Successor が離脱していた場合, 離脱したノードの Successor を自身の Successor に変更する. 図 3 の例では,$Node_{j-1}$ が $Node_j$ に状態確認メッセージを送信する. この時 $Node_j$ が離脱していた場合,$Node_{j-1}$ は $Node_j$ の離脱を検知し, 代理ノード $Node_{j+1}$ と保管ノード $Node_{j+2}$ に $Node_j$ の離脱を通知する.

離脱検知後の処理手順を図 4 に示す. ノード管理層 (図 4 右参照) で通知を受けた保管ノードは, 代理ノードに自身の保持する分した割経路情報を送信する. 分割した経路情報を受信した代理ノードは, それを自身の保持する分割した経路情報と結合し, 経路情報を取得する. 代理ノードは経路情報によって, 匿名通信路層 (図 4 左参照) での前後のノードの IP アドレスを取

表 1: Tor との通信路復旧コストとの比較

	通信路修復 メッセージ数	通信再開 最長時間
Tor	$x^2 + x$	1 分
提案方式	6	55 秒

注) x は 中継ノード数+受信ノード数 を示す

得する. 代理ノードはこれらのノードに経路修復メッセージを送信し, 通信路の修復を行う.

4 評価

評価として Tor の通信路修復手法との比較を行う. 比較は, 通信路修復に必要なメッセージ数と, 通信路修復に要する最長時間である.

メッセージ数について述べる.Tor は複数の通信路を生成し, 定期的に切り替えを行うことで通信を維持する. 通信路の生成に必要なメッセージ数は中継ノードの数に比例して大きくなる (表 1 参照). 対して提案方式では Chord による離脱検知に必要なメッセージと, 検知後の匿名通信路の修復に必要なメッセージが必要となる. これらの修復メッセージ数は通信路の中継ノード数に依存しない.(表 1 参照).

通信復旧時間について述べる.Tor の通信復旧時間は, ノードの離脱検出機能がないため, 通信路切り替え間隔に依存する. 文献 [2] によると, 通信路切り替え間隔は 1 分である.1 分より短いと, 経路生成に必要な時間が切り替え間隔を上回る場合があるため,1 分未満での切り替えは困難である. 提案方式では,Chord の安定化処理間隔に依存する. 文献 [3] によると安定化処理間隔は 30 秒である. 以上より, 通信復旧は提案方式が短時間に行うことが可能である.

本稿では提案方式を OverlayWeaver[4] 上に実装し, 通信復旧時間を測定した. 測定環境は,CPU Sempron 2800+, メモリ 1GB, ネットワーク速度 100Mbps の LAN 環境で, 全 4 ノードによる匿名通信路である.OverlayWeaver の Chord 実装における安定化処理間隔は最長 64 秒であるため, 測定結果は最長 55 秒, 最短 2 秒となり平均 28 秒であった. 安定化処理間隔（64 秒）を考慮すると, この結果は適当である. このように実装依存となるため, 今後匿名通信路用に最適化した安定化処理間隔を求めることが課題である.

5 まとめ

本稿では, 匿名通信路のノード離脱による通信路切断の復旧手法として, 代理ノードによる通信路の部分修復手法を提案した. 提案方式ではノード管理層を用いて, 代理ノード選出と経路情報保管を可能にした.

匿名通信路生成の際, 各中継ノードはノード管理層で代理ノードの選出し, 通信路の修復に必要な経路情報を保管する. ノードの離脱が発生した場合, 代理ノードはノード管理層で離脱を検知する. 離脱を検知した代理ノードは保持する経路情報を用いて, 離脱したノードの役割を引き継ぎ, 通信路を部分修復する.

既存手法の Tor との比較した結果, 通信路全体の生成が必要な Tor に対し, 部分修復を行う提案方式の方が低コストでの通信再開が可能である. 今後は Tor の通信の復旧に必要な時間の計測を行い, 提案方式とのより詳細な比較を行う.

参考文献

[1] Goldschang, D, Reed, M. and Syverson, P.: Onion routing for anonymous and private internet connections, Comm. ACM, Vol.42,No.2,pp.39-41

[2] Dingledine, R. and Mathewson, N.: Tor: The Second-Generation Onion Router, Proceedings of 13th USENIX Security Symposium, pp. 303-320 (2004).

[3] Stoica, I.,Morris, R., Karger, D., Kaashoek, F. and Balakrishnan, H.: Chord : A Scalable Peer-To-Peer Lookup Service for Internet Applications, Proc. 2001 ACM SIGCOMM Conference, pp. 149-160(2001).

[4] 首藤一幸, 田中良夫, 関口智嗣: オーバーレイ構築ツールキット OverlayWeaver, 情報処理学会論文誌: コンピューティングシステム, Vol.47, No. SIG12(ACS 15), pp. 358-367 (2006).

匿名通信路のノード離脱に対する通信路継続方式

名古屋工業大学　齋藤研究室
石黒　聖久

1

背景

- 通信内容の守秘は広く行われている
  - 「通信を行っている」という事実を保護出来ない
- 重要なビジネス情報の通信、電子投票などを行うのは危険
  - 通信を行っていることすら特定されてはいけない

匿名通信が必要

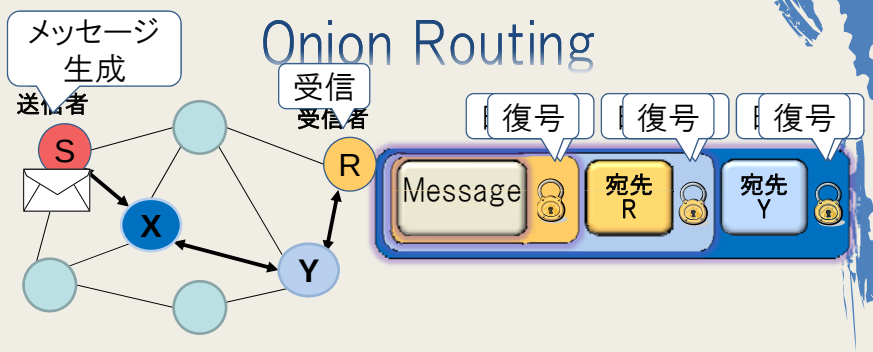
2

既存手法

- Onion Routing
  - 中継ノードと多重暗号を用いる匿名通信アルゴリズム
- Tor(The Onion Router)
  - Onion Routingで実装された匿名通信システム
    - 現在もっとも使用されている匿名通信システム
    - TorはユーザにOnion Routingを行う中継ノードを提供する

3

Onion Routing

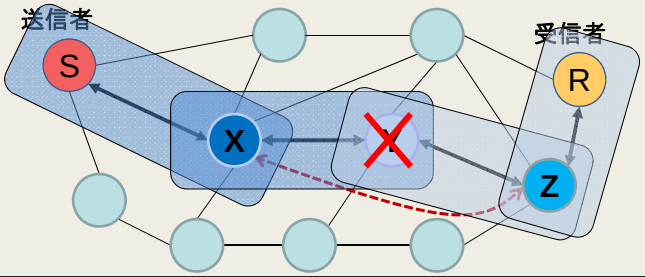


- 送信者はデータを送信する際に、各中継ノードの公開鍵で多重に暗号化する
- 受信ノードは自身の秘密鍵でデータを復号する
- 中継者であれば次の宛先のノードにデータを転送する

4

Onion Routingの欠点

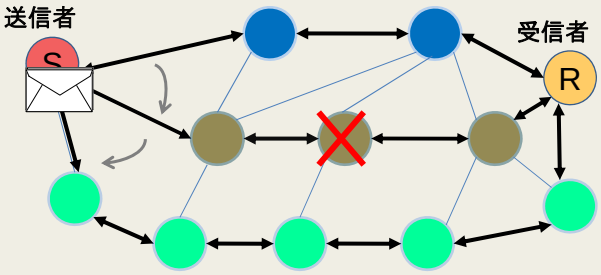
- 通信速度の低下
  - 多段中継、多重暗号が必要なため
- 中継ノードの離脱による通信の切断
  - 中継ノード同士による修復が出来ない



5

Torの経路切り替え方式

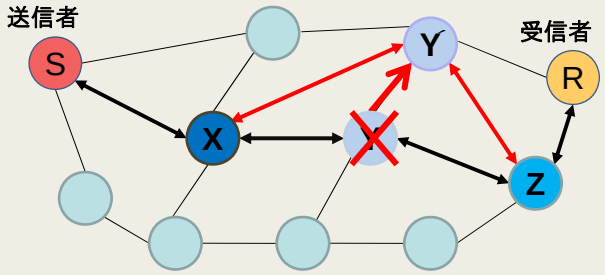
- 1分毎に経路の切り替えを行う
  - 毎回の経路生成のコストが高い
- ノードが離脱した場合、経路の切り替えが起こるまで通信が出来ない



6

提案手法

- ノード離脱による通信切断時に経路の部分修復を行う
  - 予め選出した代理ノードが修復を行う
- 通信の再開に経路全体の再生成が必要でない



7

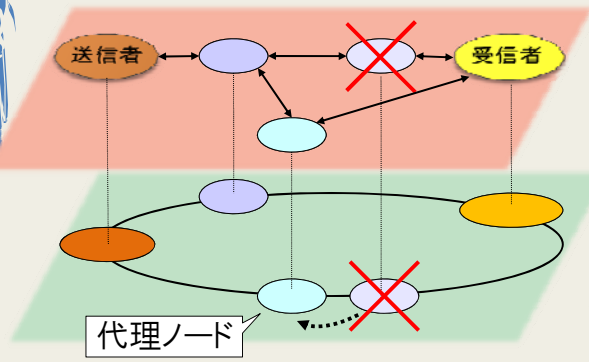
検討すべき事柄

- 代理ノードの選出
- 経路の修復に必要な経路情報の引き渡し
  - 通信用暗号鍵
  - 匿名通信路における前後ノードのIPアドレス
- ノードの離脱の検知

8

提案システム全体図

匿名通信路層

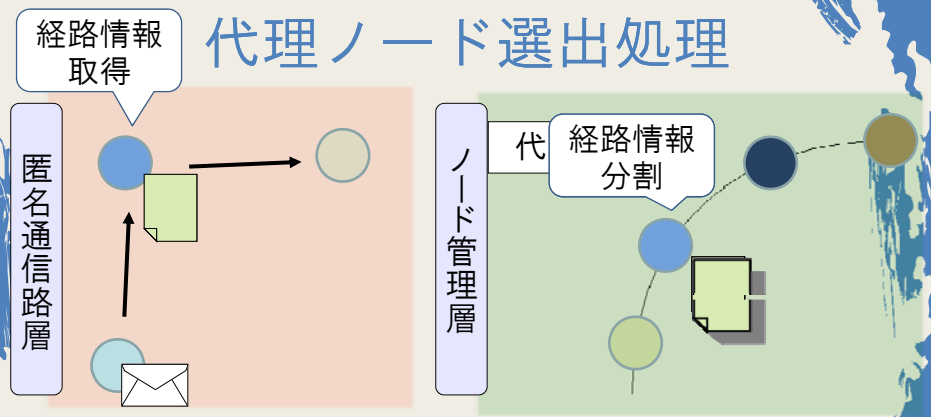


ノード管理層

- OnionRoutingによる匿名通信
- 各ノードは自身の前後の経路情報のみ持つ
- Chordによるノード探索、安定化処理
- ノード離脱の検知
- 代理ノードの検索

9

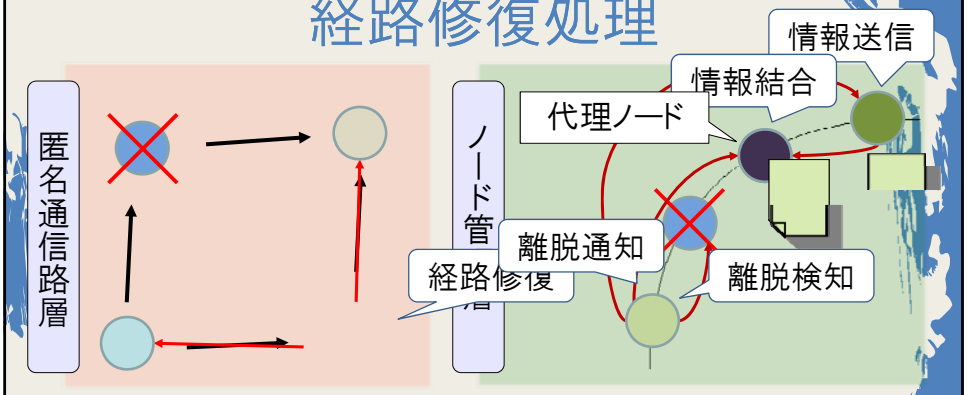
代理ノード選出処理



- 各ノードは自身の複数のSuccessorのIPアドレスを持つ
  - 代理ノード、そのSuccessorと1メッセージで通信出来る

10

経路修復処理



- Chordでは定期的に安定化処理を行う
  - この時Successorに状態確認メッセージを送信する

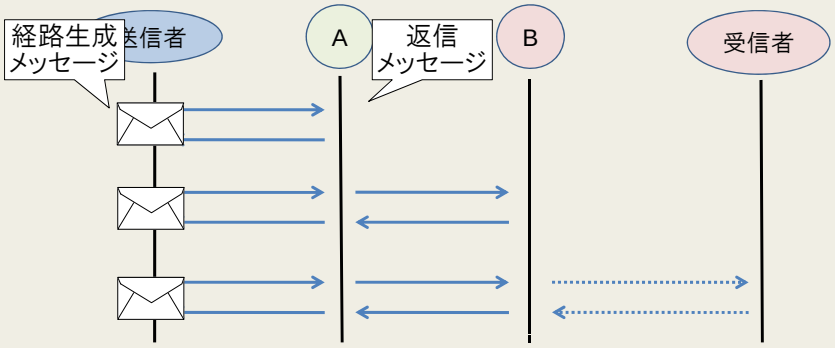
11

Torとの比較

	Tor	提案手法
経路修復に必要なメッセージ数	経路生成メッセージ数	経路修復メッセージ数
通信再開に必要な時間	経路切り替え間隔	実値の計測

12

Torによる匿名経路生成コスト



- 送信者は2～20個の中継ノードを選択する

必要メッセージ数

x：中継ノード数 ＋ 受信ノード

$$\sum_{k=0}^{x} 2k = x^2 + x$$

13

Torとの比較

	Tor	提案手法
経路修復に必要なメッセージ数	経路生成メッセージ数	経路修復メッセージ数
通信再開に必要な時間	経路切り替え間隔	実値の計測

- X=中継ノード(2～20)+受信ノード数

14

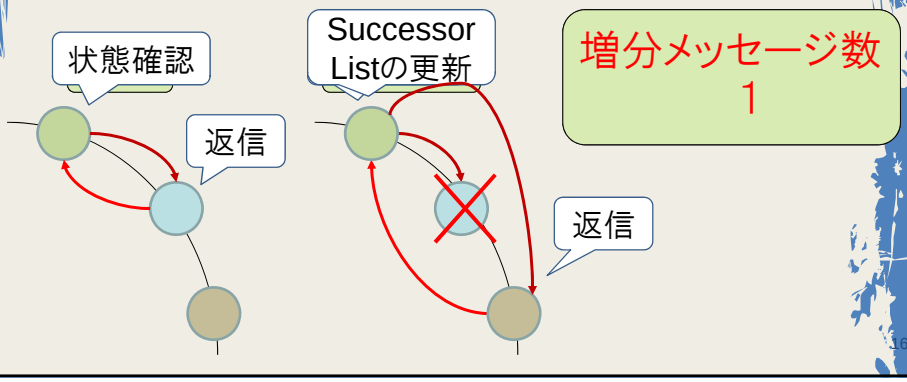
提案手法による経路修復コスト

- ノード管理層の経路修復に必要なメッセージ数
- 匿名通信路層の修復に必要なメッセージ数

15

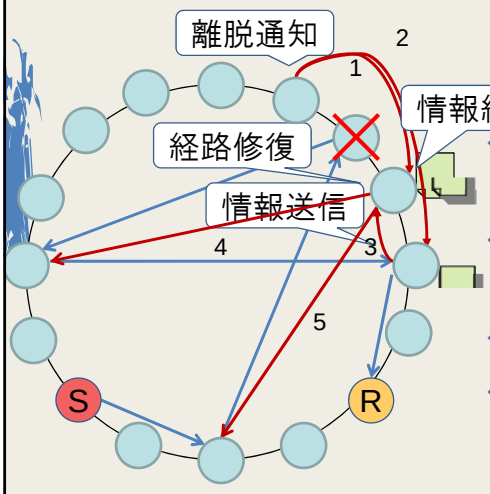
ノード管理層の修復コスト

- Chordの安定化処理によるメッセージの増分をコストとする



16

匿名通信路層の修復コスト



- ノードの離脱を検知したノードは離脱を通知する
- 代理ノードのSuccessorは分割情報の送信を行う
- 分割情報を結合する
- 匿名通信路の修復を行う

必要メッセージ数 5

17

経路修復に必要なメッセージ数

- Torの経路全体の生成コスト

$$\sum_{k=0}^{x} 2k = x^2 + x$$

  - x=中継ノード(2～20) + 受信ノード 1
  - x=3 の時、メッセージ数は12
- 提案手法での経路の部分修復コスト

Chordの修復コスト 1 + 経路部分修復コスト 5

  - 中継ノード数に関係なく6で一定

Torよりも少ないコストで修復可能

18

Torとの比較

	Tor	提案手法
経路修復に必要なメッセージ数	$x^2 + x$	経路修復メッセージ数 6
通信再開に必要な時間	経路切り替え間隔	実値の計測

- X=中継ノード(2～20)+受信ノード数
- Torの経路切り替え間隔はデフォルトで1分

19

実値の計測

- ノード管理層にはオーバレイ構築ツールであるOverlayWeaverを使用
- 4ノードから成る匿名通信路
  - Sempron 2800+, メモリ 1GB
  - 100Mbpsのネットワーク
- 中継ノード一台の機能を停止する

最大遅延	55(s)
最小遅延	2(s)
平均遅延	28(s)

20

Torとの比較

	Tor	提案手法
経路修復に必要なメッセージ数	$X^2 + X$	6
通信再開に必要な時間	最大1分必要	最大55秒（ただし可変）

- X=中継ノード(2～20)+受信ノード数
- Torの経路切り替え間隔はデフォルトで1分

21

まとめ

- 代理ノードにより匿名通信路を部分修復
- 匿名通信路層とノード管理層で分離
  - 匿名通信と検索、修復を両立可能
- 保管する経路情報は安全性のために分割
- 安定化処理時にノードの離脱を検知
  - ノードの離脱を代理ノードに通知
- 経路全体の再生成より低コストでの部分修復が可能

22

通信用公開鍵の配布機能とメッセージの並列送信機能を有した匿名通信方式

田中 寛之 † 石黒 聖久 † 近藤 正基 † 齋藤 彰一 † 松尾 啓志
† 名古屋工業大学

1 はじめに

インターネットでの通信は暗号化によって秘密にすることができるが，時間と IP アドレスから個人の特定が可能であり，送受信者が誰かということは隠せない．この問題を解決するために匿名通信方式が必要とされている．

匿名通信は，送信者が特定できないこと，受信者が特定できないこと，送受信者間を追跡できないことの 3 つの要件を満たす必要がある [1]．以下，これらをまとめて**匿名性**と言い，これら匿名性を備えた通信を**匿名通信**，匿名通信が使用する通信路を**匿名通信路**と言う．

本稿では，通信用公開鍵の配布機能と並列送信機能を有する匿名通信方式を提案する．提案方式は，これらの匿名性を実現するために多重暗号化を用い，送受信者の情報 (ID や IP アドレス，経路情報) を隠蔽する．提案方式の特徴として，通信用公開鍵の配布機能を持つため，通信用公開鍵サーバが無い．さらに，並列送信機能によって通信速度の向上を図っている．**スーパーノード**(**Super Node: SN**) を導入することで，この 2 つの機能を実現した．

本稿では，2 章で既存方式について述べ，3 章で提案方式の詳細と動作について述べる．そして 4 章で評価を行い，最後に 5 章でまとめる．

2 既存方式

我々の研究室で提案した既存方式 [3] がある．既存方式はノード管理層と匿名通信路層で構成される．匿名通信とノード管理を分離することで，システムは匿名性を考慮することなく容易にノード管理を行える．ノード管理層では DHT である Chord を用いて，ノードの参加離脱処理と検索処理と公開鍵の配布処理を行う．匿名通信路層は多重暗号を用いて，匿名通信路の構築，メッセージの生成と暗号化，メッセージの送受信を行う．

図 1 に既存方式の動作を示す．既存方式では，メッセージは受信エリアと呼ばれる連続した ID 空間を経由する．図 1 の破線で囲まれた領域が受信エリアである．

中継の流れは，まず，送信者が匿名通信路を構成するノード (中継ノード) を選択する．経路決定後，(1) 送信者は最初の受信エリアの始点ノードを検索し，(2) 送信する．送信者からのメッセージは各受信エリアのすべてのノードを経由後，受信エリア終端ノードによってメッセージが復号される．(3) 次の宛先を検索し，(4) 次の受信エリアへメッセージを送信する．受信者は通信経路中の任意の位置に配置される．

3 提案方式

提案方式は，既存方式を改良した方式で，通信用公開鍵の配布機能とメッセージの並列送信機能を持つ．実装はオーバーレイ構築ツールキットである Overlay Weaver [4] を基盤に行った．なお，公開鍵暗号には RSA を用い，共有鍵暗号には AES を用いる．

3.1 スーパーノード

既存方式ではすべてのノードが通信用公開鍵を持つのに対し，提案方式では通信用公開鍵を持つノードを一部のノードに限定している．提案方式の，通信用公開鍵を持つノードをスーパーノードと呼び，通信用公開鍵を持たないノードを通常ノード (Regular Node: RN) と呼ぶ．

スーパーノードは，参加ノードの数に対して一定の割合存在し，ノードがシステムに参加する時に役割が与えられる．また，ノード数が減った場合にはスーパーノードを通常ノードに戻して，数を減らす．

スーパーノードは通信用公開鍵の配布，メッセージの復号，メッセージの並列送信を行う．スーパーノードの数は通常ノードの数に比べて少なく，スーパーノード同士はお互いの状態の把握が比較的容易である．そのため，スーパーノード同士での公開鍵の交換が可能

Hiroyuki TANAKA†Kiyohisa Ishiguro †Masaki Kondo †Shoichi Saito †Hiroshi Matsuo †
†Nagoya Institute of Technology

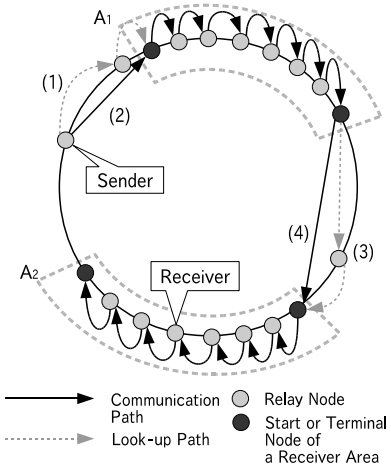


図 1: 既存方式の中継の流れ

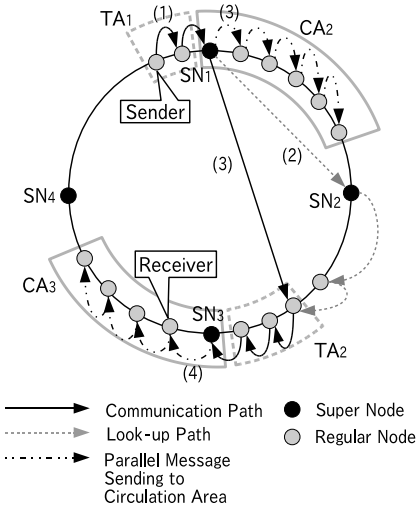


図 2: 提案方式の中継の流れ

となり，鍵を配布することが出来る．また，スーパーノードは公開鍵と秘密鍵のペアを持ち，メッセージを復号できるため，Successor だけでなく，任意のノードへの送信を並列に行える．

3.2 公開鍵の配布機能

経路構築のために送信者が公開鍵を集める処理を無くし，既存方式よりも匿名性を向上させるために公開鍵配布機能を実現する．各スーパーノードは，自ノードから次のスーパーノードまでの連続した ID 空間に含まれるノードへの公開鍵とメッセージの配布を担当する．このエリアを（スーパーノードを含めて）**配布エリア**(**Circulation Area: CA**) と呼ぶ．公開鍵の配布は以下の 2 つのステップで行われる．(1) 各スーパーノード同士で公開鍵を交換してすべての公開鍵を集める．(2) 各スーパーノードが配布エリアへ公開鍵を配布する．これによりすべてのノードに公開鍵が行き渡る．ここで配る公開鍵は経路情報を暗号化するために用いる．

3.3 並列送信機能

提案方式では通信を高速化するためにメッセージの並列送信機能を実現する．受信エリアの終端ではなく，配布エリアの始点であるスーパーノードで復号を行うため，既存方式よりも速くメッセージを次の受信エリアに送ることが出来る．

図 2 に提案方式の動作を示す．提案方式における，通常ノードから次にメッセージを復号するスーパーノードまでの連続した ID 空間を**誘導エリア**(**TaxiingArea: TA**) と呼ぶ．図 2 では誘導エリアは破線で囲まれた領域 TA_i である．配布エリアは実線で囲まれた領域 CA_i である．提案方式では CA_i と TA_i を合わせた領域を受信エリアと呼ぶ．図 2 の TA_1 は送信者から最初に復号を行うスーパーノードまでの範囲であり，どの受信エリアにも属さない．また，最後の受信エリアは誘導エリアを持たない．

送信者はメッセージ生成後，(1)TA_1 のノードを介して，メッセージを SN1 まで送る．この例では，SN1 はヘッダ部の復号に成功し，次にメッセージを送るべき宛先 ID が得られる．(2) 宛先 ID を担当するノードを Chord を用いて検索し，(3) 得られた宛先 ID のノードと CA_2 への送信を同時に行う．その後，TA_2 を経由して SN_3 受信すると再び復号が行われる．ここで SN_3 は復号したヘッダから次の宛先が無く，終端であることを知り，(4) CA_3 にのみメッセージを送る．

4 評価

100Mbps のイーサネットにネットワークスイッチを介して接続した 32 台の計算機を用いて，環状のオーバーレイネットワークを構成し，提案方式の評価実験を行った．実験に用いた計算機のスペックは Sempron 2800+/1.6GHz，メモリ 1GB，OS は Linux である．

実験 1 では往路と復路ともに暗号化の多重度が 2(往復で 4)，16hop の経路 (往復で 32hop) を構築し，送信データサイズを $L_M = 1, 64, 128, 256, 512, 1024, 2048, 4096[KB]$ と変化させた場合の RTT を計測する．実験 2 では，暗号化の多重度を変化させ，128KB の送信データに対する処理時間を計測する．各実験では 1 つの配布エリアに 8 台のノードを割り当て，TA_1 は送信者が

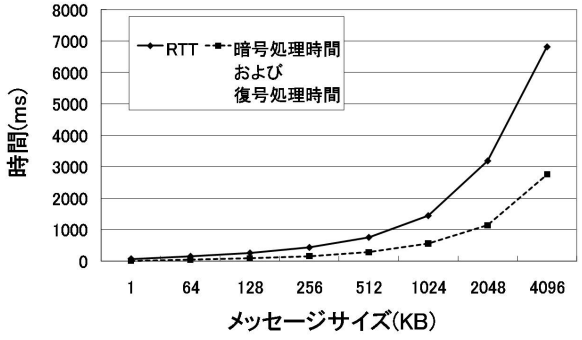


図 3: 実験 1：RTT と暗号復号処理時間 (片道暗号化多重度 2，片道中継ノード数 16)

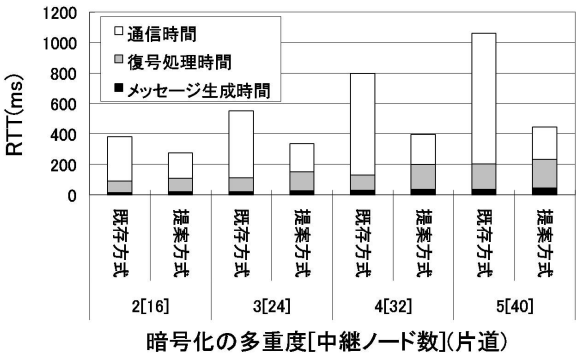


図 4: 実験 2：暗号化の多重度とデータ通信時間

1 台いる以外は，TA に属するノード 0 台としたため “エリア数 * 8”が中継ノード数となる．また，送信者と受信者がそれぞれ復路と往路の終端に位置するように通信路を設定した．

実験 1 の結果を図 3 に示す．図 3 から RTT は，通信データサイズが 1MB の場合で RTT は約 1.4 秒である．既存方式は約 2.6 秒 [3] であり，RTT が約 1.2 秒短くなっている．これは並列送信機能の効果である．

実ネットワークで稼動している Tor[2] では，中継ノード数 4 台，片道 4hop の通信路の生成時間が 7 秒，データ通信時間が約 2 秒である [5]．提案方式では，片道中継ノード数が 16 台の場合で片道 8hop である．また，既存方式では片道 16hop である．実験では LAN を用い，Tor の性能評価はインターネットで行っているが，中継ノード数を考慮すると，提案方式は十分実用的な性能であると考えられる．

図 4 に実験 2 の結果を示す．通信時間はメッセージが伝搬するのに要した時間であり，復号処理時間はメッセージの復号に要した時間の合計である．また，メッセージ生成時間にはメッセージの暗号化時間も含まれる．

既存方式は暗号化の多重度が増加すると，通信時間が大きく増加しているのに対し，提案方式はほとんど増加していないことがわかる．既存方式では，SN の暗号化の多重度が 1 つ増えると受信エリア数が 1 つ増え，中継ノード数と hop 数も 8 増加する．しかし，提案方式はメッセージの並列送信機能により，1hop しか増えない．このため，エリア数が増えて中継ノード数が増えても通信時間の増加は既存方式より少なく，増加時間は既存方式の 8 分の 1 程度である．以上から，メッセージ送信の並列化の効果が現れていることが分かる．

5 まとめ

本稿では，通信経路を秘匿するための公開鍵の配布機能を有する匿名通信方式を提案した．提案方式では多重暗号化と Chord を組み合わせて用いていることで離脱耐性の高い匿名通信路を実現している．スーパーノードを導入することで，公開鍵の配布機能とメッセージの並列送信を実現した．また，提案方式をオーバーレイ構築ツールキットである Overlay Weaver を基盤に提案方式の実装を行い，評価を行った．その結果，メッセージの並列送信機能により RTT が短縮されていることを確認した．インターネット環境での評価と，受信者との共有鍵の共有方法が今後の課題である．

謝辞

本研究の一部は，財団法人堀情報科学振興財団の研究助成，および文部科学省科学技術研究補助金基盤研究 C(課題番号:20500064) によるものである．

参考文献

[1] Pfitzmann, A. and Waidner, M.: Networks without user observability, Eurocrypt’85, LNCS 219, pp. 245–253 (1986).

[2] Dingledine, R. and Mathewson, N.: Tor: The Second-Generation Onion Router, Proceedings of 13th USENIX Security Symposium, pp. 303-320 (2004).

[3] 近藤 正基, 田中 寛之, 齋藤 彰一, 松尾 啓志 : 分散ハッシュテーブルによるノード管理を行う匿名通信方式の設計と実装, 情報処理学会研究報告書, 2009–OS–111 No.22 (2009).

[4] 首藤一幸, 田中良夫, 関口智嗣. オーバーレイ構築ツールキット Overlay Weaver. 情報処理学会論文誌, コンピューティングシステム, Vol. 47, No. ACS15 (2006).

[5] Andriy, P., Lexi, P. and Johannes, R.: Performance Analysis of Anonymous Communication Channels Provided by Tor, Third International Conference on Availability, Reliability and Security, pp. 221–228 (2008).

通信用公開鍵配布機能と メッセージの並列送信機能を有した 匿名通信方式

田中　寛之
（名古屋工業大学）

研究の背景

- 実社会
  - 匿名を前提とした仕組み
    - 投票，内部告発，医療相談，など

ギャップ

匿名通信が必要

- インターネット
  - IPアドレスと時間から個人の特定が可能

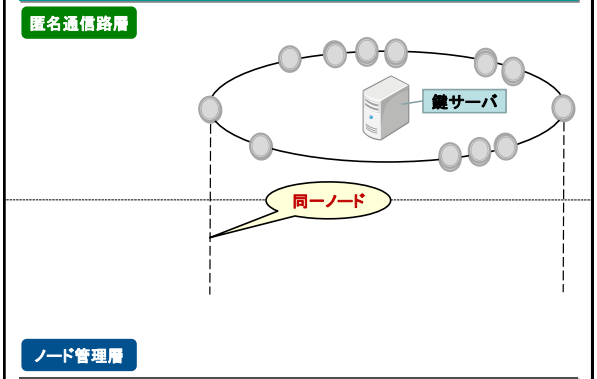
既存方式

- DHTを用いた双方向匿名通信方式 [近藤 2008]

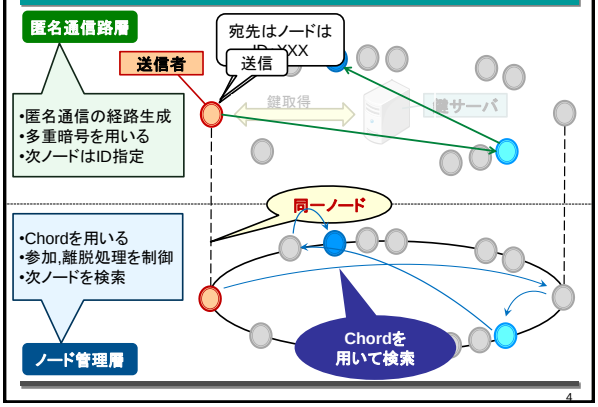
Chordとオニオンルーティングを組み合わせることで柔軟な経路生成が可能な匿名通信路を実現している

- 鍵サーバによる問題
- 通信遅延の問題

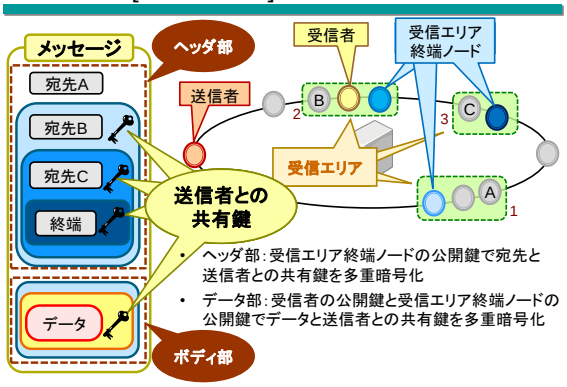
既存方式　[近藤　2008]：基本構造



既存方式　[近藤　2008]：基本構造

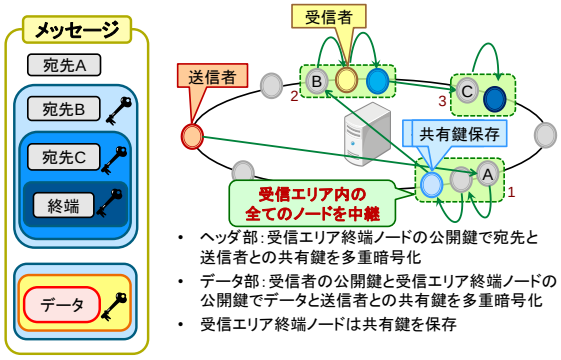


既存方式　[近藤　2008]：動作概要



- ヘッダ部：受信エリア終端ノードの公開鍵で宛先と送信者との共有鍵を多重暗号化
- データ部：受信者の公開鍵と受信エリア終端ノードの公開鍵でデータと送信者との共有鍵を多重暗号化

既存方式 [近藤 2008] : 動作概要



- ヘッダ部 : 受信エリア終端ノードの公開鍵で宛先と送信者との共有鍵を多重暗号化
- データ部 : 受信者の公開鍵と受信エリア終端ノードの公開鍵でデータと送信者との共有鍵を多重暗号化
- 受信エリア終端ノードは共有鍵を保存

6

既存方式 : 問題点

- DHTを用いた双方向匿名通信方式 [近藤 2008]

Chordとオニオンルーティングを組み合わせることで柔軟な経路生成が可能な匿名通信路を実現している

- 鍵サーバによる問題
  - 全参加ノードの鍵を管理するコストが高い
  - 鍵サーバを監視することで送受信者が分かる
- 通信遅延の問題
  - 中継ノード数の増加による通信遅延の増加が大きい

高速かつ鍵サーバの存在しないモデルが必要

7

提案方式

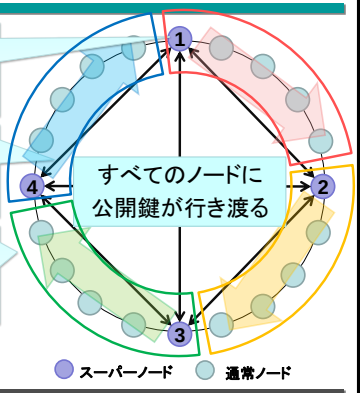
通信用公開鍵の配布機能とメッセージの並列送信機能を有する匿名通信路を提案する

- 通信用公開鍵の配布機能
  - 通信用公開鍵サーバが存在しないため匿名性が向上
- メッセージ送信の並列化
  - 最長hop数を削減し、通信速度が向上
- 通信用公開鍵を持つノードを一部のノードに限定
  - 通信用公開鍵を持つノードを**スーパーノード**と呼ぶ
  - 通信用公開鍵を持たないノードを通常ノードと呼ぶ

8

提案方式 : 公開鍵の配布動作

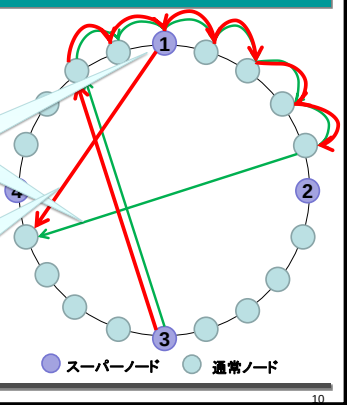
- 自分の公開鍵を持つノード（スーパーノード）
- スーパーノード同士で公開鍵を交換する
- 公開鍵を担当する配布エリアの通常ノードに配る



9

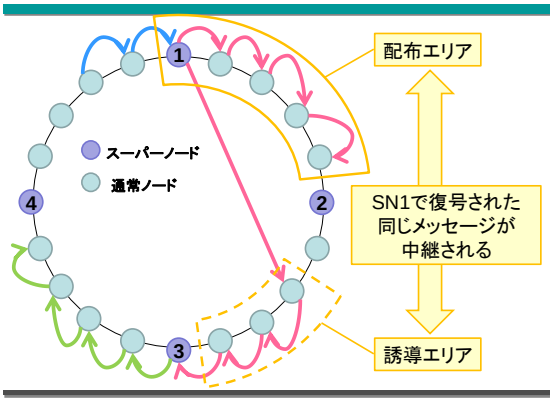
提案方式 : メッセージ送信の並列化

- 既存方式は受信エリアの全てのノードを中継し、次のノードへ
- スーパーノードでメッセージ送信を並列化する
- 提案方式はスーパーノードから次のノードへ

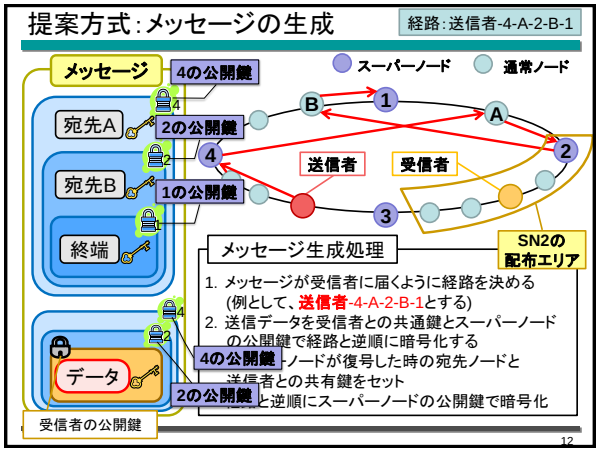


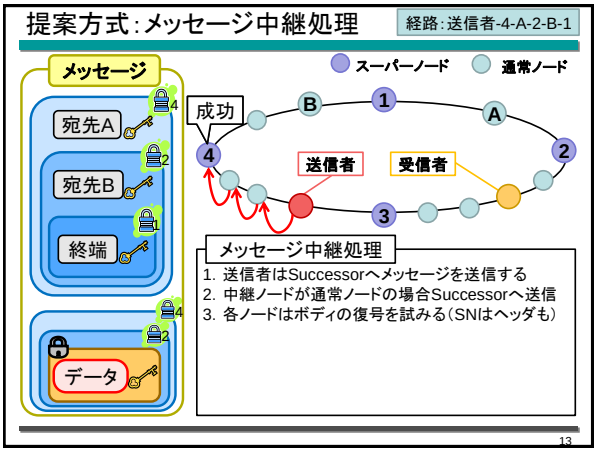
10

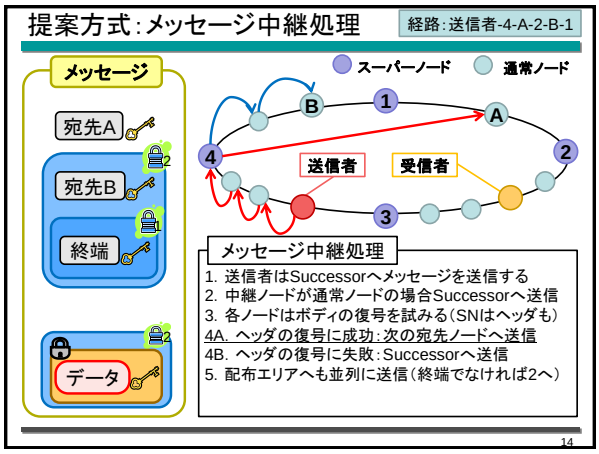
提案方式 : 配布エリアと誘導エリア

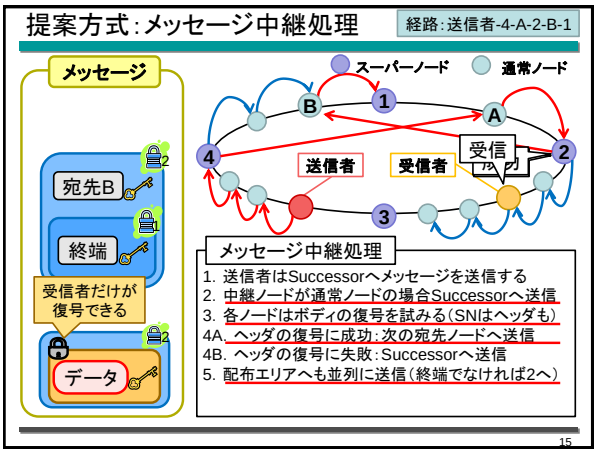


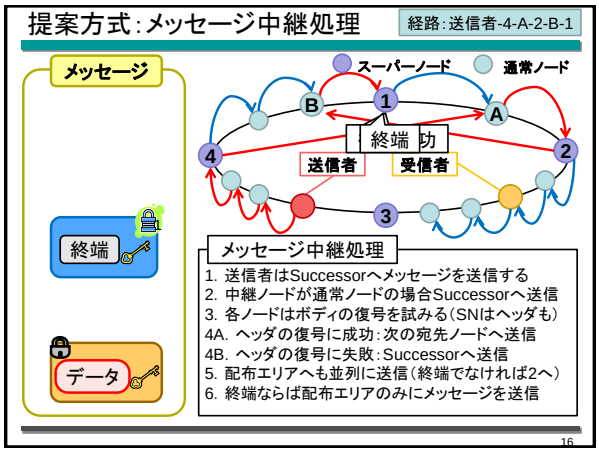
11











評価

- 1ノードを1台の計算機上で実行
- 32台の計算機が提案方式の匿名通信路に参加
- 送信データサイズを1KB～4MBとしてRTTを計測
  - 送信データサイズによるRTTの変化を確認
- 経路構築時間と匿名通信処理時間の計測
  - 多重暗号化の多重度による匿名通信処理時間の変化を確認

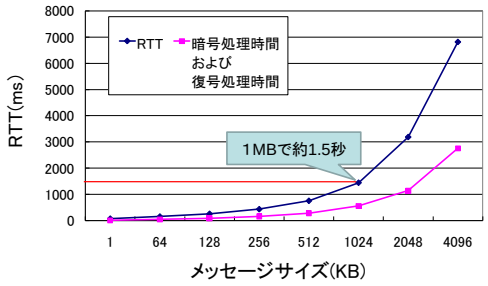
実装	評価環境
• 使用言語　Java • Overlay Weaver※を用いて実装	• Sempron 2800+, メモリ 1GB • 100Mbpsのネットワークに ネットワークスイッチを介して接続

※首藤 一幸他："オーバーレイ構築ツールキットOverlayWeaver"，2005

17

送信データサイズによるRTTの変化

- 復号回数2,中継ノード数16 （片道）

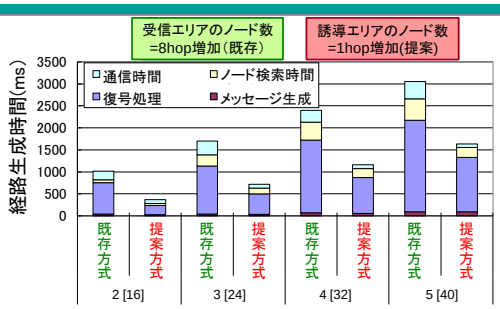


➢千田氏の手法※
送信データサイズ1MB：約2秒 （片道復号回数2回 片道2ホップ）

※千田 浩司他:"不正者追跡可能な匿名通信方式の実装と評価" 2005

18

経路生成時間



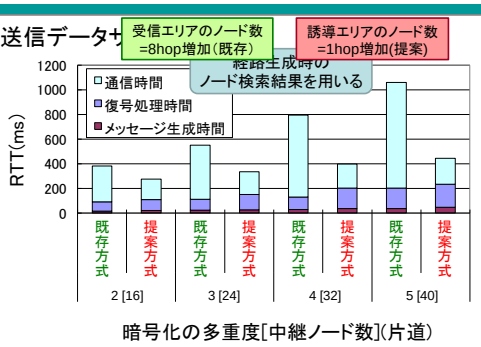
➢TOR経路生成時間： 4360ms （片道復号回数 3回，片道3ホップ)※

※Andriy Panchenko 他:" Performance Analysis of Anonymous Communicate Channels Provided by Tor" 2008

19

匿名通信処理時間

- 送信データサ



➢TOR RTT評価 ：約2秒 （片道復号回数4回，片道4ホップ)※

※Andriy Panchenko 他:" Performance Analysis of Anonymous Communicate Channels Provided by Tor" 2008

20

まとめ

- 通信用公開鍵の配布機能と並列送信機能を有する匿名通信路を提案
  - 公開鍵の配布機能により鍵サーバの負荷と監視される機会を減らしている
- 評価
  - 暗号と復号処理のコストが大きい
    （経路生成時：70%~80%、生成後：40%~50%）
  - メッセージの並列送信機能により既存方式に比べ2倍高速化
  - WEBサービスを利用するのに十分な性能
    （片道復号4回、中継ノード数32、128KBでRTTは400ms）
- 今後の課題
  - インターネット環境での評価
  - 送受信者間で共有鍵を共有する方法の検討

21

ユーザレベルで情報漏洩を防止するミドルウェア *User-Mode Salvia* の構築

河島 裕亮 †
† 立命館大学大学院理工学研究科

1 はじめに

近年, 計算機や情報通信技術の発展に伴い, プライバシ情報を電子データとして管理する情報の電子化が進んでいる. それに伴い, 電子化されたプライバシ情報が, データ保有者の意図に反して漏洩する事件が多発している. 文献 [1] では, 情報漏洩を引き起こす要因として, 情報の正規利用者の管理ミスによる流出や紛失, 外部への持ち出し, アプリケーションの誤操作, 盗難が挙げられている. 特に, 情報の正規利用者による情報漏洩が, 発生原因の多くを占めることを示している. しかし, 暗号化や認証といった既存のセキュリティ技術は, 外部からの攻撃を防止することを目的とするため, 人為的なミスや内部犯による情報漏洩を防止することは困難である.

以上の背景により, 人為的なミスや内部犯による情報漏洩を防止する手法として, Privacy-aware OS *Salvia*[2](以下, Salvia と記す) の開発を行ってきた. 上述の情報漏洩は, プロセスが発行するシステムコールを契機として発生する. そのため, *Salvia* では, システムコールを通じて重要なデータが漏洩しているか否かを検査する仕組を実現した. ただし, *Salvia* は, カーネルレベルで実装されているため, 保護機構の移植性向上や機能拡張を容易にすることを目的として, *Salvia* の保護機構をユーザレベルで実現するミドルウェア *User-Mode Salvia*(以下, UMS と記す) を構築している.

UMS は, 現在 Linux の ptrace システムコールを用いて *Salvia* の保護機構をユーザレベルで実現している. すなわち, UMS の動作に最低限必要な ptrace 相当のインタフェースが存在すれば, 僅かな修正で種々の OS へ適応可能となる. また, OS において, バージョンアップなどの仕様変更がある場合でも, そのインタフェースが変更されることは少ないため, UMS の保護機構を大きく修正する必要がない, といった利点がある.

以下, 本稿では, 2 章で UMS の概要について述べ, 3 章で実装した機能ついて述べる. また, 4 章で機能評価について述べ, 5 章で本稿をまとめる.

2 ユーザレベルでの情報漏洩防止機構

2.1 概要

計算機上で管理される情報が漏洩する際, プロセスで発行されたシステムコールを必ず経由することになる. そのため, *Salvia* は, 情報漏洩の原因となるシステムコールを制御することで, 情報漏洩を防止する. *Salvia* では, ファイルごとにデータ保護ポリシを設定可能とし, 漏洩の要因となるシステムコールが発行されると, 保護ポリシやコンテキストの内容に基き, そのシステムコールの実行可否を判定する. UMS は, このようなプロセスの監視, 保護ポリシ・コンテキストの管理, システムコールの実行制御といった機能をユーザレベルで実現する.

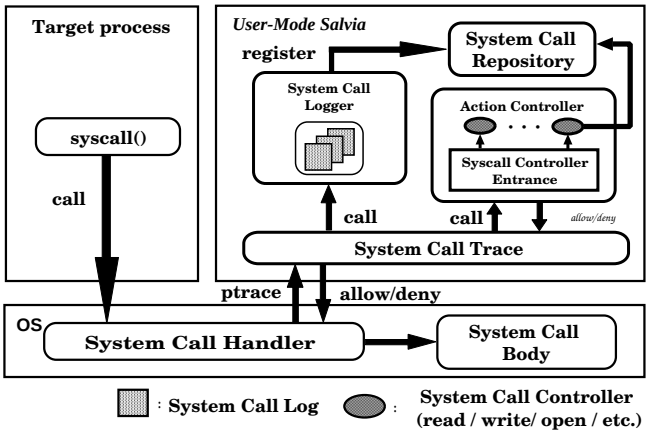


図 1 *User-Mode Salvia* の構成

2.2 全体構成

UMS は, Linux で提供される ptrace システムコールを基に Linux 上に実装している. UMS の情報漏洩防止機構 (図 1 参照) は, プロセスが発行するシステムコールの検知を行う System Call Trace, プロセスが発行するシステムコールを履歴として収集する System Call Logger, その履歴を時系列データとして管理する System Call Repository, データ保護ポリシ, コンテキストの取得, システムコールの実行制御を行う Action Controller から成る.

System Call Trace は, UMS によるプロセスの監視を行うために, ptrace を用いて監視対象プロセスと UMS 間の接続処理を行う. また, 監視対象のプロセスからシステムコールが発行されると, システムコールの入出口において, システムコール番号と返り値を取得する. ここで取得したシステムコール情報は, System Call Logger や Action Controller に渡す.

System Call Logger は, プロセスが発行したシステムコールを履歴として収集する. システムコール履歴の収集には, システムコールごとに異なる引数や返り値の数やデータ構造に対応させるため, それぞれに履歴取得用関数を設ける. 取得した履歴は, System Call Repository

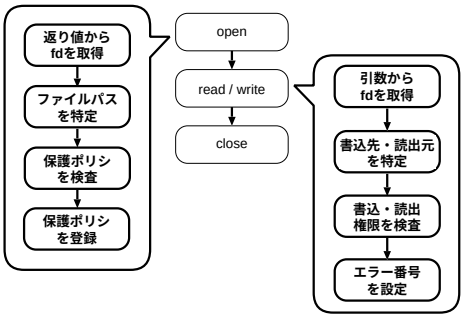


図 2　ファイル読出し・書込み制御手順

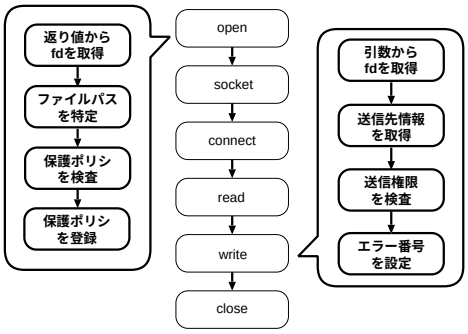


図 3　ソケット通信制御手順

において時系列データとして管理を行い, コンテキストの一種として利用する.

Action Controller は, 保護ポリシの取得や, 取得した保護ポリシの内容に応じて, 情報漏洩の要因となるシステムコールの制御を行う. 保護ポリシは, ファイルの拡張属性として管理する. そのため, open システムコールの発行時に, オープンするファイルにポリシが存在するか否かを確認することで, ポリシの有無を判定する. ポリシが存在する場合, その内容を登録する. その登録されたポリシの内容に応じてシステムコールの実行可否を制御する. システムコールを制御するためには, システムコールごとの性質や引数・返り値のデータ構造に対応させるため, それぞれに System Call Controller を用いる.

3　実装

従来の *Salvia* では, プロセスを制御する方法をデータの伝搬範囲に基づき, 以下のように分類する.

1. read：ファイルからデータを読出す処理
2. write：ファイルにデータを書込む処理
3. send_local：同一計算機上でのプロセス間通信
4. send_remote：他の計算機へのデータの送信

現在, UMS では, 1, 2, 4 を制御する機構の基本部分の実装を完了した.

3.1　ファイル読出し・書込み制御

open システムコール発行時　プロセスが open システムコールを用いてファイルにアクセスした際, そのファイルの拡張属性領域に保護ポリシが存在するか否かを確認する. 保護ポリシが存在する場合, その内容を UMS のポリシリストに登録する.

read/write システムコール発行時　read/write システムコールの第 1 引数の値を取得し, その内容から読出元, 書込先ファイルを特定する. 次に, 特定したファイルに対応する保護ポリシが UMS のポリシリストに登録されているか否かを検査し, ファイルに対する読出し, 書込み権限を調べる. 保護ポリシを検査した結果, ファイルへの読出し, 書込みが許可された場合, 通常のシステムコールの処理を行い, 禁止された場合, システムコールの返り値にエラー番号を格納し, システムコールの処理を中断する.

3.2　ソケット通信制御

open システムコール発行時　ファイル読出し・書込み制御時と同様の処理を行う.

send/write システムコール発行時　send/write システムコールの第 1 引数の値を取得し, これらシステムコールがソケット通信目的で発行されているか否かを検査する. 次に, 取得した引数の値と監視対象となっているプロセスのプロセス ID を基に, "/proc/net"からソケット通信に関する情報を取得する. 取得した通信先の情報と open 時に登録した保護ポリシを比較し, 通信先へのデータ転送を制御すべきか否かを検査する. 通信先へのデータ転送が許可された場合, 通常の send(write) システムコールの処理を実行し, 禁止された場合, システムコールの返り値にエラー番号をセットし, システムコールの処理を中断する.

4　機能評価

前章で述べた, ユーザレベルにおいてデータ保護ポリシに基づいたファイルアクセス制御が実現可能であることを示すために, 以下の実験を行った.

実験 1：ファイル読出し制御　ファイルの読出し処理に関する処理を禁止と設定した保護ポリシを付与したファイルを cat コマンドでアクセスした際の動作を確認

実験 2：ファイル書込み制御　ファイルの書込み処理に関する処理を禁止と設定した保護ポリシを付与したファイルを テキストエディタでアクセスし, データ書込みを行った際の動作を確認

実験 3：ソケット通信制御　計算機外へのデータ転送を禁止と設定した保護ポリシを付与したファイルを転送しようとしたときの動作を確認

実験の結果, 実験 1 ではファイルからデータを読出す際に発行される read システムコールが, 保護ポリシに基づき制御され, データ出力を防止したことを確認できた. 実験 2 では, テキストエディタでファイルにデータを書込み, 書き込んだ内容を保存する際に発行される write システムコールが保護ポリシに基づいて適切に制御され, データの書込みを防止したことを確認した. また, 実験 3 では, データ転送時に発行される write システムコールが保護ポリシに基づいて適切に制御され, 計算機外へのデータの週出防止を確認できた. 以上のことから, ユーザモードからファイル読出し, 書込み, ソケット通信を制御し, データ漏洩を防止可能であることが示せた.

5 おわりに

本稿では, ユーザレベルで情報漏洩を防止するミドルウェア *User-Mode Salvia* の概要, 実装, 機能評価について述べた. 今後の課題としては, 使用可能なコンテキストの拡張やインタフェースの改善, 同一計算機上でのプロセス間通信を制御する機構の実装がある.

参考文献

[1] NPO 日本ネットワークセキュリティ協会: JNSA 2007 年情報セキュリティインシデントに関する調査報告書, http://www.jnsa.org/result/2007/pol /incident/2007incidentsurvey_v1.32.pdf, 2008.

[2] 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 (ACS 13), Vol. 47, No. SIG3, pp. 1–15, 2006.

ユーザレベルで情報漏洩を防止するミドルウェア *User-Mode Salvia*の構築

立命館大学大学院　理工学研究科
毛利研究室
河島 裕亮

発表内容

- はじめに
- Privacy-aware OS *Salvia*の概要
- *User-Mode Salvia*
  - 全体構成
  - 実現した機能
    - ファイルアクセス制御
    - ソケット通信制御
  - デモ
- 今後の課題
- おわりに

2

はじめに(1/2)

- 近年，機密情報が漏洩する事件が多発している
  - ➢計算機，情報通信技術の発達に伴い，
    電子化された機密情報の漏洩頻度，規模が拡大
- 情報漏洩事件のおもな原因
  1. 正当な利用者の不注意による流出や紛失
  2. 正当な利用者による機密情報の持出し
  3. アプリケーションの誤操作
  4. 盗難

1, 2のような機密情報の正当な利用者による
情報漏洩が最も頻度が高い

3

はじめに(2/2)

- 暗号化や認証など，既存のセキュリティ技術は，
  外部からの攻撃を防止すること目的としている
- 人為的ミスや内部犯による情報漏洩を
  防止することは困難

Privacy-aware OS *Salvia*

4

Privacy-aware OS *Salvia* の概要 (1/2)

- アクセス制御機構を備えたOS
  - アプリケーションの信頼度に関わらず統一的に制御可能
- 保護方式
  - 保護単位：ファイル
    - 保護したいファイルと保護ポリシを一組にして管理
  - 制御対象：プロセス
    - プロセスがデータを漏洩させる動作を制御
    - プロセスが実行するシステムコールをチェックすることで制御
  - コンテキストの利用
    - ユーザや計算機の状況のこと
    - ユーザ，時間，計算機の場所，IPアドレス　など

5

Privacy-aware OS *Salvia* の概要 (2/2)

現在の*Salvia*は，
- OSへ変更を加える形で保護機構を構築

そこで，
- ユーザによる保護機構の導入の容易化
- 機能拡張や移植作業の容易化

を目的として，

Salvia の保護機構をユーザレベルで実現する
User-Mode Salvia の構築

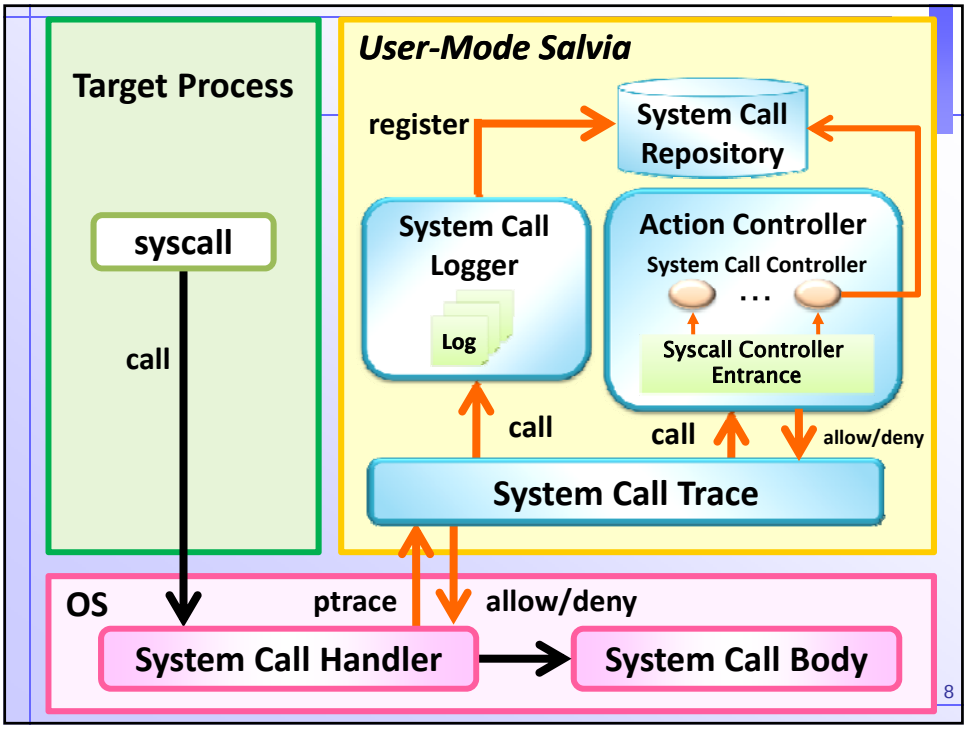
6

User-Mode Salvia

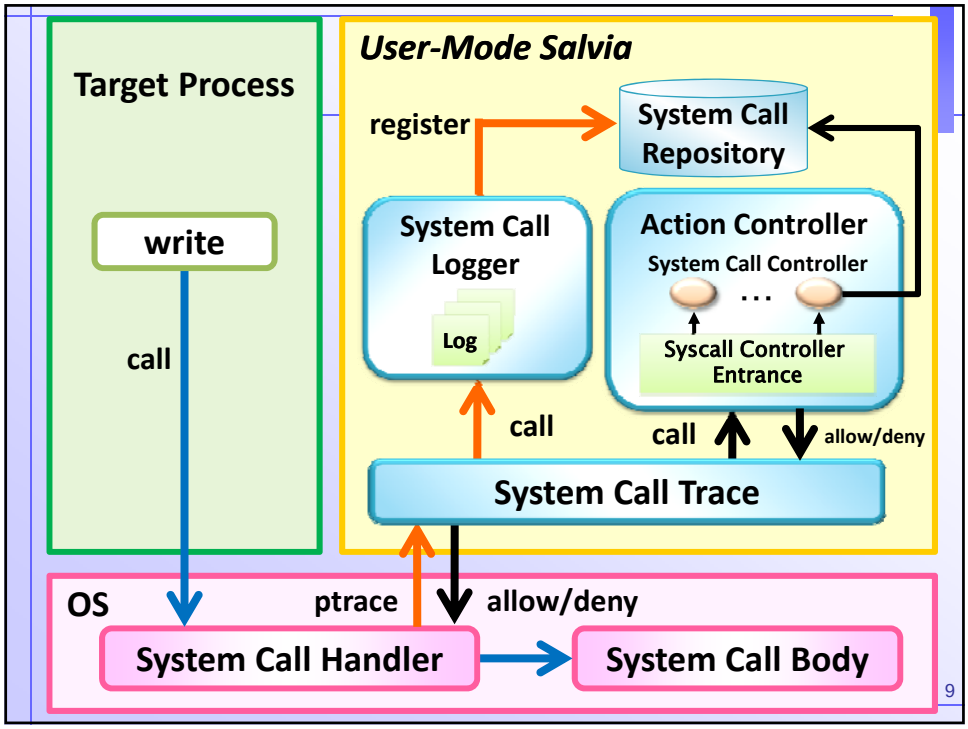
- *User-Mode Salvia*は,
  - プロセスの監視
  - 保護ポリシ・コンテキストの管理
  - システムコールの実行制御

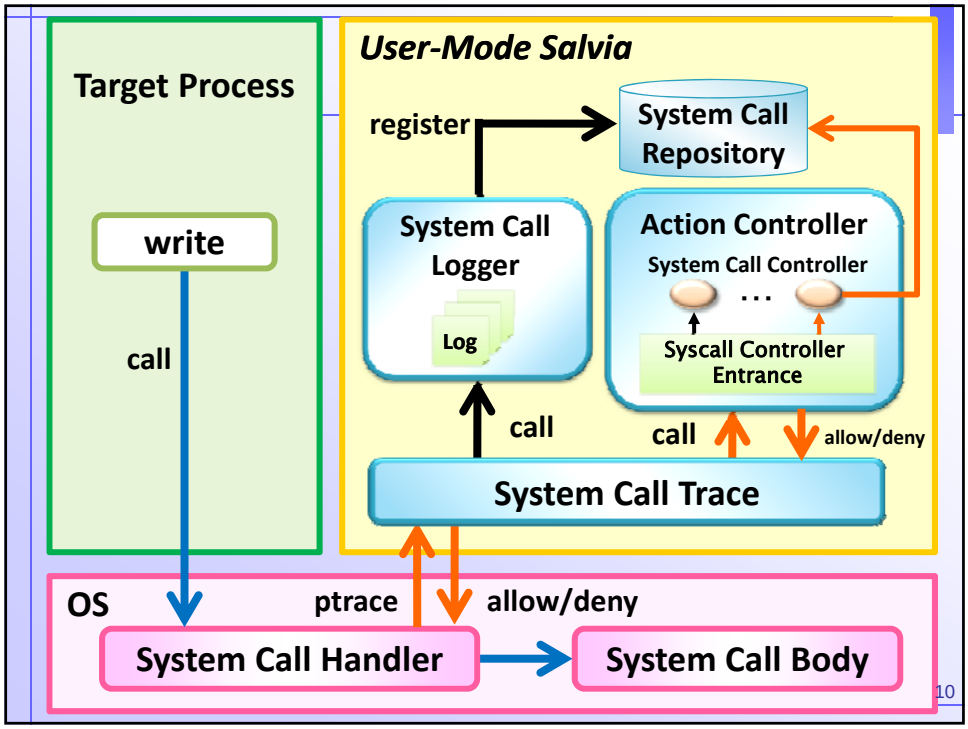
  をユーザレベルで実現するミドルウェア
- ptraceシステムコールを基に保護機構を実現
- *Salvia* をユーザレベルで実現することで,
  - ユーザによる保護機構の導入が容易
  - 移植性を向上
    - 導入環境にptrace相当の機能とインタフェースが存在すれば, 僅かな修正で種々のOSへ適応可能
  - デバッグや試験運用が容易

7



8


Target Process
write
call
User-Mode Salvia
register
System Call Repository
System Call Logger
Log
Action Controller
System Call Controller
Syscall Controller Entrance
call
call
allow/deny
System Call Trace
OS
ptrace
allow/deny
System Call Handler
System Call Body
9


Target Process
write
call
User-Mode Salvia
register
System Call Repository
System Call Logger
Log
Action Controller
System Call Controller
Syscall Controller Entrance
call
call
allow/deny
System Call Trace
OS
ptrace
allow/deny
System Call Handler
System Call Body
10

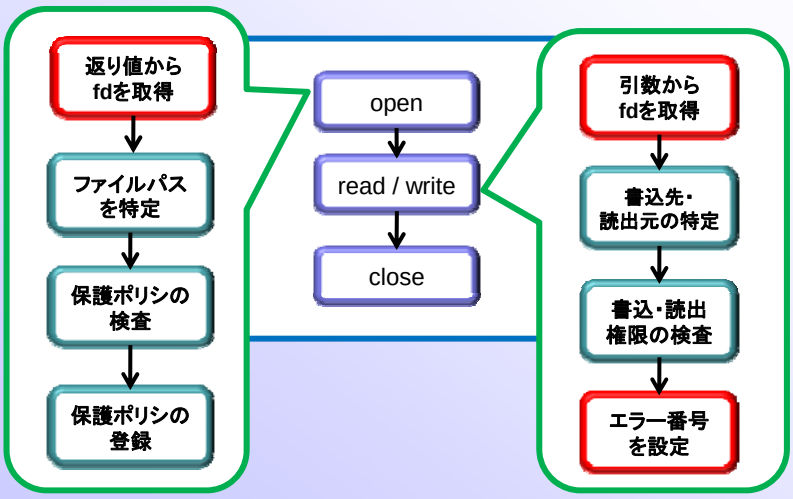
実装した機能

- *Salvia*では, データの伝搬範囲に基づき, 制御対象を分類
  1. ファイルに対する読出し(read)
  2. ファイルに対する書込み(write)
  3. 同一計算機上でのプロセス間通信(send_local)
     - パイプ、共有メモリ、ソケットなど
  4. 他の計算機への送信(send_remote)

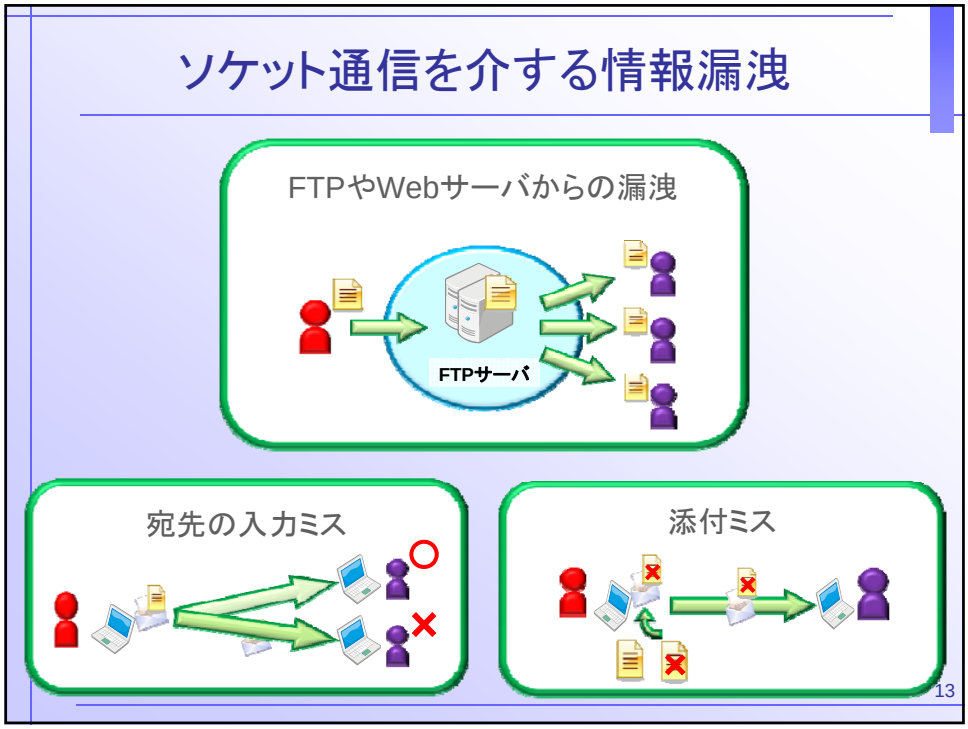
1, 2, 4を制御する機構の基本部分を実装

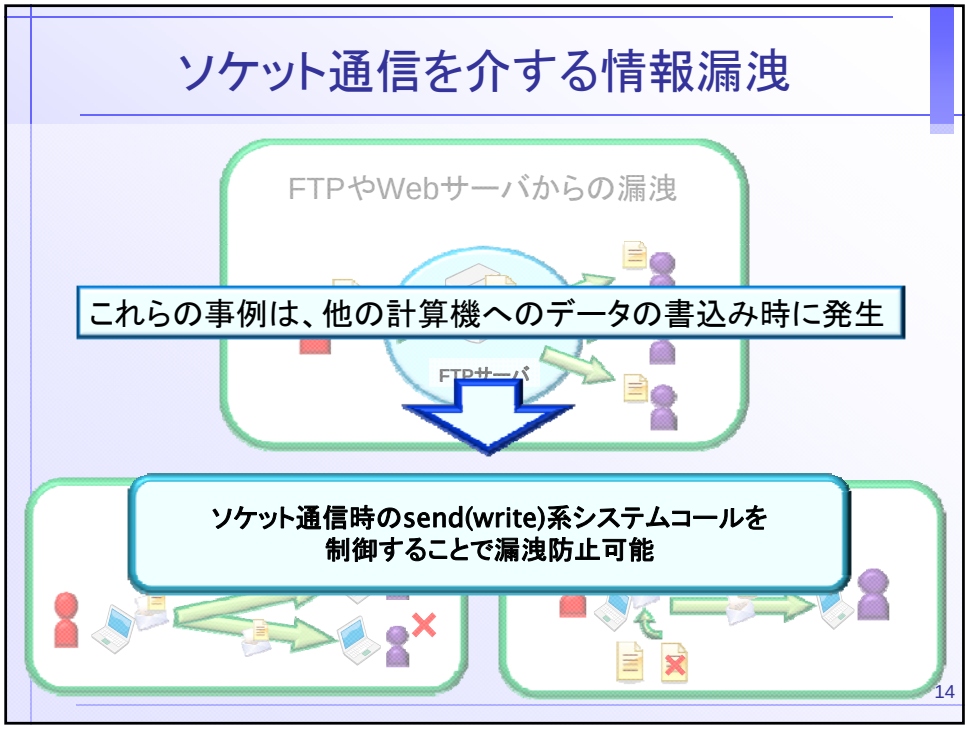
11

ファイル読出し, 書込み時の制御手順

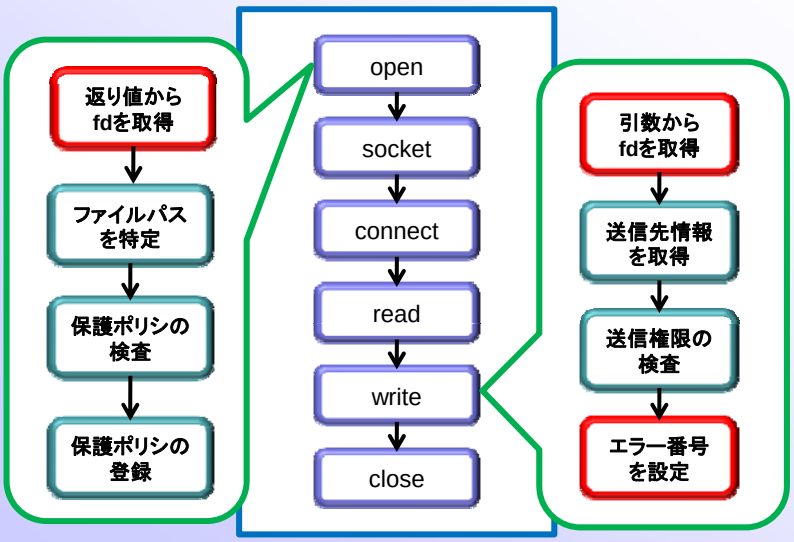


12


ソケット通信を介する情報漏洩
FTPやWebサーバからの漏洩
FTPサーバ
宛先の入力ミス
添付ミス
13


ソケット通信を介する情報漏洩
FTPやWebサーバからの漏洩
これらの事例は、他の計算機へのデータの書込み時に発生
ソケット通信時のsend(write)系システムコールを
制御することで漏洩防止可能
14

ソケット通信時の制御手順



15

デモ

- ファイル読出しの制御　(default - read)
- ファイル書込みの制御　(default - write)
- scpコマンドの制御 (default - send_remote)

16

動作に必要な機能

- ptraceシステムコール
  - ユーザレベルでプロセスを監視するために必要
- procファイルシステム
  - ユーザレベルでプロセスの詳細な情報を取得するために必要
  - IPアドレス, ファイルディスクリプタ
- Expat
  - xmlパーザツールキット
  - 保護ポリシをバイナリに変換するために必要
- i-node拡張属性領域
  - 保護ポリシを管理するために必要
  - なければ, ファイルによる保護ポリシの管理

17

今後の課題

- 使用可能なコンテキストの拡張
  - ユーザID, 時間, RFIDタグ情報, GPS
- インタフェースの改善
- 共有メモリ制御機構の検討
- 他OSへの移植実験
  - BSD, Mac

18

おわりに

- *User-Mode Salvia*
  - ファイル読出し・書込み制御機能の実装
  - ソケット通信制御機能の実装
  - デモ
- 今後の課題
  - 使用可能なコンテキストの拡張
  - インタフェースの改善
  - 共有メモリ制御機構の検討
  - 他OSへの移植実験

19

災害情報システムにおけるDTNルーティングを用いたデータ配送機構

陶山 優一† 横田 裕介†† 大久保 英嗣††
† 立命館大学大学院理工学研究科 †† 立命館大学情報理工学部

1 はじめに

大規模災害発生時における被害状況の把握は，救助活動や避難経路確保において非常に重要となる．しかし，災害時には停電やネットワークケーブルの切断，通信施設の倒壊などにより，既存のネットワークインフラを利用できない場合がある．

このような環境では，既存のネットワークインフラに頼らない無線アドホックネットワーク技術が有効となる．このアドホックネットワーク技術を災害情報の収集に利用することが検討されてきている [1, 2]．しかし，アドホックネットワークではノードの位置や密集度によって，頻繁にネットワークの分断が発生する．このため，データを送る際に相手までのリンクが存在しない，転送中にリンクが切断されるといった可能性がある．

そこで，本研究では，災害情報システムにおける Delay/Disruption Tolerant Networking（以下，DTN と記す）を用いたデータ配送機構を提案する．本機構を用いることで，広域の災害情報を高い信頼性で収集することが可能となる．

2 Delay/Disruption Tolerant Networking

2.1 概要

DTN とは，データをストアアンドキャリー方式で伝搬することにより，断続的な接続性や通信遅延が発生する環境におけるデータ配送を可能にする技術である．

これまで主に用いられてきた TCP/IP では，エンドポイント間に常にパスが存在することが前提となっている．しかし，DTN では中継ノード間をアプリケーションデータ単位でデータの転送，蓄積を行う．データを所持するノードは，ノードの移動によるトポロジの変化や通信環境の改善等により次の中継ノードまたは目的ノードとの接続が可能になった際にデータを転送する．これを繰り返すことにより，エンドポイント間でパスが存在しない環境下でもデータ通信を可能としている．

2.2 災害情報システムへの DTN 適用における課題

DTN では，データ転送がノードのモビリティに依存するため，ネットワークが分断されている環境下では，既存の通信と比較し，大きな伝送遅延が発生する．また，ストレージ超過による中継データの置換や中継ノードの離脱によるデータの紛失も発生する．このため，DTN によるネットワーク網を既存の通信インフラの代替としてそのまま適用することは好ましくない．そこで，本研究では，災害情報システムの特性を考慮した DTN によるデータ配送機構を提案する．

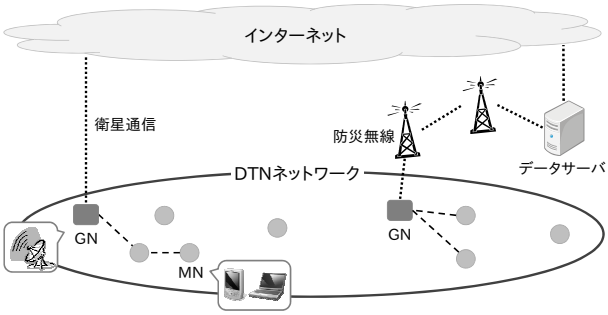


図 1 ネットワークアーキテクチャ

3 DTN を用いたデータ配送機構

3.1 ネットワークアーキテクチャ

本研究では，既存の災害情報システムで用いられているネットワークと DTN ネットワーク網を組み合わせたネットワークを想定する．図 1 に想定する災害情報システムのネットワークアーキテクチャを示す．

本研究で想定するネットワークは，DTN ネットワークレイヤと専用回線レイヤに分類される．DTN ネットワークは，無線 LAN に対応した携帯電話や PDA，ノート PC のようなモバイルノード（MN）および衛星通信や防災無線等の専用回線へのゲートウェイノード（GN）で構成される．

本ネットワークでは，既存の災害情報システムで用いられる通信デバイスと同様のものを用いている．このため，通信インフラに被害が発生し，本データ配送機構に切り替えを行う場合にオーバヘッドを必要としない．また，本ネットワークでは，DTN ネットワークと信頼性の高い専用回線をゲートウェイで結ぶことにより，専用機器を持たないユーザに対するインターネットアクセスの提供，目的ノードへのホップ数削減による通信遅延や到達確率の改善が可能となる．

また，本研究では，ネットワークに参加するモバイルノードをパブリックノードとプライベートノードに分類する．パブリックノードは，消防，救助隊が所持する端末であり，パブリックな目的に用いるため，他のノードからのデータを受信し，転送するといったルーティングを行う役割を担う．特定の個人のサービスのためだけに動作することはない．プライベートノードは，特定の個人に対応するノードであり，所有する個人が受けるサー

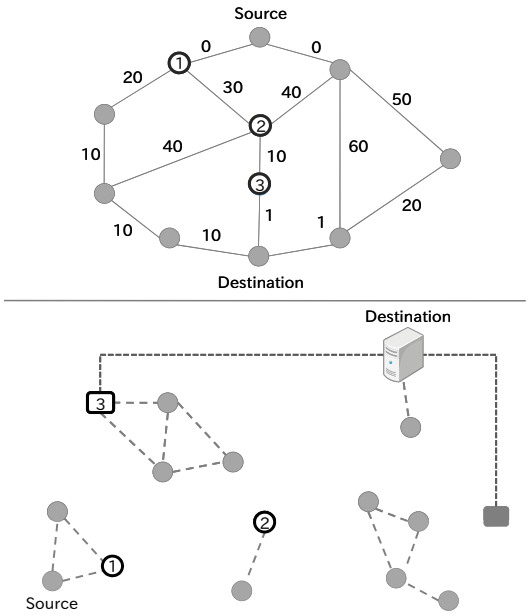


図 2　経路決定

ビスのために動作する．基本的に他のノードからのデータをルーティング，リレーする役割は担わない．これにより，被災者への電力負担を軽減する．

3.2　ルーティング

本機構では，ユニキャスト，マルチキャスト，ブロードキャストの 3 種類の通信方式を提供する．以下，各通信方式について述べる．

3.2.1　ユニキャスト

ユニキャストは，特定のノードにのみデータを送信したい場合，例えば災害対策本部への災害情報や救助要請の送信，部隊間通信などに用いる．ユニキャストにおける通信手順を次に示す．

1. ネットワークに参加するノードは，定期的に自他ノードの情報（ノード ID，グループ ID，ノードの種別，ノード間のコスト）を交換する．ノード ab 間のコスト c(a,b) は次式によって得られる．$P(a,b)$ はノード a がノード b と接続可能である確率，T_{term} は過去任意の期間，T_b は過去 T_{term} の間にノード a がノード b と接続可能であった時間を示す．

$$P(a,b) = \frac{T_b}{T_{term}} \tag{1}$$

$$c(a,b) = P(a,b)^{-1} \tag{2}$$

2. ソースノードは，手順 1 で得られたノード情報を基にノードを頂点，上式で得られたコストを辺の重みとしたグラフを構築し，ダイクストラ法を用いて経路を得る（図 2 参照）．図 2 では，Souce，1，2，3，Destination が最短経路として求められる．

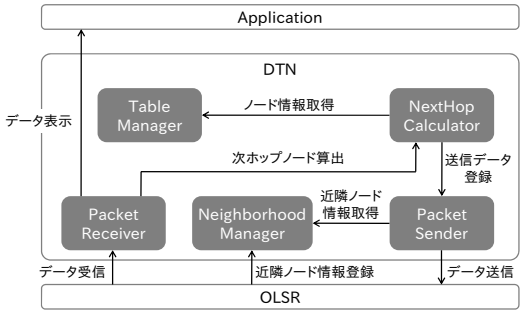


図 3　データ中継処理におけるモジュール間の関係

3. データを所持するノードは，同一ネットワークに経路上のノードを検知した場合，データを転送する．

3.2.2　マルチキャスト

マルチキャストは，現場における特定のグループにデータを送信したい場合，例えば災害現場の動画情報など現地での情報共有に用いる．災害現場では，災害発生から時間経過に伴い活動部隊が増減する．このため，本機構では，データサーバで定期的にマルチキャストグループリストを作成，更新を行い，ネットワーク上のパブリックノードにブロードキャスト送信する．マルチキャストを行う場合，このマルチキャストグループリストを用いて，送信対象を特定し，ユニキャストと同様のアルゴリズムを用いて送信する．

3.2.3　ブロードキャスト

ブロードキャストは，できるだけ広い範囲にデータを送信したい場合，例えば被災地の地理情報や安否情報，避難場所など不特定多数が必要とする情報を配布する際に用いる．ブロードキャストでは，ユニキャスト，マルチキャストとは異なり，フラッディングベース（epidemic routing）でデータ送信を行う．また，ブロードキャストで扱うデータにはバージョン情報を付与し，転送中に新しいバージョンのデータが見つかった場合，古いデータは破棄する．

3.3　プロトタイプ実装

現在，本機構を Linux 環境で動作するアプリケーションとして実装を行っている．アドホックネットワークのルーティングプロトコルには，OLSR の Linux 実装である olsrd[3] を利用している．OLSR は，事前にルーティングテーブルを作成するプロアクティブ型のプロトコルである．本機構における各モジュールの説明を次に示す．また，データの中継処理におけるモジュール間の関連を図 3 に示す．

PacketReceiver　受信したデータの種類（中継，最終，ノード情報）に応じて処理を切り分ける．

NextHopCalculator　TableManager を参照し，次のホップ先の情報をデータに書き込み，PacketSender に登録する．

TableManager　ノードテーブルおよびコストテーブルを管理する．

NeighborhoodManager　同一リンク上にいるノードの情報を管理する．

PacketSender　NeighborhoodManager を参照し，登録されたデータの次ホップノードが同一リンク上に存在する場合，データを OLSR に渡す．

4　おわりに

本稿では，災害情報システムにおける DTN を用いたデータ配送機構について述べた．今後は，まず本機構で扱うデータおよびシナリオの再検討を行う．その後，災害時のシナリオを作成し，本キャンパス内で実験，評価を行う予定である．

参考文献

[1] 柴山明寛，遠藤真，滝澤修，細川直史，市居嗣之，久田嘉章，座間信作，村上正浩: 地震災害時における情報収集支援システムの開発，日本建築学会技術報告集，no.23，pp.497–502，2006.

[2] 行田弘一，岡田和則，滝澤修: アドホックネットワークを用いた非常時通信モデルの基礎検討，電子情報通信学会技術研究報告．CQ，コミュニケーションクオリティ，vol.106，no.153，pp.95–98，2006.

[3] olsrd an adhoc wireless mesh routing daemon, http://www.olsr.org/

災害情報システムにおける DTNルーティングを用いたデータ配送機構

立命館大学大学院
大久保・横田 研究室
陶山 優一

Contents

はじめに

- ✓ 災害時のネットワークインフラ
  - 停電、ケーブルの切断、通信施設の倒壊、輻輳などによる通信障害
  - ➡ 災害情報（救助要請・被害情報）のやり取りが困難
- ✓ アドホックネットワーク
  - モバイル端末同士で動的に構築するネットワーク
  - 一般的な機器で利用可能（スマートフォン、PDA等）
  - 広域なエリアをカバーすることは困難

- ➡ Delay/Disruption Tolerant Networking

DTN:Delay/Disruption Tolerant Networking

- ✓ ストア アンド キャリー方式によるデータ伝搬

- ➡ 分断されたネットワーク環境下でもデータ通信が可能に
- ✓ データの伝搬遅延、到達性が問題
  - 緊急性の高いメッセージがなかなか届かない・途中で紛失
  - 代替の通信インフラとしてそのまま用いるのは不適切

実用的なDTNを用いたデータ配送機構を提案

既存システムにおける課題

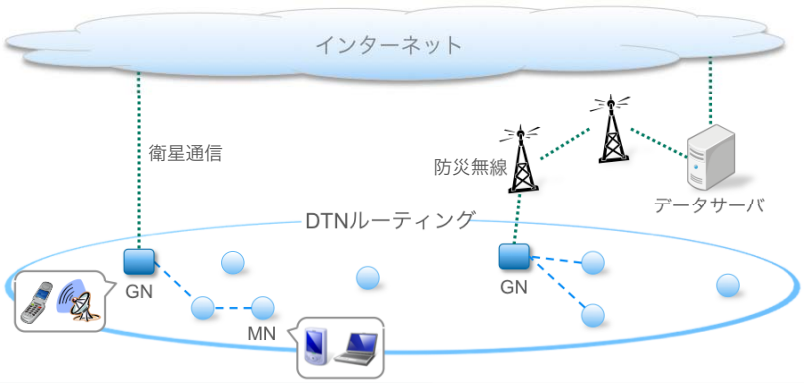
- ✓ 既存の災害情報システム
  - 携帯電話への依存
    - ✓防災・消防無線のチャネルが少ないため、すべての通信を専用通信で行うことは困難
    - ➡携帯電話網が利用できなくなった場合、災害情報システム自体が麻痺する恐れも
  - 衛星電話等の専用機器の配備コスト
    - ✓携帯電話の代替として用いるのはコスト面で困難

5

立命館大学大学院　大久保・横田研究室

ネットワークアーキテクチャ

- ✓ 既存システムとDTNネットワーク網のハイブリッド
  - 携帯等の無線LANを用いてDTNネットワーク網を構築
  - 衛星電話、防災無線等をゲートウェイとして利用



6

立命館大学大学院　大久保・横田研究室

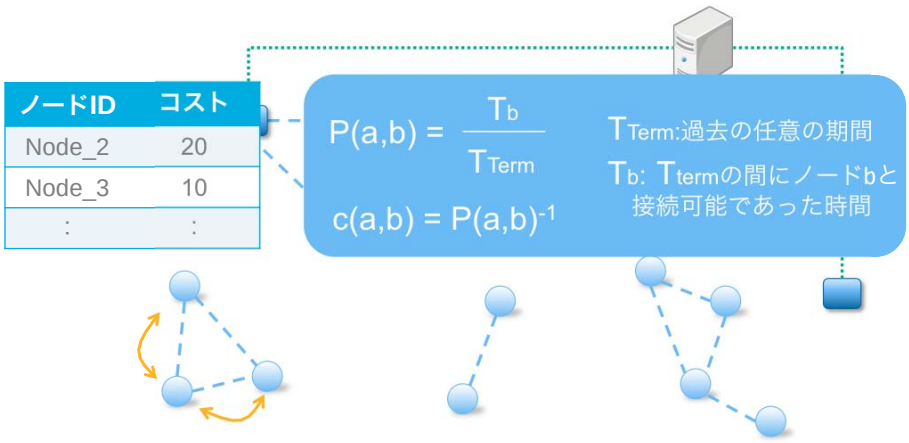
ネットワークアーキテクチャ：ノードの分類

- ✓ Gateway Node
  - 無線LAN ＋ 衛星通信、防災無線等のネットワークインタフェースを持つ端末
- ✓ Mobile Node
  - Public Node
    - ✓無線LAN対応の端末（消防、救助隊等が所持）
    - ✓アドホックネットワークを構築、維持
  - Private Node
    - ✓無線LAN対応の端末（被災者等が所持）
    - ✓情報を提供、取得する時のみネットワークに参加

ルーティング

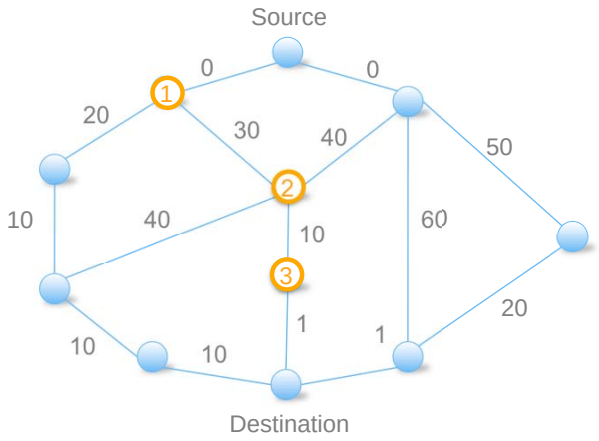
- ✓ ユニキャスト
  - 利用例：データサーバへの災害情報・救助要請の送信など
- ✓ マルチキャスト
  - 利用例：部隊間通信など
- ✓ ブロードキャスト
  - 利用例：地理情報、安否情報、避難経路等

ルーティング：ユニキャスト



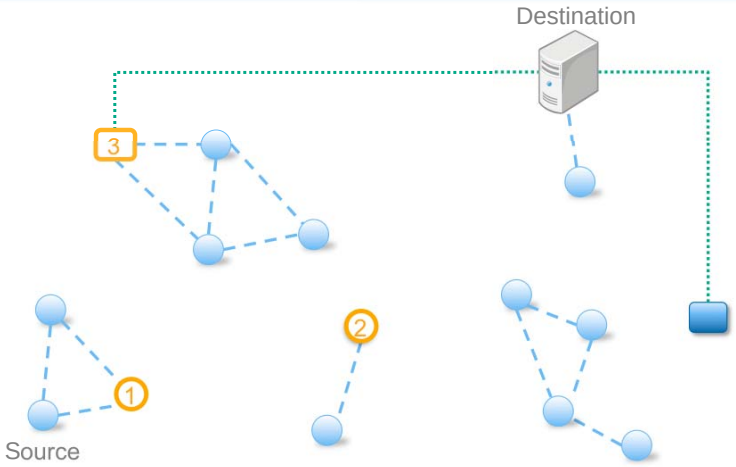
0. ノード情報（ノードの種類、コスト）を定期的に近隣ノードと交換

ルーティング：ユニキャスト



1．目的ノードまでの経路を決定
（各ノード間のコストからダイクストラ法を用いる）

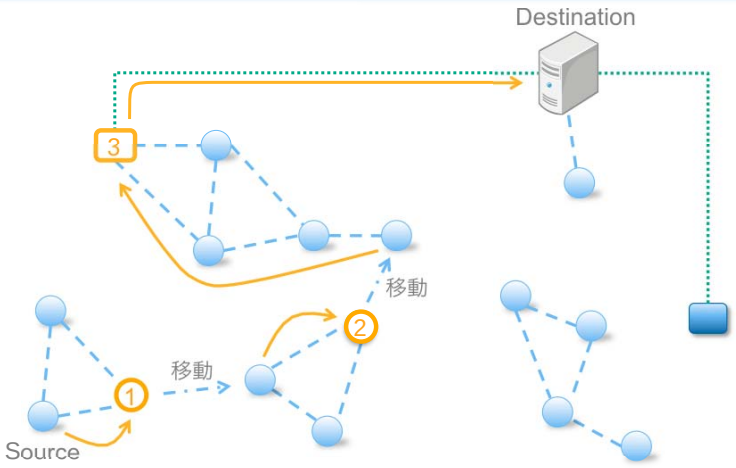
ルーティング：ユニキャスト



1．目的ノードまでの経路を決定
（各ノード間のコストからダイクストラ法を用いる）

11

ルーティング：ユニキャスト

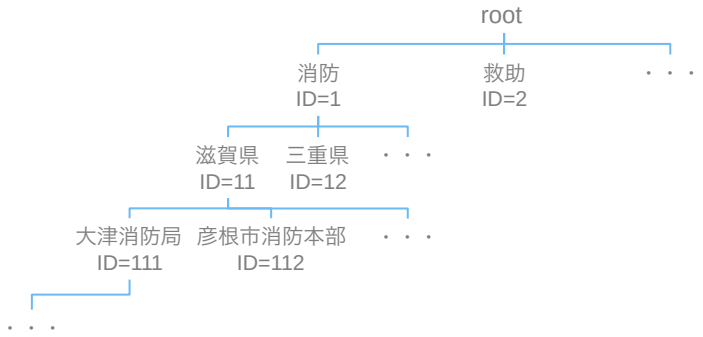


2．同一ネットワーク上に経路上のノードを検知した場合、
データを転送

12

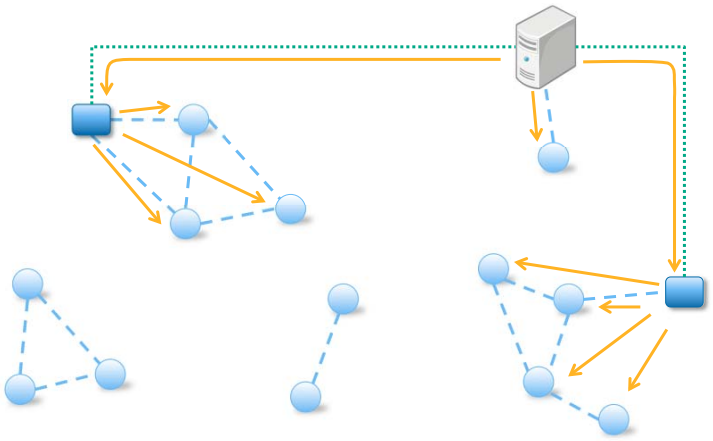
ルーティング：マルチキャスト

- ✓ データサーバから定期的にマルチキャストグループを更新 ＋ ブロードキャスト
- ✓ グループIDに応じてデータを配送



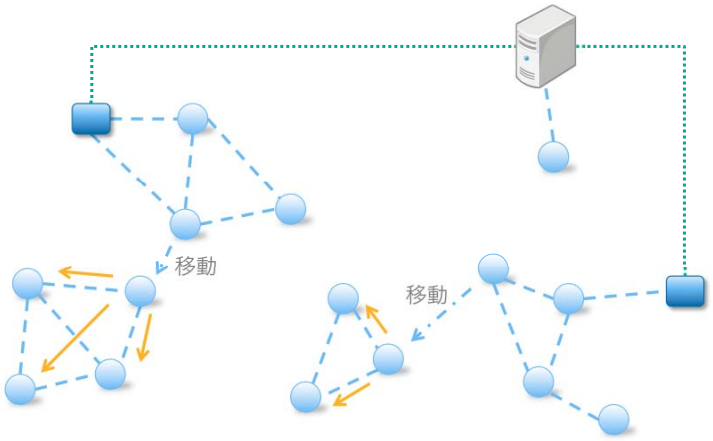
ルーティング：ブロードキャスト

- ✓ フラッディングベース(Epidemic Routing)でデータを配布



ルーティング：ブロードキャスト

- ✓ フラッディングベース(Epidemic Routing)でデータを配布



15

プロトタイプ実装

- ✓ Linux上のアプリケーションとして実装
- ✓ アドホックネットワークのルーティングプロトコルとしてOLSRを利用

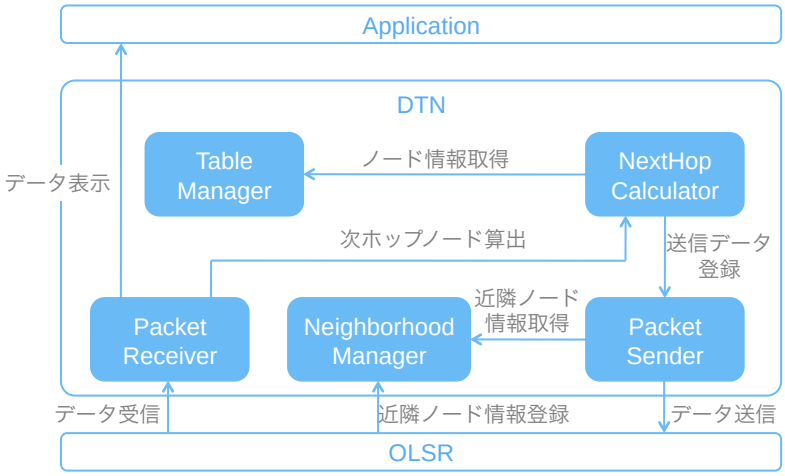
- ✓ OLSR
  - プロアクティブ型のルーティングプロトコル
  - OLSRのLinux実装としてolsrdを利用

Application DTN	
Transport(TCP, UDP)	
OLSR	IP(IPv6)
DataLink	
Physical	

16

モジュール構成

- データの中継処理



おわりに

- 今後の予定
  - シナリオに沿った実験を行い、動作確認 ＋ 性能評価
  - シミュレータによる手法の評価
- まとめ
  - 既存の災害情報システムの問題点
  - Delay/Disruption Tolerant Networking
  - DTNを用いたデータ配送機構
  - プロトタイプ実装

MANETにおけるDHTを用いたデータベースの横断検索

西原 雄太 †† 横田 裕介 † 大久保 英嗣 †
† 立命館大学情報理工学部 †† 立命館大学大学院理工学研究科

1 はじめに

本研究は，MANETを構成する各ノードがデータベースを持っている環境において，それらを横断的に検索する手法を提案する．従来，このような検索は，フラッディング方式を用いた検索によって実現されている．この方式では，柔軟なクエリを処理することができ，各ノードのスキーマが異なる場合にも対応できる．しかし，発生するトラフィックが膨大になるため，大規模なネットワークを構築できないことや検索結果入手の不確実性が問題となる．このため，提案手法では，DHTを用いることでスケーラブルかつ，検索結果入手の確実性を保証できる検索方法を提案する．

本稿では，2章でインターネット上でノードが保持するデータベースを検索する研究であるPIER(Peer-to-peer Information Exchange and Retrieval) について述べる．3章で提案するMANETにおけるDHTを用いたデータベースの横断検索について述べる．4章で評価方法と評価項目について述べる．

2 関連研究

本章では，インターネット上で構築されているP2Pネットワークにおいて各ノードが保持するデータベースをDHTを用いて検索するシステムであるPIER[1] について述べる．PIERは，各ノード上にあるデータベースが共通のグローバルなスキーマで定義されている環境を想定している．

まず各ノードは，DHTを用いてオーバレイネットワークを構築する．続いてノードは，自身の保持するテーブルの各タプルについて，テーブル名，属性名，属性値をまとめてハッシュ化したものをキーとし，テーブル名，主キー，所持ノードのIPアドレスといったデータの所在情報を値としてDHTに登録する．例えば，ノード (IPアドレス=123.45.67.89) が保持する社員テーブル (ID, 名前, 出身地) に，(012, 佐藤, 奈良) というタプルがある場合は，DHTに対してput(hash(”社員,ID,012”),”123.45.67.89, 社員,012”)，put(hash(”社員, 名前, 佐藤”),”123.45.67.89, 社員,012”)，put(hash(”社員, 出身地, 奈良”),”123.45.67.89, 社員,012”) を実行する [2]．つまり，(”社員, 名前, 佐藤”) や，(”社員, 出身地, 奈良”) の所在情報を，DHT上でそれぞれひとつのノードが管理することになる．

次に，PIERでの検索について述べる．例えば，任意のノードが出身地が奈良の社員の名前を知りたい場合は，get(hash(”社員, 出身地, 奈良”)) を実行する．これに対してDHTは，hash(”社員, 出身地, 奈良”) に登録されているデータの所在情報を返す．つまりノードは，ネットワークに存在する (”社員, 出身地, 奈良”) の所在情報をすべて取得する．ノードは，取得した所在情報から各ノードとマルチキャストで通信し，出身地が奈良の社員の名前を取得する．

3 提案手法

本章では，2章で述べたPIERをMANETに適用することでデータベースを検索する手法について述べる．これにより，スケーラブルで検索結果入手の確実性を保証できる検索を実現する．PIERをMANETに適用する際の問題は，PIERがインターネットを想定して設計されていることにある．具体的には，インターネットを想定しているDHT上に実装されている点と，put関数やget関数によって発生するトラフィックが多い点が問題となる．どちらも，高速で安定した通信が可能なインターネット環境では問題にならないが，無線によって通信するMANETではネットワークへの負荷が高く，パフォーマンスが低下してしまう．

提案手法では，MANETを想定しているDHTであるMADPastry[3] 上に提案手法を実現することで解決する．MADPastryとは，ネットワークをクラスタに分割し，クラスタ内のノードに論理的に近いノードIDを割り振ることにより，ノード間の近接性を考慮したオーバレイネットワークの構築を行うものである．

put関数やget関数によって発生するトラフィックの問題は，put関数の変更と，事前にデータの特徴や発行されるクエリを考慮してデータベースをDHTに効率的に登録することで解決する．以下，put関数の変更とデータベースのDHTへの効率的な登録方法について述べる．

3.1 put関数の変更によるテーブル単位での登録

発生するトラフィックを削減するためにPIERのput関数を変更する．PIERのput関数は，put(hash(”社員,ID,012”),”133.19.1.225, 社員,012”) のように，タプルごとに所在情報をDHTに登録している．このため，1つのテーブルをDHTに登録する場合，(タプルの総数 × 属性数) 回のput関数を実行する必要がある．また，タプルの挿入や更新のたびに，属性数回のput関数を実行する必要がある．提案手法では，PIERのput関数をput(hash(”社員,ID,012”),”123.45.67.89, 社員”) のように，テーブルごとに所在情報をDHTに登録するように変更し，put関数の実行回数を削減することで，トラフィックの削減を実現する．

3.2 データベースのDHTへの効率的な登録方法

事前にデータの特徴や発行されるクエリを考慮し，データベースをDHTに効率的に登録することで，発生するトラフィックを削減する．つまり，アプリケーションを設計する際にデータベースのDHTへの登録方法を設計する．

事前に発行されるクエリがわかる場合は，検索で利用しない属性をDHTに登録しない．これは，getすることのないデータの所在情報を登録する必要がないためである．また，事前に発行されるクエリを変更できる場合は，可能な限り検索で利用する属性が少なくなるよう

Database query processing in MANET using DHT
Yuta Nishihara†, Yusuke Yokota† and Eiji Okubo†
†College of Information Science and Engineering, Ritsumeikan Univ.
††Graduate School of Science and Engineering, Ritsumeikan Univ.

に設計し，登録する属性数を削減する．つぎに，提案手法では利用できるハッシュ関数を複数提供する．アプリケーションの設計者は，データベースの属性値の特徴によってハッシュ関数を変更して DHT へ効率的に登録する．以下，データを圧縮して登録する方法，クラスタごとにデータを登録する方法，これら 2 つの登録方法を組み合わせて登録する方法について述べる．

3.2.1 データを圧縮して登録

本項では，いくつかのデータを圧縮，すなわち同じハッシュ値として扱うハッシュ関数を利用して DHT へデータの所在情報を登録する方法について述べる．この方法では，例えば，温度データを DHT に登録する場合，$hash(\text{”テーブル名, 温度, 0”}) = \cdots = hash(\text{”テーブル名, 温度, 9”})$ のように 0 度から 9 度までを同じハッシュ値として扱う．これにより，0 度から 9 度までの温度データを 1 回の put によって登録することができる．また，0 度から 9 度までの温度データの所在情報を 1 回の get によって取得することができ，レンジクエリを効率よく処理することができる．

しかしこの方法は，0 度だけの温度データの所在情報を取得するようなレンジクエリでない処理を実行する場合，効率が悪くなってしまう欠点がある．このため，同じハッシュ値として扱うデータの範囲を，発行されるクエリを考慮して慎重に決定する必要がある．

3.2.2 クラスタごとにデータを登録

本項では，MADPastry によってクラスタに分割されたネットワークを利用し，任意のデータの所在情報を自ノードと同じクラスタ内のノードが管理する方法について述べる．この方法では，任意のデータの所在情報を管理するノードがネットワークに複数 (クラスタの個数分) 存在することになる．ノードは，任意のデータの所在情報を DHT に登録する際，自ノードと同じクラスタ内のノードに登録すればよい．これにより，1 回の put によって発生するトラフィックを軽減できる．また，この方法では，データを回収する際，クラスタごとに存在する任意のデータの所在情報を管理するノードにクエリを転送する．転送されてきたクエリを受け取ったノードは，クラスタ内のデータを回収して結果をまとめてソースノードに送信する．この仕組みにより，データの回収で発生するトラフィックを軽減できる．図 1 の PIER は，クエリを発行したノードが DHT に対して get(hash(”社員, 出身地, 奈良”)) を実行して取得した出身地が奈良の社員の所在情報を元にマルチキャストで各データベースにアクセスし，データを回収している様子の一部を示す．図 1 の提案手法は，クラスタごとに存在する hash(”社員, 出身地, 奈良”) を管理するノードにクエリを転送し，クラスタごとにデータを回収している様子を示す．提案手法では，最後に，hash(”社員, 出身地, 奈良”) を管理するノードが回収したデータを一括してソースノードに送信することでクエリの処理を完了する．

しかしこの方法では，クエリを転送するために，get 関数をクラスタの数だけ実行する必要がある．このため，削減できるトラフィック量が，get 関数の実行回数の増加によるトラフィック量より多くなる属性値にのみ，この方法を適用する必要がある．

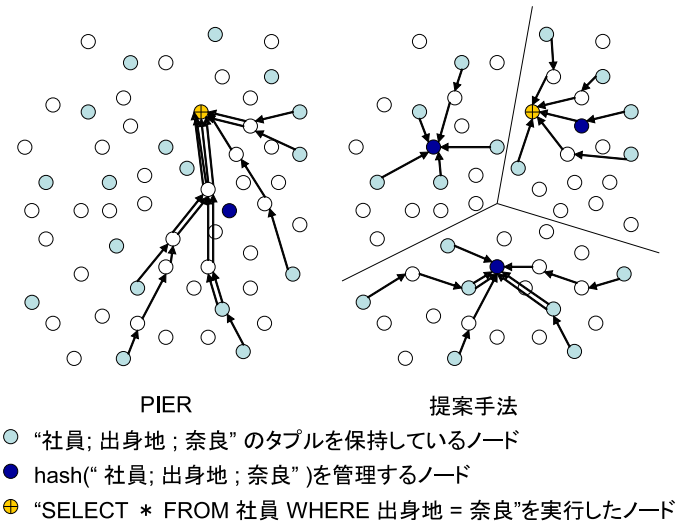


図 1　データの回収

3.2.3 組み合わせ

本項では，3.2.1 項，3.2.2 項で述べた方法を組み合わせて DHT へ登録する方法について述べる．この方法では，まず 3.2.1 項で述べた方法でデータを圧縮し，さらに 3.2.2 項で述べた方法で，自ノードと同じクラスタ内のノードがデータの所在情報を管理するようにする．これにより，2 つの登録方法の利点を得ることができる．

4 評価

本章では，提案手法の評価方法について述べる．評価は，Network Simulator 2[4] にフラッディングを用いたデータベースの検索，PIER，提案手法を実装することで行う．評価項目は，トラフィック量，検索成功率，応答時間についてネットワークの規模，ノードの移動速度を変化させて比較することを予定している．

5 おわりに

本稿では，MANET を構成する各ノードが保持しているノードを検索する手法について述べた．今後は，4 章で述べたように実装とシミュレーションによる評価を行う予定である．

参考文献

[1] R. Huebsch, J.M. Hellerstein, N. Lanham, B.T. Loo, S. Shenker, and I. Stoica. Querying the Internet with PIER. In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pp. 321–332. VLDB Endowment, 2003.

[2] 佐藤麻美, 渡辺知恵美. P2P を利用した地球物理データのネットワーク横断検索・共有システムの実現に向けて. 第 18 回データ工学ワークショップ (DEWS2007), D, Vol. 1, .

[3] T. Zahn and J. Schiller. MADPastry: A DHT substrate for practicably sized MANETs. In *Proc. of ASWN*, 2005.

[4] The Network Simulator - ns2. `http://www.isi.edu/nsnam/ns/`.

MANETにおけるDHTを用いたデータベースの横断検索

立命館大学 理工学研究科
大久保・横田研究室
M1 西原 雄太

1

目次

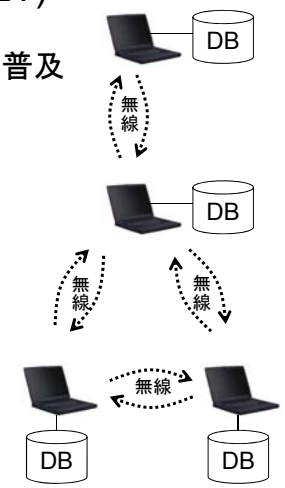
- はじめに
  - MANETにおけるデータベースの横断検索
- DHTを用いたデータベースの横断検索
  - PIER
- MANETにおけるDHTを用いたデータベースの横断検索
  - MANETを想定しているDHTを利用
  - 事前にデータベースや発行されるクエリを考慮して，DHTへの登録方法を設計
    - クラスタごとにデータを管理
    - データを次元圧縮して登録
- 評価
  - 評価方法・項目の検討
- おわりに

2

はじめに（1）

- モバイルアドホックネットワーク(MANET)が注目
  - 高機能な移動端末（ノード）の低価格化と普及
- MANETとは
  - ノード同士が無線により自律的に構成
  - サーバが存在しない
    - P2Pネットワーク

→ノードがデータベースを保持
  - ノードにセンサが搭載
  - さまざまな情報が格納
  - データベースを横断的に検索したい



はじめに（2）

- 目的
  - MANETを構成しているノードが保持しているDBを横断的に検索する
- 既存の方法：フラッディングを利用
  - 利点
    - 実装が簡単
    - 柔軟なクエリやDBのスキーマに対応できる
  - 欠点
    - 大規模なネットワークに対応できない
    - 検索結果入手の確実性を保障できない

→分散ハッシュテーブル（DHT）を用いることで解決

4

既存研究：PIER

- P2Pネットワーク上のノードが持つDBをDHTを用いて検索するシステム
  - インターネットスケールで動作する大規模分散クエリエンジン
  - 検索結果入手の確実性を保障
- DBのすべてのタプルをDHTに登録
  - テーブル名，属性名，属性値をまとめたハッシュ値にタプルの所在情報を登録
    - 各ハッシュ値を管理するノードがタプルの所在情報を管理
  - 社員テーブルに「ID=012,名前=佐藤,出身地=奈良」というタプルがある場合
    - put (hash(“ 社員; ID ; 012”) , “ 123.45.67.89; 社員; 012”)
    - put (hash(“ 社員;名前 ;佐藤”) , “ 123.45.67.89; 社員; 012”)
    - put (hash(“ 社員;出身地 ;奈良”) , “ 123.45.67.89; 社員; 012”)

5

PIERでの検索

- get関数
  - DHTに登録してあるタプルの所在情報を取得
    - get(hash(“社員;出身地;奈良”))→“123.45.67.89;社員;012”, …
  - ハッシュ値を管理するノードがタプルの所在情報を返す
    - ネットワークに存在する”社員;出身地;奈良”のタプルの所在情報をすべて取得できる
- クエリの処理
  1. “SELECT ＊ FROM 社員 WHERE 出身地 = 奈良” というクエリが発行
  2. DHTに対して，get(hash(“社員,出身地,奈良”))を実行
  3. タプルの所在情報を使って，マルチキャストで各DBにアクセス

提案手法

- PIERの仕組みをMANETに応用
  - 大規模なネットワークに対応できる
  - 検索結果入手の確実性を保証できる

  - 実装が難しい
  - 柔軟なクエリやスキーマに対応できない

- 問題
  - ✓インターネットを想定しているDHTを利用
    - 物理ネットワークを考慮していない
  - ✓発生するトラフィックが多い
    - (タプルの総数×属性数)回の登録(put)が必要
    - タプルの挿入のたびに属性回の登録が必要

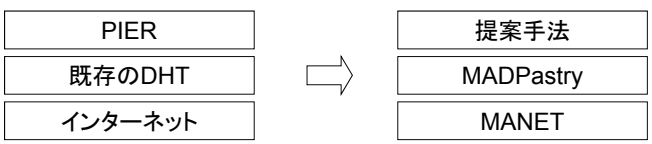
7

PIERの問題点と解決方法(1)

- インターネットを想定しているDHTを利用
  - 物理ネットワークを考慮していない

→MANETを想定したDHT(MADPastry)を利用

- MADPastry
  - 物理ネットワークを考慮してトポロジを構築するDHT
    - 無駄なホップを削減
  - 物理的に近いノードに論理的に近いノードIDを割り振る
    - ネットワークをクラスタに分割
    - クラスタ内のノードに論理的に近いノードIDを割り振る



8

PIERの問題点と解決方法（2）

- 発生するトラフィックが多い
  - (タプルの総数×属性数)回の登録（put）が必要
  - タプルの挿入のたびに（属性数）回の登録が必要

→DHTにDBを効率的に登録

- put回数の削減
  - put関数を変更
- 事前にDBや発行されるクエリを考慮
  - 登録する必要のない属性は登録しない
  - 属性値の特徴によってDHTへの登録方法を変更

9

put回数の削減

put関数を変更

- PIER
  - put (hash(“ 社員;名前;佐藤”) , “ 123.45.67.89; 社員;012”)
  - タプルの所在情報
    - タプルごとにDHTに登録
- 提案手法
  - put (hash(“ 社員;名前;佐藤 ”) , “ 123.45.67.89; 社員”)
  - テーブルの所在情報
    - テーブルごとにDHTに登録
- 欠点
  - getした段階で，タプル数を把握できなくなる

10

事前にDBや発行されるクエリを考慮

- アプリケーションによってDHTへの登録方法を設計
- 登録する必要のない属性は登録しない
  - 検索で利用しない属性は登録しない
  - 発行されるクエリを変更
    - 検索で利用する属性数を減らす

  →put回数の削減
- 属性値の特徴によってDHTへの登録方法を変更
  - 利用するハッシュ関数を変更
    - 3つ提案

11

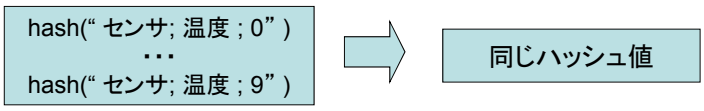
DHTへの登録方法

1. データを圧縮して登録
   - いくつかのデータを同じハッシュ値として扱う
     - put回数を削減
     - get回数を削減
2. クラスタごとにデータを登録
   - 自ノードと物理的に近いノードがデータを管理
     - 1回のputで発生するトラフィックを削減
     - データ収集の効率化
3. 組み合わせ

12

データを圧縮して登録

- いくつかのデータを同じハッシュ値として扱う
  - 例：温度データ
    →0度～9度までをひとつのノードが管理

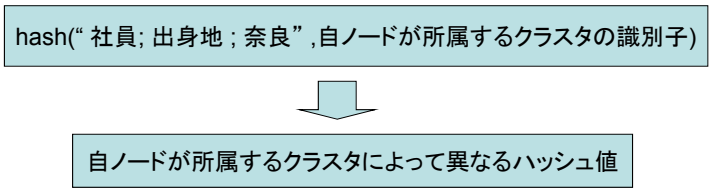


- 利点
  - 1回のputで0度～9度までのデータを登録
    - PIERでは10回のputが必要
  - レンジクエリを効率よく処理できる
    - 1回のgetで0度～9度までのデータの所在情報を取得
- 欠点
  - レンジクエリでないクエリの処理では効率が悪い
  - 適切なハッシュ関数を設計する必要がある
    - 発行されるクエリを考慮

13

クラスタごとにデータを登録 (1)

- 自ノードが所属するクラスタ内のノードがデータを管理

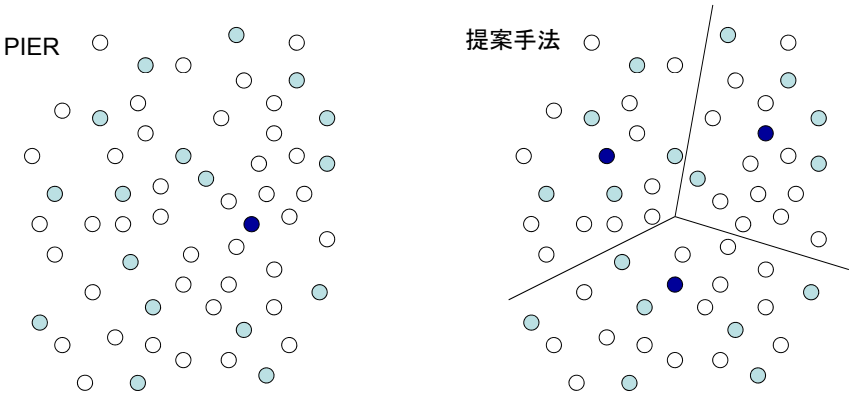


- 利点
  - 1回のputで発生するトラフィックを削減
    - 物理的に近いノードに登録
  - データ収集の効率化
    - クラスタごとにデータを収集
- 欠点
  - 検索時にget回数がクラスタの数だけ必要
    - （ getによって増加するトラフィック量 > 削減できるトラフィック量 ）にならないように設計

14

クラスタごとにデータを登録 (2)

- 1回のputで発生するトラフィックを削減
  - 物理的に近いノードに登録

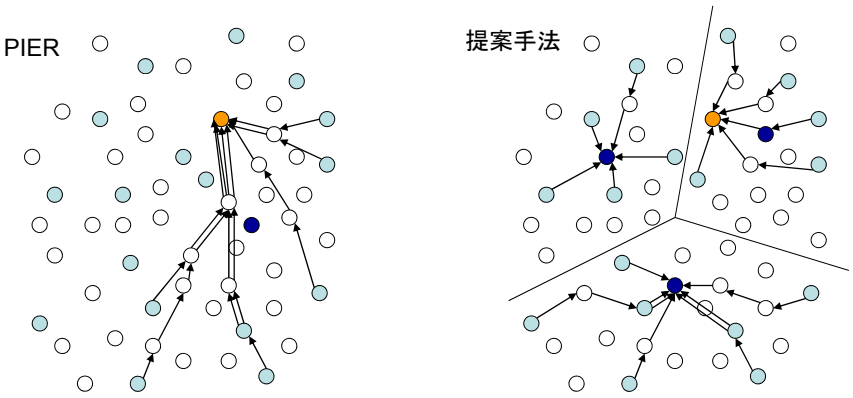


- “社員; 出身地 ; 奈良” のタプルを保持しているノード
- hash(“ 社員; 出身地 ; 奈良”)を管理するノード

15

クラスタごとにデータを登録 (3)

- データ収集の効率化
  - クラスタごとにデータを収集する仕組みを加える



- “社員; 出身地 ; 奈良” のタプルを保持しているノード
- hash(“ 社員; 出身地 ; 奈良”)を管理するノード
- “SELECT ＊ FROM 社員 WHERE 出身地 = 奈良”を実行したノード

16

組み合わせ

- 2つのDHTへの登録方法を適用
  - データの圧縮＋クラスタごとにデータを登録
    - データを圧縮してから，自ノードが所属するクラスタの識別子を組み込む

17

評価

- PIER
  - シミュレーションで評価
    - GT-ITM（ Georgia Tech Internetwork Topology Models ）を利用してトポロジを生成
- MADPastry
  - シミュレーションで評価
    - Network simulator 2 に実装
- 提案手法
  - NS2に実装を予定
    - DBのアクセス時間は無視してトラフィックを中心に比較
    - ファイルシステムに近い形で実装

18

評価対象と評価項目

- 比較対象
  - フラッディングベースのDBの横断検索
  - PIER
- 比較項目
  - トラフィック量
  - 検索成功率
    - 検索結果入手の確実性
  - 応答時間

  - ネットワークの規模
  - ノードの移動速度

19

おわりに

- MANETにおけるDHTを用いたデータベースの横断検索
  - MANETを想定したDHTを利用
  - データベースのDHTへの登録を効率化
    - put回数の削減
    - 3つの登録方法を提案
- 今後の予定
  - 関連研究の調査
  - シミュレーションによる評価
    - NS2に実装を進める

20

複数の無線基地局を用いたQoS制御システムにおける基地局切替えアルゴリズム

川口　雄二郎 †　　毛利 公一 ††
† 立命館大学大学院理工学研究科　†† 立命館大学情報理工学部

1　はじめに

近年の無線通信では, VoIP(Voice over Internet Protocol) や動画像のストリーミングなどのリアルタイム性が高く, かつ大容量な通信を利用する機会が増えている. しかし, IEEE802.11 a, b, g などの方式では, 通信の品質である QoS(Quality of Service) が保証がされておらず, 動画像などを実際に利用しようとすると, 通信品質が低下する可能性がある. 無線 LAN で QoS 制御を実現するための規格として, IEEE 802.11e がある. IEEE 802.11e は, 2 つの機能を有する. 1 つが EDCA(Enhanced Distributed Channel Access) で, 待機時間を調整することで, 優先度の高いデータを先に送ることを目的としている. しかし, ネットワークが高負荷状態になると, QoS 保証が困難になる. もう 1 つの方式が, HCCA(Hybrid Coordination Function Control Channel Access) である. これは, 優先度の高いデータに通信時間を割り当て, その時間は確実にデータ送信を可能とする. しかし, この方式に対応した機器が少ないのが現状である. よって, 現在でも幅広く使われている IEEE 802.11a, b, g などを利用し, QoS 保証を実現する方式を提案する [1].

通信品質の低下が起きる原因として, 1 台の無線基地局に対して複数の無線端末が同一のチャネルを利用して通信するため, 複数の無線端末が通信待ち状態に陥ることが挙げられる. これを回避するための手段として, 複数の無線基地局を配置し, QoS を考慮しつつ, 基地局と端末の組合せを適切に分散させることによって, 全体のスループットを向上させる.

本稿では, 2 章で想定環境について, 3 章で本システムにおける想定環境での処理手順について, 4 章では無線基地局と無線端末の機能について, 5 章では適応的接続制御アルゴリズムについて, 6 章で適応的接続制御アルゴリズムの実装について述べ, 7 章で本稿のまとめとする.

2　QoS 制御システム

本システムでは, 図 1 に示すように, 特定の範囲内に複数の無線基地局を配置する. 複数の無線基地局が近くにある場合では, 通信範囲が重なり合うため, 異なるチャネルを使用する. 一方で, 範囲が重ならない場合は, どのチャネルを用いても構わない. ESSID は, 基地局で同一にする. 基地局間同士のデータの通信は, Ethernet を用いる.

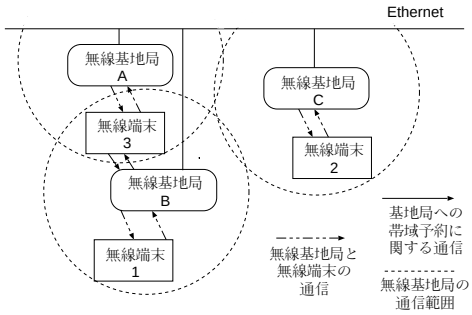


図 1　QoS 制御システムの想定例

3　QoS 制御システムの処理手順

図 1 では, 基地局が A 〜 C の 3 台あり, 端末が 1 〜 3 の 3 台あるとする. さらに, ある時点で, 基地局 B は端末 1 が, 基地局 C と端末 2 が通信を行っているとする. このとき, 端末 3 が基地局 B に対して, QoS が保証された通信を要求したとする. 基地局 A が端末 3 の要求を満たせるにもかかわらず, 端末 3 が要求を満たせない基地局 B へ要求を出しているとすると, システム全体での効率が低下することになる. このような場合において, システム全体の効率を下げないように処理する手順は以下である.

1. 端末 3 が基地局 B に対して, 帯域予約を行う.
2. 帯域予約を受けた基地局 B は, 代表基地局と呼ばれる, 基地局や端末から送信された情報を管理する基地局に対して, 要求内容を Ethernet を用いて送信する.
3. 代表基地局は, 端末 3 の帯域要求に応えられる基地局 A に通信を切替えるように, 端末 3 に対して指示を出す.
4. 無線端末 3 は, 無線基地局 B から無線基地局 A に切替え, 要求した帯域での通信を行う.

以上の処理を行うことで, 特定の基地局に対して QoS が保証された通信の要求が集中しても, 代表基地局が基地局の情報や端末の情報を随時収集を行い, 指示を出すことで, システム全体の効率を向上させることができる.

4 無線基地局と無線端末の機能

帯域を保証した通信を実現するためには, 基地局だけではなく, 端末にも帯域保証のためのシステムを構築する必要がある. 端末が周辺基地局を探索し, ある特定の基地局に帯域予約をする. その基地局が帯域予約情報を代表基地局に対して送信し, 代表基地局がその情報を基に, 接続先基地局を決定する.

端末と基地局では, 主に以下の機能が必要とされる.

- 帯域予約

  ユーザ毎にアプリケーションのプロセスで利用する帯域は異なるため, プロセス別に基地局に帯域の要求予約を行う. 要求帯域を満たせる基地局がフロー制御機構のパケットスケジューラに, 帯域の予約を行う. 端末もパケットスケジューラを用いて, 帯域を予約する.

- フロー制御

  端末と基地局の双方でパケットスケジューリングを行うことで, 基地局から端末だけの通信だけではなく, 端末から基地局への通信にもフロー制御を行い, 端末が予約した帯域の保証を実施する. これにより, 刻一刻と変動し続ける帯域の保証を実現する.

- 通信状況の収集

  基地局と端末がフロー制御を行うために, 通信状況を収集する必要がある. 基地局では, 接続されている端末のプロセスのパケットを監視し, 通信状況を収集する. 端末では, 送受信しているプロセスのパケット, 接続が可能な基地局の電波品質を監視し, 定期的に基地局に送信する.

これらの機能は, 帯域を保証した通信を行うために必要な共通した機能である. 4.1 では, 端末固有の機能について, 4.2 では, 端末や基地局の情報を一括して管理する, 代表基地局を説明する. 4.3 では, 接続先基地局の切替え回数の課題について, 述べる.

4.1 端末における機能

端末側の詳細な機能は, 以下である.

- 接続可能な基地局の調査

  基地局の電波品質と MAC アドレスを無線 LAN カードから取得する. ここで取得した情報は, 基地局へ送信することにより, 基地局が端末へ最も適した基地局への切り替え指示を可能とする.

- 基地局への帯域予約

  基地局へ対して, 帯域の予約を行うため, 必要な情報を送信する.

- 基地局への収集情報の送信

  接続可能な基地局, プロセス毎の通信状況の情報を, 基地局に送信する.

4.2 代表基地局

各基地局で収集したプロセス毎の通信状況, 無線端末からの帯域要求などは, 代表となる基地局を選出し, それら情報を 1 台の基地局で管理することで, QoS が保証された通信を可能とする. 代表基地局の機能を, 以下に示す.

- 代表基地局による切替え先基地局の決定

  各基地局からのプロセス毎の通信状況, 端末からの電波品質などから, 端末が要求した帯域を保証可能な基地局を, 決定する.

- 基地局への帯域予約

  基地局に対して代表基地局は, 帯域を予約するように, 指示を出す. 指示を受けた基地局は, パケットスケジューリングアルゴリズムを用いて, 帯域の保証を行う.

- 端末への切替え先基地局情報の送信

  端末から帯域要求が行われた場合, 端末の接続先基地局が要求に応えられないとすると, 接続先を切替える必要がある. 代表基地局は, 端末に対して, 接続先基地局を切替えるように, 命令を出す.

代表基地局以外の無線基地局は, プロセス毎に収集した通信状況を代表基地局に対して送信する.

5 適応的接続制御アルゴリズム

システム全体での効率を高めるため, 代表基地局は端末に対して接続先基地局の切替えを指示する. しかし, 過度な切替え指示は, 接続の断絶が増大し, 遅延や帯域の低下によるシステムの性能低下が起こる. そのため, 全体のシステム効率を考えた場合, 切替え基準には以下の 2 通りが挙げられる.

- 切替え回数を抑える方式
- 各基地局への負荷の低下を優先する方式

これら 2 通りのアルゴリズムを以後, 適応的接続制御アルゴリズムと呼ぶ. 適応的接続制御アルゴリズムでは, 代表基地局が基地局, 端末から収集した通信状況などの情報を基に求めた割付時間割合を使用する. 割付時間割合については, 5.1 で述べる. 適応的接続制御アルゴリズムについては, 5.2 で述べる.

5.1 割付時間割合

ある端末が特定の基地局と通信を行う場合, 同一の基地局, 同一のチャネルを利用した, 別の端末が存在していると, 別の端末が通信を行っている間, 通信の見合わせが起きる. 周囲に存在する端末の影響を極力受けないようにするため, 割付時間割合を本システムでは用いる. 割付時間割合を求める式は, 式 1 である. 端末が基地局に対して帯域の予約が可能かどうかを, 割付時間割合の

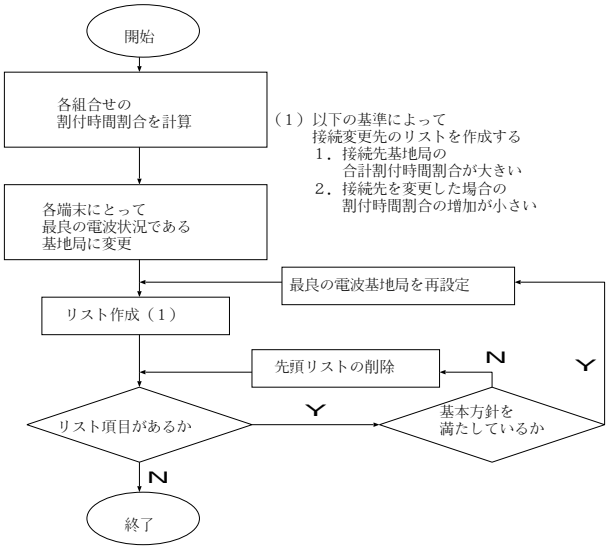


図 2　基地局への負荷の低下を優先する方式

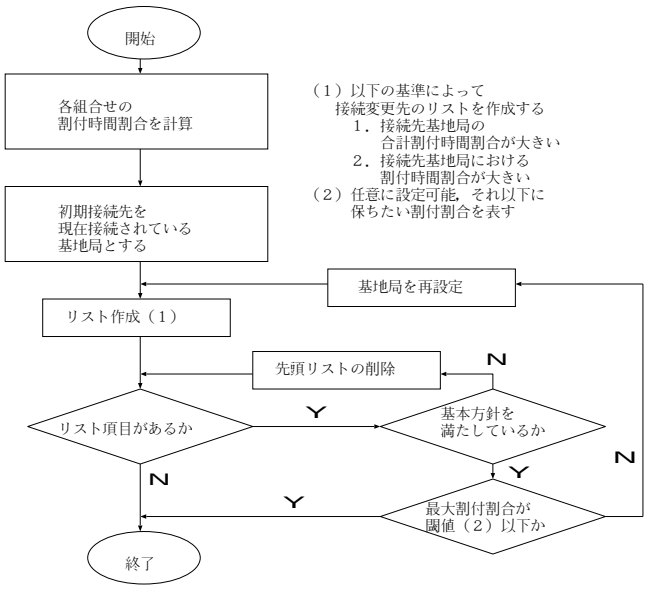


図 3　切替え回数を抑える方式

値によって決める．割付時間割合は, 端末が接続可能か基地局の電波品質から求める．ユーザから予約された帯域を R, 電波品質から求まる最大通信速度を f(Q), 割付時間割合は C と表す．

$$C = \frac{R}{\mathrm{f}(Q)} \cdot 100 \tag{1}$$

5.2　アルゴリズム

2 つのアルゴリズムの基本方針は, 以下である．

- 時間割合の合計の大きい基地局から小さい基地局へ切替えを行う．
- 切替え元の時間割合の合計以上に, 切替え後の切替え先の割付時間割合の合計を越す場合は, 切替えを実行しない．

各基地局の割付時間割合を低く抑えることを優先する方式について, 図 2 に示す．この方式は, 端末と基地局の組合わせの割付時間割合を計算し, 各端末の接続先を割付時間割合の最も小さい基地局に変更する．基地局の切替えは, アルゴリズムの終了後に行う．以下の優先順位で, 接続変更先をソートし, リスト化する．

1. 接続先基地局の割付時間割合の合計が大きい．
2. 接続先を変更した場合に, 割付時間割合の増加が小さい．

作成したリストの先頭から順に, 方針を満たしているかチェックし, 満たしているものは, 接続先の基地局を切り替える．その後, 再度リストを作成し, 満たしているものを, 切り替える．

切替え回数を抑える方式については, 図 3 に示す．端末と基地局の組合わせの割付時間割合を計算し, 各端末の接続先を, 現在接続されている基地局とする．そのため, 最終的な接続先は, 現在接続されている基地局によって変化する．以下の優先順位で, 接続変更先をソートし, リスト化する．

1. 接続先基地局の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．

作成したリストの先頭から順に, 方針を満たしているかチェックし, 満たしているものは, 接続先を変更し, 一番大きい割付割合が任意に設定可能な閾値以上であれば, 再度リストを作成する．一番割付割合が閾値以下となった場合, リストに方針を満たすものがなかった場合, 接続先を切替える．

6　実装

端末の接続先基地局を決定するアルゴリズムの 1 つである, 基地局への負荷の低下を優先する方式について実装行った．本アルゴリズムの処理手順, 端末のサンプルデータを与えた場合の結果と考察をこの章では述べる．各端末の情報について, 表 1, 表 2 に示す．

6.1　処理手順

本アルゴリズムの処理手順について以下に示す．

1. 各基地局と端末の組み合わせについて, 電波品質と要求帯域から割付時間割合を求める．今回の実装では割付時間割合を 100 - 電波品質とした．
2. 端末が通信可能な基地局のなかで最も良い電波品質をもつ基地局を初期接続先基地局とする．

表 1　端末別情報

端末名	AP1 の電波品質	AP1 の割付時間割合	AP2 の電波品質	AP2 の割付時間割合
端末 1	20	80	54	46
端末 2	90	10	71	29
端末 3	89	11	87	13
端末 4	80	20	57	43
端末 5	77	23	90	10
端末 6	50	50	-	-
端末 7	52	48	90	10
端末 8	-	-	68	32
端末 9	-	-	45	55
端末 10	50	50	-	-

表 2　端末別情報 (続き)

端末名	AP3 の電波品質	AP3 の割付時間割合	AP4 の電波品質	AP4 の割付時間割合
端末 1	55	45	-	-
端末 2	91	9	77	23
端末 3	86	14	44	56
端末 4	-	-	67	33
端末 5	84	16	75	25
端末 6	-	-	20	80
端末 7	73	27	68	32
端末 8	70	30	-	-
端末 9	22	78	62	38
端末 10	-	-	20	80

3. 接続先基地局の合計割付時間割合が大きく, 接続先を変更した場合の割付時間割合の増加が小さい順にソートし, リストを作成する.

4. リスト項目が存在する場合, 基本方針を満たしているか判別する. 満たしていない場合先頭の組み合わせを削除し, 次の組み合わせについて判別する. 満たしている場合, 再度 (3) の処理を実行する. リスト項目が存在しない場合, 本アルゴリズムを終了する.

6.2　考察

各端末の情報を与えた場合に, 最良の電波品質をもつ基地局を端末の接続先とした場合における基地局の合計割付時間割合, 基地局に接続している端末について図 4 に示している. 初期の合計割付時間割合が最も多い基地局は AP3 である. また, AP3 に接続している端末のなかで最も端末の割付割付時間割合の増加が小さいものは, 端末 1, 端末 2 となっており, 端末 8 はその次に割付時間割合の増加が小さい端末となっている. 次に, アルゴリズムの基本的な方針を通過した後, 再度リストの作成を行った各組み合わせは図 5 である. AP3 に接続していた端末 8 が接続先を AP2 に変更することで, 最も合計割付時間割合が多い基地局が AP3 から AP1 に変わっている. もう 1 度, 基本的方針を通過した後に, 再度リストを作成した後の組み合わせは図 6 である. 基地局 AP1 から AP2 に端末 3 が接続先を切替えたことで, 基地局 AP1 の合計割付時間割合が減少し, 基地局 AP2 の合計割付時間割合が増加している. その後, リスト項目のチェックをするが基本方針を満たしている組み合わせが存在しないため, 最終的に図 7 となる. 本アルゴリズムを通したことで, それぞれの基地局の合計割付時間割合が初期の組み合わせでの合計時間割合より, 平均化されていることが分かり, 基地局への負担が分散されている.

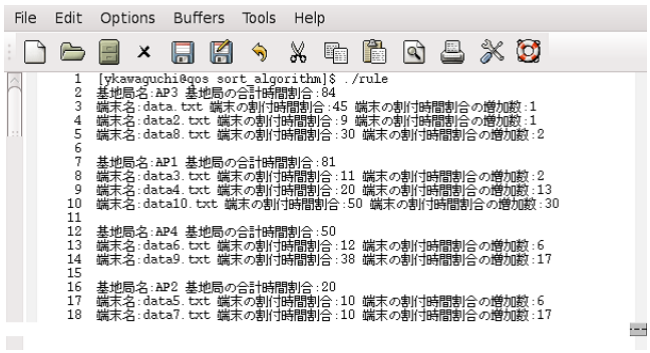


図 4　初期組み合わせ

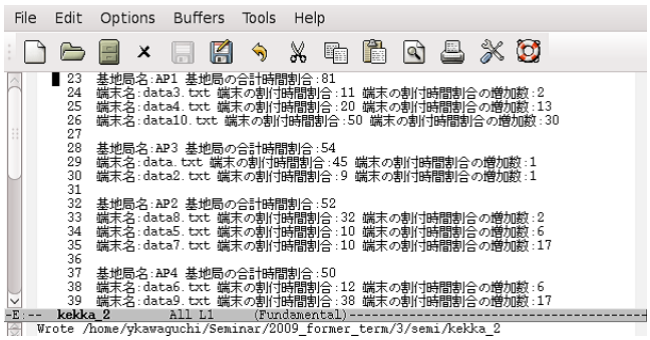


図 5　1 度目の基本方針通過後の組み合わせ

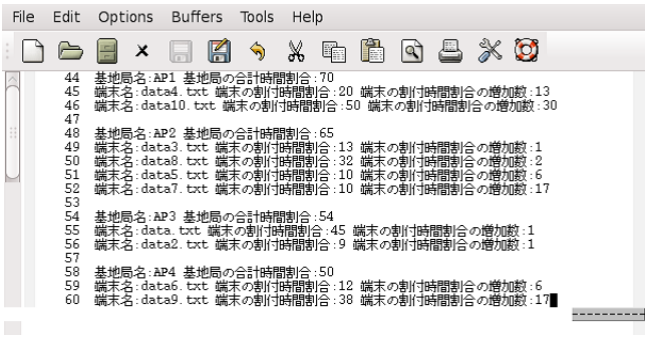


図 6　2 度目の基本方針通過後の組み合わせ

7　おわりに

本稿では, 無線基地局と端末が協調して, 帯域を保証した通信を行う, QoS 制御システムについて述べた. 今後は, 切替え回数を抑える方式の実装, パケットのスケジューリングの検討が挙げられる.

参考文献

[1] 水野　邦彦, 伊藤 淳, 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適応的接続制御アルゴリズム，マルチメディア，分散，協調とモバイル (DICOMO2007) シンポジウム，pp.1614-1621, 2007.

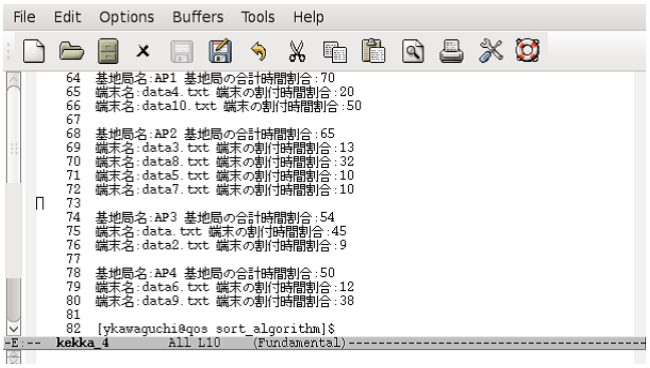


図 7　最終的な組み合わせ

複数の無線基地局を用いた QoS制御システムにおける 基地局切替えアルゴリズム

立命館大学大学院
毛利研究室
川口 雄二郎

発表内容

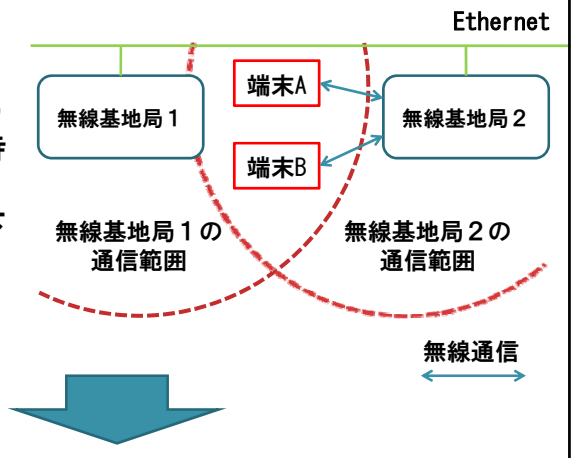
- はじめに
- QoS制御システム
- 帯域予約の通信順序
- 電波品質と通信速度
- 適応的接続制御アルゴリズム
- 基地局の負担を低く抑える方式の評価
- おわりに

はじめに (1/2)

- VOIP, スカイプなど大容量な通信が増大
  - 基地局の帯域を多く使用
  - 端末によって，通信が利用できない場合あり
- 現在の一般的なIEEE 802.11aなどの無線規格では，QoS(Quality of Service)保証を考えた通信を行わない
  - 優先したい通信を使用できない場合が存在

はじめに (2/2)

- 基地局と通信を行う端末が増加すると待機時間が長くなり，スループットが低下

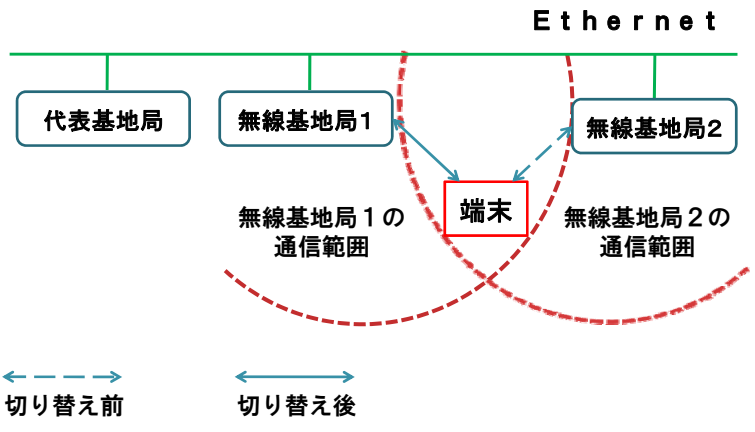


複数の基地局が協調することで全体のスループット低下を防止するシステム

QoS制御システム (1/2)

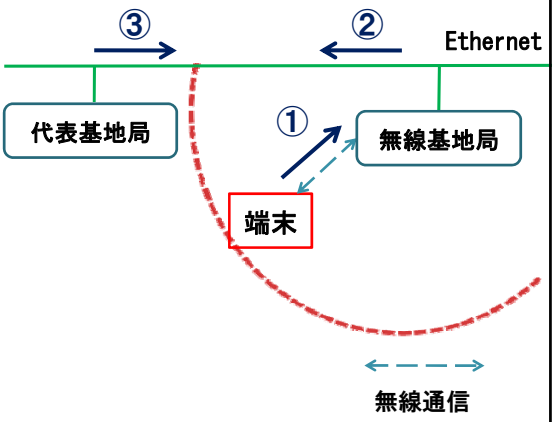
- 端末の各基地局に対する電波状況を収集
  - 基地局は端末の各基地局に対する電波状況を収集し，端末の接続先基地局の決定に利用(代表基地局)
- 基地局同士が通信情報を共有
  - 基地局が現在行っている通信に関する情報を共有することで，システム全体の通信を把握できる(代表基地局)
- 端末が帯域を予約
  - 端末は基地局に対し帯域を予約
  - 基地局は電波状況や帯域要求に応じて端末の接続先基地局を決定(代表基地局)

QoS制御システム (2/2)



帯域予約の通信順序

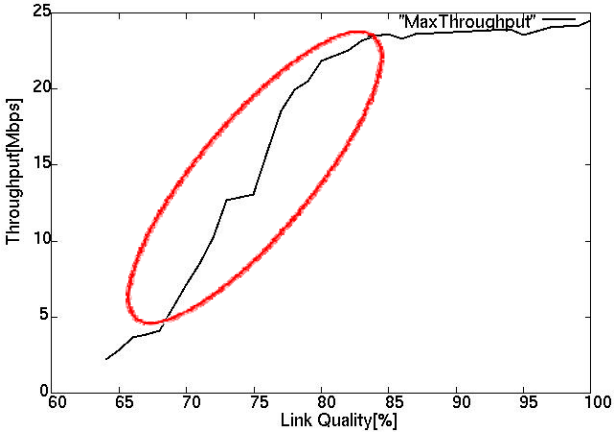
1. 端末は基地局に対し帯域予約を出す
2. 情報を受け取った基地局は代表基地局に情報を送信する
3. 収集していた情報を基に接続先基地局を決定し，その情報を基地局と端末に送信する



電波品質と通信速度

- 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

適応的接続制御アルゴリズム(1/2)

- 代表基地局は，端末の接続可能な基地局の電波品質から，接続先基地局での帯域予約の割付時間割合を求め，予約が可能か判断
  - ユーザからの帯域要求 = R
  - 端末の電波品質 = Q
  - 電波品質から求まる最大通信速度 = $f(Q)$
  - 割付時間割合 = C

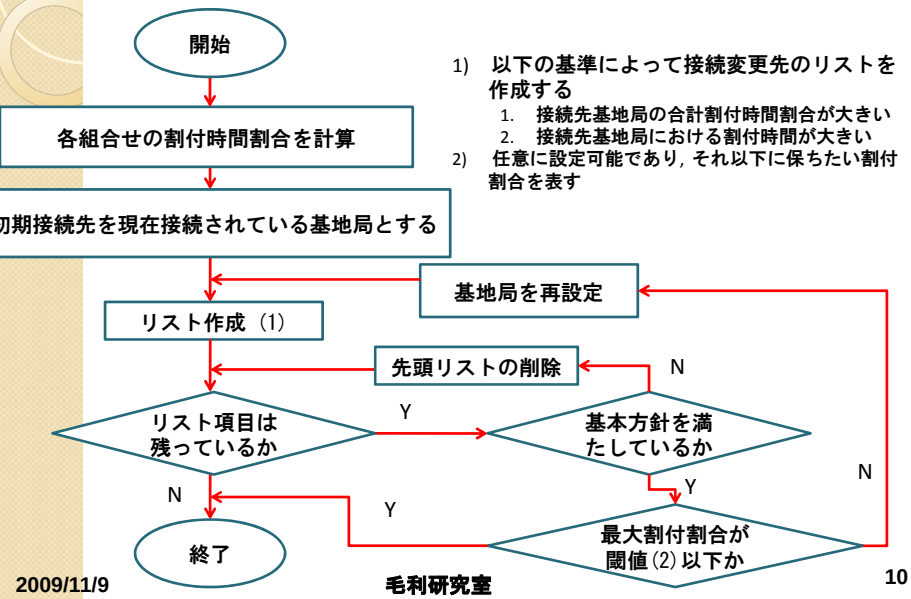
$$C = \frac{R}{f(Q)} * 100$$

- ➢ 要求帯域を時間で予約
- 通信において発生する問題
  - 切り替え回数が増加すると接続の断続時間が増大
  - ある基地局への複数の端末による過度の通信要求

適応的接続制御アルゴリズム(2/2)

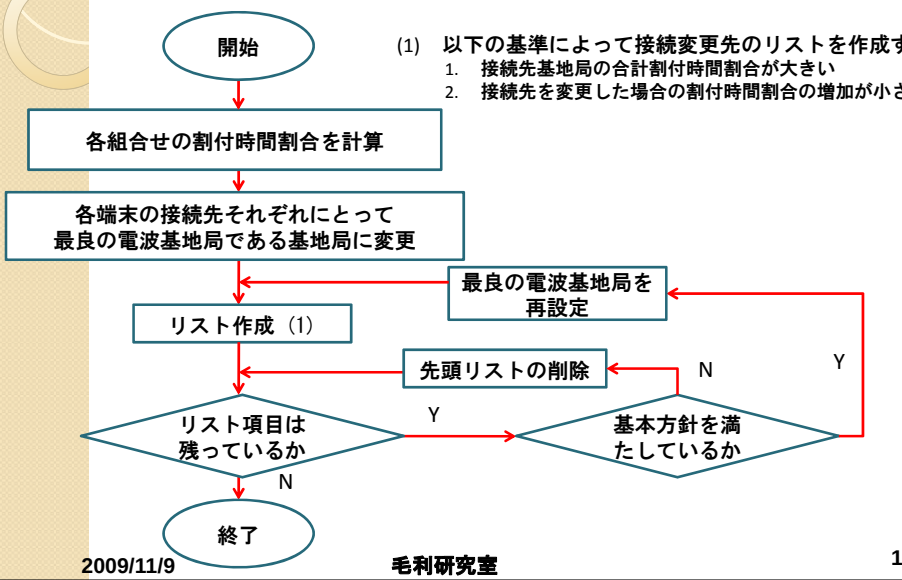
- 2通りのアルゴリズムが考えられる
  - 無線基地局への過度の負担を抑えることを優先する方式
  - 切り替え回数を抑える方式
- アルゴリズムにおける基本方針
  - 基地局の合計の割付時間割合が大きい基地局から小さい基地局へ切替え
  - 切替え後の合計の割付時間割合が切替え前を超す場合，切替えを行わない

切替え回数を抑える方式



1) 以下の基準によって接続変更先のリストを作成する
   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

基地局の負担を低く抑える方式



(1) 以下の基準によって接続変更先のリストを作成する
   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

基地局の負担を低く抑える方式の評価環境

- 以下の環境で動作を検証
  - 基地局がAP1からAP4
  - 端末は端末1から端末10

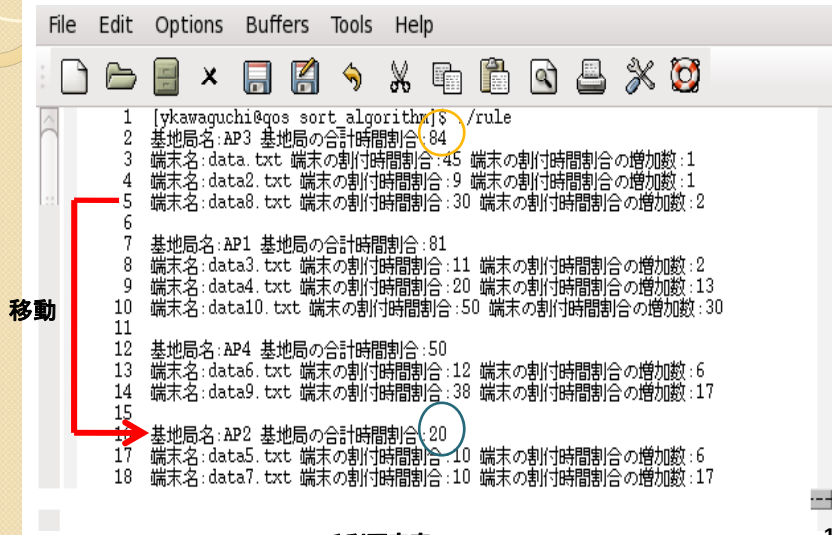
端末別の情報 (1/2)

端末名	AP1の電波品質	AP1の割付時間割合	AP2の電波品質	AP2の割付時間割合
端末1	20	80	54	46
端末2	90	10	71	29
端末3	89	11	87	13
端末4	80	20	57	43
端末5	77	23	90	10
端末6	50	50	–	–
端末7	52	48	90	10
端末8	–	–	68	32
端末9	–	–	45	55
端末10	50	50	–	–

端末別の情報 (2/2)

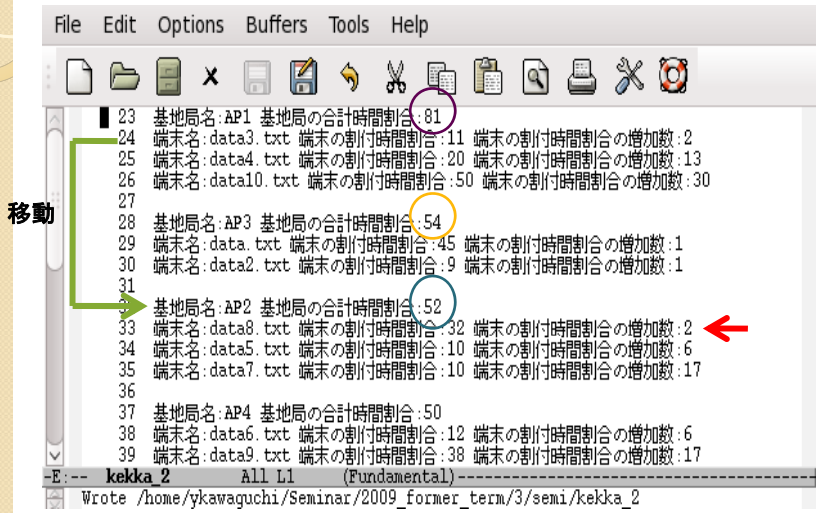
端末名	AP3の電波品質	AP3の割付時間割合	AP4の電波品質	AP4の割付時間割合
端末1	55	45	–	–
端末2	91	9	77	23
端末3	86	14	44	56
端末4	–	–	67	33
端末5	84	16	75	25
端末6	–	–	20	80
端末7	73	27	68	32
端末8	70	30	–	–
端末9	22	78	62	38
端末10	–	–	20	80

リスト作成 (1/4)

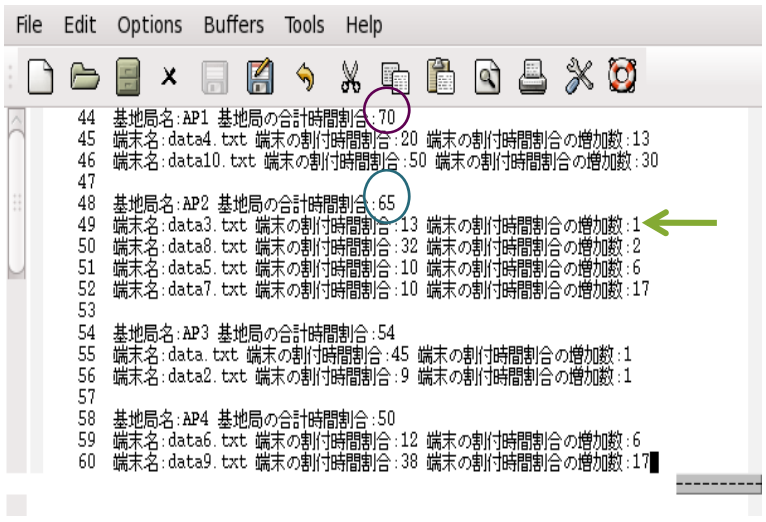


2009/11/9 毛利研究室 15

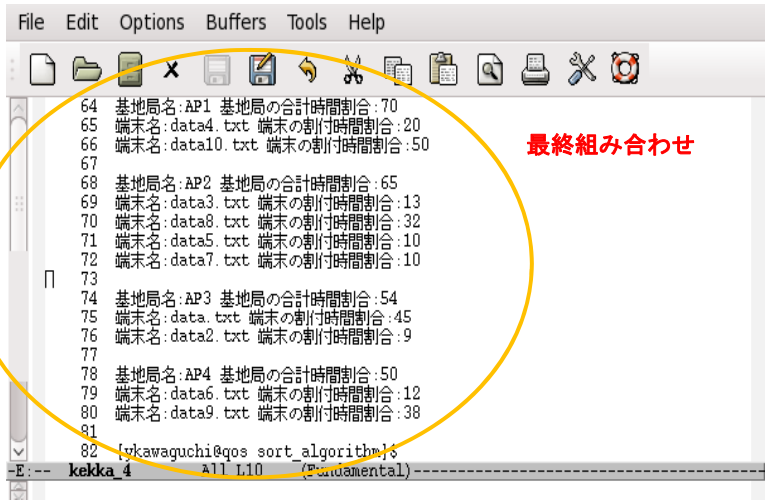
リスト作成（2/4）



リスト作成（3/4）



リスト作成（4/4）



考察

- 総当たりでの組み合わせは55296
- 今回最終的な組み合わせの決定までに，
  2度端末の接続先基地局を変更
- 基地局の合計割付時間割合が最も多いAP1が70，最も合計時間割合が少ないAP4が50とあまり差がない
  - ➢基地局の負担を分散

おわりに

- QoS制御システム
  - 基地局同士が情報を共有
  - 端末が帯域を予約
  - 基地局は予約された帯域を端末に提供
- 基地局の負担を低く抑える方式の評価
  - 初期接続先基地局は電波品質が最も良い基地局
  - 割付時間割合を基に接続先基地局を決定
- 今後の課題
  - 切替え回数を抑える方式の実装
  - パケットスケジューリングの検討

Java による x86 ユーザーモードエミュレータの実装と評価

川口直也[†]

1. はじめに

ネイティブコードからなるユーザーモードアプリケーションの異なる OS、アーキテクチャ間における可搬的利用を阻害する要因として、コンパイル済みバイナリの機械語命令列とシステムコールが挙げられる。

このうちシステムコールについては、機械語命令に互換性のあるアーキテクチャ上であっても、OS が提供する機能に大きく依存するため、ファイルシステムなどの周辺機能も含めてハードウェア機構全体を仮想化するフルシステムエミュレーションによって解決する方法が試みられている[1]。しかし、アプリケーションからは直接利用することのない特権命令や入出力デバイスを含めてエミュレーションするこの方式では、単にユーザーモードの環境を模倣するという観点からは、ユーザ側の負担もオーバーヘッドも大きい。

他方、ユーザーモードにおける機械語命令のエミュレーションのみを行い、システムコールは、その環境のネイティブな OS に対し処理を依頼するユーザーモードエミュレーション[2][3]は、エミュレーションする対象のハードウェア機構が少ないことによるオーバーヘッド低減だけでなく、動作環境の OS 資源をアプリケーションから直接利用出来るという点において有用である。しかしながら、この方式では動作環境が同一のシステムコールライブラリを実装した OS 間に限定されてしまう。

本報告では、ユーザーモードエミュレーションにおけるこの問題を解決し、異種 OS 間における x86/Linux アプリケーションの可搬的利用を目的とした、JavaVM 上で動作するユーザーモードエミューレータとその性能評価を示す。JavaVM 上に実装することでエミュレーション機構自体にも可搬性を付与し、またシステムコールにおいては、Linux 以外の POSIX-API を提供する OS 上において動作するよう非POSIX 準拠のシステムコールのエミュレーションを行った。

2. 目標

筆者は、今日一般的に使われている Linux アプリケーションが本システムの上で動作することを期した。具体的には、I/O 処理やユーザからの入力待ちにおいて大部分の時間を消費し、CPU 命令のエミュレーションによる性能低下の影響が限定的と考えられる、GUI アプリケーションである、OpenOffice や GIMP などを想定している。またこれらのデスクトップアプリケーションがアプレットとして Web ブラウザの上で動作することも期した。このため、エミュレータの実装にあたり、以下の点を目標とした。

- x86/Linux 向けの ELF バイナリを実行可能
- 共有ライブラリをロード可能
- Linux 以外の POSIX 互換環境上で動作する
- 可能な限り Java で記述し
  エミュレータ自身にも可搬性を持たせる
- 一般的な GUI アプリケーションが動作する
- マルチスレッドプログラムが動作する
- Java アプレット化が可能である

3. 設計

図 1 に本エミュレータの構成を示す。x86/Linux 用のユーザーモードプログラムを構成する機械語命令列のうち、内部割込み以外の機械語命令は本エミュレータ上でエミュレーションする。システムコールを利用するための内部割込みは JavaVM の下で動作する JNI(Java Native Interface)モジュールを経由して、実機の OS が提供するシステムコール資源を利用する。これにより、ネットワークやファイル I/O といったハードウェア資源を仮想化することなくアプリケーションから直接利用可能とする。なお、Windows 上では Cygwin のライブラリを使用することで POSIX システムコール機能を利用可能とした。

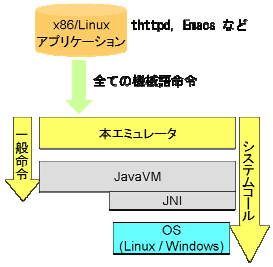


図.1 エミュレータの構成

† 東京農工大学
Tokyo University of Agriculture and Technology

4. 評価

本エミュレータを実装し ls,wc コマンドなどの簡易なプログラムの動作のほか、マルチスレッドや共有ライブラリのロードなどを実現した。さらに xlib をリンクした簡単な GUI プログラムの動作も確認している。

しかし、命令エミュレーションの再現性は完全ではなく、xclock のような複雑なプログラムはまだ動作していない。

以下に本エミュレータの基本性能と今回動作を確認した簡易な GUI プログラムの例とアプレット化の実現について述べる。

4.1 基本性能

本エミュレータ上での性能が、実機上でプログラムを動作させた場合に比べどの程度低速になるかを評価した。対象としたプログラムはフィボナッチ、空の for ループ、ADD 命令を線形に配置した分岐の無いコード、及び/dev/zero に対する read システムコール、/dev/null に対する write システムコールである。

表 1 に本エミュレータ上での各プログラムの実行時性能が、実機で実行した場合に比べ何倍低下したかを示す。

エミュレータ上での分岐の無いコードの実行時性能が特に低下する理由として、本エミュレータでは命令デコード結果をキャッシュして再使用することが挙げられる。

また、read/write システムコールの結果は、本エミュレータの有用性が純粋な演算ではなく I/O 処理において発揮されることを示唆している。

表 1 本エミュレータと実機との性能比較

プログラム	実機に対する性能低下 (N 倍)
フィボナッチ	2 万～7 万
空の for ループ	10 万～14 万
分岐の無いコード	14 万～42 万
read(/dev/zero)	2000～3000
write(/dev/null)	1600～3 万

4.2 GUI プログラム

共有ライブラリの xlib.so を実行時にロードする簡易な GUI プログラムの動作を検証した。図 2 に動作を確認したプログラムの画面を示す。

ウィンドウの拡縮といった基本操作が可能であるが、操作が反映されるまでの時間に大幅な遅延が見られた。

図 2 GUI プログラムの動作画面

4.3 エミュレータのアプレット化

本エミュレータと Java で実装された X サーバを組み合わせることにより、Linux 用の GUI プログラムを Java アプレットとして Web ブラウザ上で動作させることを可能とした。図 3 に動作画面を示す。Windows/IE6 上でマウスからの入力を元に図形が描画されることを確認している。

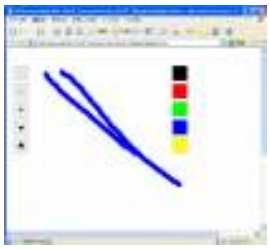


図 3 アプレット化した GUI プログラム

5. 終わりに

x86/Linux 用アプリケーションを JavaVM 上で動作させるためのユーザーモードエミュレータを実装し評価を行った。エミュレータの再現性として GUI プログラムや共有ライブラリのロードが可能である一方、OpenOffice や GIMP といったより一般的なプログラムの動作は確認できていない。また性能面でも改善する必要がある。

今後の展開としては NestedVM[3]のように POSIX システムコールを Java 言語の標準ライブラリでエミュレーションすることで、JNI を使用しない Pure Java なユーザーモードエミュレーションを実現することが挙げられる。

参考文献

1) VMWare http://www.vmware.com/
2) QEMU Project, http://bellard.org/qemu/
3) ALLIET, B., AND MEGACZ, A. Complete translation of unsafe native code to safe bytecode. *In Proceedings of the 2004 Workshop on Interpreters, Virtual Machines and Emulators*, pp. 32–41,2004.

Javaによる x86ユーザーモードエミュレータの 実装と評価

東京農工大学
川口　直也

1

研究背景

問題点

- ネイティブコードからなるアプリケーション(OpenOffice,GIMPなど)の
  バイナリレベルでの異種OS・アーキテクチャ間における可搬的実行は困難
  - 機械語命令列, システムコール, メモリ管理…etc

→ 再コンパイルによる問題解決
  - コンパイラのバージョンやライブラリなど開発環境に対する依存性

→ エミュレータによる問題解決
  - 機械語命令以外の機能をどのように提供するか
  - 性能と汎用性のトレードオフ
    (JITや特殊な命令ディスパッチは
    実装言語・動作環境に大きく依存)

2

既存技術

- **ユーザーモードエミュレーション**
  - 機械語命令(ユーザーモード)のエミュレーションのみ
  - システムコールはエミュレータの下で動作するOSの機能を利用

  ファイルやネットワークI/Oを仮想化せず実機の資源を利用可能

  QEMUの一部機能：
  ターゲットはLinux・ELFバイナリ
  動作環境はLinuxに限定
  （Linux以外の環境で動作しない）

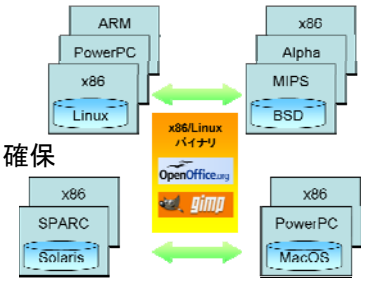
  NestedVM：
  Javaで書かれたエミュレータ
  ターゲットはmips-unknown-elf
  システムコールもJavaでエミュレーション
  マルチスレッドアプリは非サポート

3

目標

- 異種OS・CPU間におけるx86/Linuxユーザーモードアプリの可搬性向上
- VM型の実行環境(Java)向けにエミュレータを実装
  (ターゲットアプリだけでなくエミュレータ自身にも可搬性を付与)
- OpenOffice, GIMPなどの実用的なGUIアプリケーションが動作
- JavaアプレットとしてWebアプリ化できる

- 標準的なLinux向けプログラムが動作する
  - JavaVMが動作する全ての環境での
    x86/Linuxアプリケーションの可搬性の確保

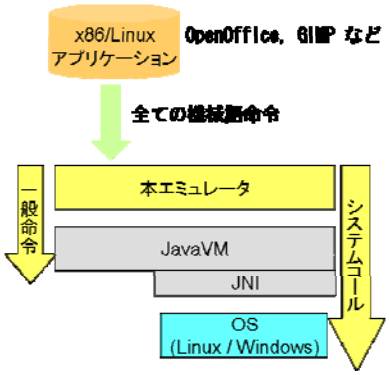


技術的課題

- x86演算資源のJava言語による再現
  - レジスタ,命令セット,メモリ空間, …etc
- システムコール
  - POSIX互換環境でのネイティブコードAPIとの連携
  - Linux独自システムコールのエミュレーション
- ELFバイナリ,共有ライブラリのロード機能
- アプレット化のための機能
  - 本エミュレータ, エミュレーションする対象のプログラム, Xサーバ を配信可能な状態にパッケージング

5

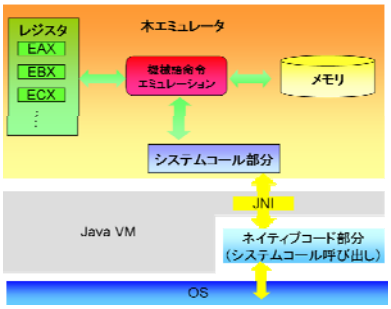
システムの構成

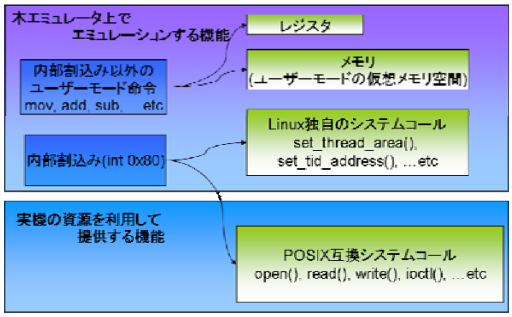


- 仮想的なx86演算資源の提供
  ⇒ Pure Java
- ネイティブなシステムコール資源の活用
  ⇒ JNI
- 非POSIX-APIのエミュレーションによる異種OS間での可搬性確保
  ⇒ Pure Java
- JavaVMによるエミュレータの可搬性確保
  ⇒WindowsではCygwinを利用

6

エミュレータの全体設計



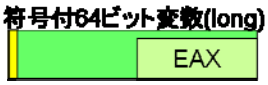


- アーキテクチャに依存しないCPU機能の提供
  → x86のレジスタ/メモリ/命令はJavaVM上で提供
- プロセスメモリ空間の互換性確保
  → 下層で動作するOSから独立したメモリ空間
- ファイル・ネットワークなどに実機の資源を利用
  → JNIによるネイティブなシステムコール資源の活用

7

x86レジスタ/メモリの設計

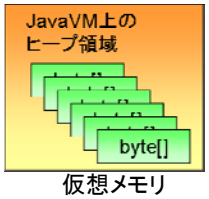
- x86の各レジスタはJava言語の64bit long型変数として確保
  - Javaには符号無し変数が無い.
    32bit値も64bit型変数に格納



仮想レジスタ

レジスタを変数として定義することで
ヒープより高速なアクセス性能が期待できる

- x86のメモリはJava言語上のbyte型配列の集合として確保
- 各byte型配列のサイズは512KBとした



仮想メモリ

JavaVM・ネイティブプロセスから
独立したx86のメモリ空間を提供

8

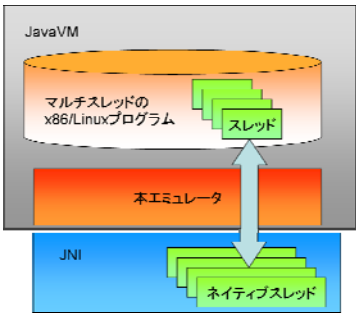
機械語命令の設計

命令種別	例	状況	備考
整数演算	ADD,SUB, MULなど	○	全て実装済み
論理演算	AND,OR,NOT など	○	全て実装済み
比較	CMPなど	○	全て実装済み
シフト/ローテート	SAR,ROLなど	○	全て実装済み
10進算術演算	AAA,AASなど	○	全て実装済み
制御転送	CALL,RET, JMPなど	△	セグメント間ジャンプ などは不要なため未実装
システム	LIDT,STR, ARPLなど	×	ユーザーモードでは不要
ストリング	MOVS,CMPS など	○	全て実装済み
FPU	FLDCW,FILD, FADDなど	△	64ビットまでの演算のみ対応
MMX	MOVQ,PADDB, PANDなど	×	未実装
SSE	MOVAPS,CMPPS など	×	未実装

- 内部割込み以外はJava言語で実装
  - アーキテクチャからの独立性
- 本エミュレータが提供する
  仮想レジスタ・仮想メモリに対し演算を行う
- 浮動小数演算命令はFPU以外未実装
- 各種特権命令は実装しない
  -ユーザーモードプログラムは使用しない

9

マルチスレッドプログラムへの対応



- Linuxネイティブスレッドをpthreadで再現
  - sys_clone()をpthreadライブラリでエミュレート
  - エミュレータ上のプログラムのスレッドが、
    JNI側のネイティブスレッドと対応

10

ELFローダーとスタックの初期化

ELFローダー
- (本エミュレータ単体では)
  静的リンクされたELFバイナリを対象

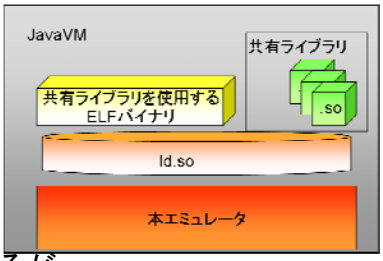
スタックの初期化
- Linuxカーネルがプロセスに渡す定数値

- 環境変数とコマンド引数
  - DISPLAY環境変数など、
    Windows環境に存在しないが必須のものは
    本エミュレータが提供する
  - コマンド引数は本エミュレータのプログラムに対する
    コマンド引数をもとに作成

11

共有ライブラリのロード機能

- ld.soをエミュレータ上で動作させて間接的に対応
  Linuxにおけるローダープログラム
  静的リンクされた単独実行可能なELFバイナリ
  gedit起動時のld.soの使用例)
  ld.so –library-path . /usr/bingedit



本エミュレータのELFローダは
静的リンクされたバイナリのみに対応するが、
ld.soをエミュレータ上で動作させることで
間接的に共有ライブラリのロードを実現

12

実現

- JDK6で55,651行(ネイティブコード部分を含まず)
  ※　wcコマンドをWindows上でエミュレーションして計測した。
- 命令デコード結果をキャッシュして再利用
  - エミュレーションするバイナリが大きくなると性能が低下
  - 使用メモリサイズの大部分を占める
- デコーダーはnasmに付属するndisasmを
  1命令単位でデコードするよう改変したものを利用
- 現状でサポートするOSはLinuxとWindows(Cygwin)
  - 機能を限定すれば(writeシステムコールのみ使用等)
    MacOSX, Solaris上でも動作することを確認済み
  - システムコールで使用する定数値などの変換作業が足かせ
    AF_FILE, PF_UNIX, ….etc

13

達成状況

- 簡易なプログラムの動作
  - ls,wcコマンドなど
  - xlibをリンクした簡易なGUIプログラム
- Javaアプレット化
  - ウィンドウマネージャの無いGUI
- 共有ライブラリのロード
  - 10個以上のライブラリに依存するプログラムのロードを確認
- マルチスレッド
  - 5個までのスレッド生成と同期制御を確認
- シグナル
  - SIGINTによるハンドラの起動を確認

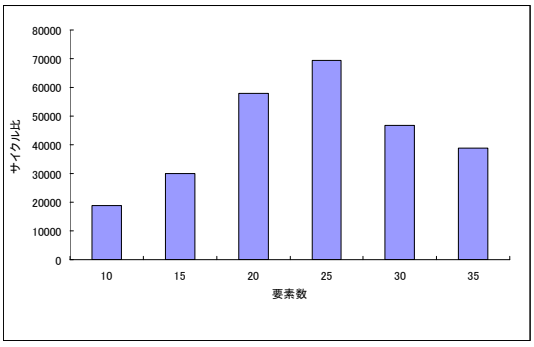
14

基本性能の評価

- 以下のテストプログラムを用いて測定
  - フィボナッチ数列
  - 空のforループ
  - 分岐の無いコード
  - I/O入出力
- 測定環境はVMware上のLinux(Fedora8)
  （ホスト環境はWindowsXP,Core2Quad 2.4GHz)
- 本エミュレータとネイティブ環境の実行に要したサイクル数の比率から性能を測定
  (エミュレータのサイクル数 / ネイティブ環境のサイクル数)
- サイクル数はテストプログラムの内部で取得した
  エミュレータ上でサイクル数を取得する命令が出された場合は
  ネイティブ環境の値を返す仕様とした

15

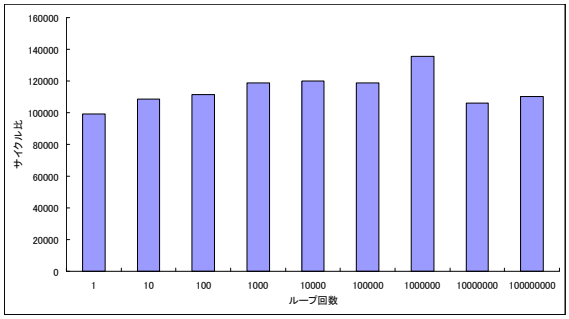
基本性能評価：フィボナッチ数列



- 再帰におけるパフォーマンス
- 本エミュレータはネイティブに比べ
  2万～7万倍程低速

要素数	10	15	20	25	30	35
本システム (サイクル数)	4.47×10^{9}	9.98×10^{9}	6.43×10^{10}	6.66×10^{11}	7.38×10^{12}	8.08×10^{13}
ネイティブ (サイクル数)	2.38×10^{5}	3.31×10^{5}	1.11×10^{6}	9.61×10^{6}	1.58×10^{8}	2.08×10^{9}
サイクル比	**1.87×10^{4}**	**3.01×10^{4}**	**5.79×10^{4}**	**6.93×10^{4}**	**4.67×10^{4}**	**3.87×10^{4}**

基本性能評価：空のforループ

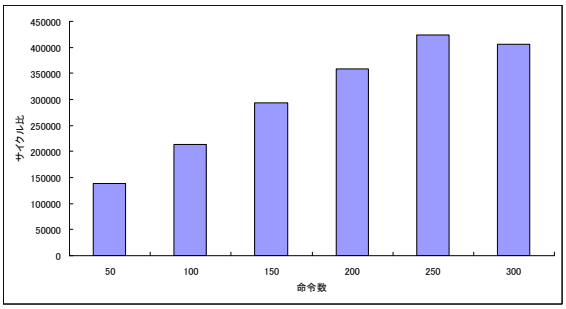


- 内部で何も行わない
  C言語のforループ
- 本エミュレータはネイティブに比べ
  10万～14万倍ほど低速

ループ回数	1	10	100	1000	1万	10万	100万	1000万	1億
本システム（サイクル数）	1.6×10^{8}	1.9×10^{8}	3.4×10^{8}	1.8×10^{9}	1.6×10^{10}	1.6×10^{11}	1.6×10^{12}	1.6×10^{13}	1.6×10^{14}
ネイティブ（サイクル数）	1.6×10^{3}	1.8×10^{3}	3.1×10^{3}	1.5×10^{4}	1.4×10^{5}	1.3×10^{6}	1.2×10^{7}	1.5×10^{8}	1.4×10^{9}
サイクル比	**9.9×10^{4}**	**1.0×10^{5}**	**1.1×10^{5}**	**1.1×10^{5}**	**1.2×10^{5}**	**1.1×10^{5}**	**1.3×10^{5}**	**1.0×10^{5}**	**1.1×10^{5}**

17

基本性能評価：分岐の無いコード

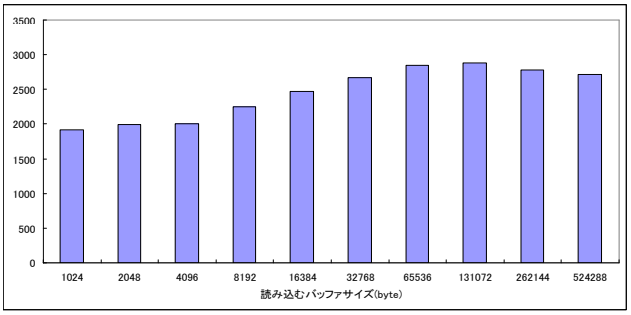


- ADD命令を分岐無しで
  直線的に実行するコード
- 本エミュレータはネイティブに比べ
  14万倍～42万倍程低速

命令数	50	100	150	200	250	300
本システム(サイクル数)	2.0×10^{8}	3.4×10^{8}	4.9×10^{8}	6.3×10^{8}	7.8×10^{8}	9.2×10^{8}
ネイティブ(サイクル数)	1.4×10^{3}	1.6×10^{3}	1.6×10^{3}	1.7×10^{3}	1.8×10^{3}	2.2×10^{3}
サイクル比	**1.3×10^{5}**	**2.1×10^{5}**	**2.9×10^{5}**	**3.5×10^{5}**	**4.2×10^{5}**	**4.0×10^{5}**

18

基本性能評価：I/O読み込み



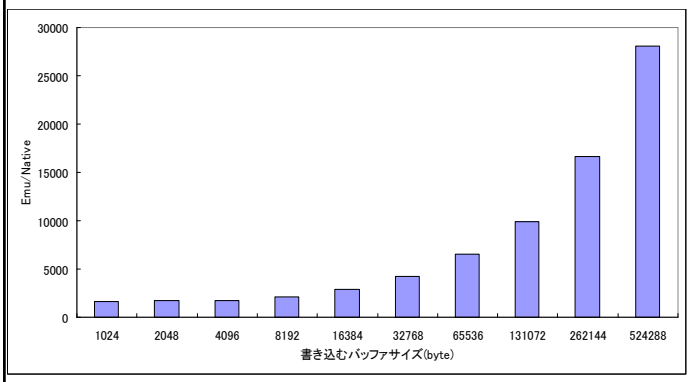
・/dev/zero からread()を1000回実行

・read()で読み込むサイズを変化させた

・約2000～3000倍程度低速

バッファサイズ (byte)	1024	2048	4096	8192	16384	32768	65536	131072	262144	524288
サイクル比	**1919**	**1998**	**2008**	**2250**	**2471**	**2671**	**2843**	**2876**	**2781**	**2710**

基本性能評価：I/O書き込み



・/dev/null にwrite()を1000回実行

・write()で書き込むサイズを変化させた

・1600～3万倍程度低速

バッファサイズ (byte)	1024	2048	4096	8192	16384	32768	65536	131072	262144	524288
サイクル比	**1618**	**1688**	**1726**	**2128**	**2928**	**4246**	**6576**	**9930**	**16663**	**28042**

基本性能に対する考察

- 本エミュレータの純粋な演算性能はネイティブ環境に比べ
  サイクル数ベースで2万～40万倍程度遅い

  - 命令デコード結果を文字列として処理する設計
    (特に命令デコード結果のキャッシュが効かない場合は低速)

- I/O性能は1600倍～2.8万倍程度で実現
  - システムコール性能が純粋な演算に比べ良いのは
    ユーザーモードエミュレータの特性

- 参考)QEMUのユーザーモードエミュレーション機能では
  - 純粋な演算性能はネイティブの3倍～50倍程度低速
  - I/O性能はネイティブの1.3～8倍程度低速

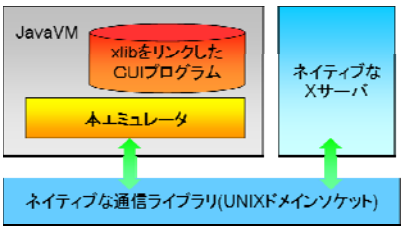
システムコールを多用するアプリケーションでの実用性

21

GUIアプリケーションの実行

- 別途ネイティブ環境のXサーバが描画を行う
  - 本エミュレータはXサーバとの通信部分までをエミュレーションする

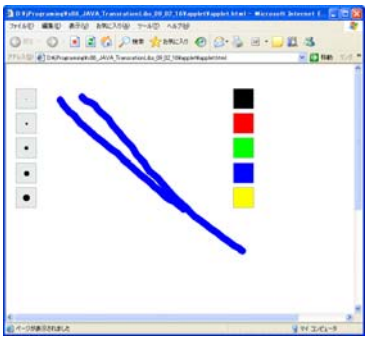
- 共有ライブラリのxlib.soを使用するプログラムの実行を確認



22

アプレット化の実現

- JavaベースのXサーバ(WeirdX)と本エミュレータを組み合わせた
- ファイルシステム等はクライアント側の資源にアクセス



Windows・IE上での
アプレット化を確認

x86/Linux用デスクトップアプリケーションをWebアプリ化できる

23

まとめ

- x86/Linuxアプリケーションを対象としたユーザーモードエミュレータをJava上で実装し、その評価を行った
- 大幅な性能低下が見られたものの、異種OS間での可搬性確認や以下の機能をサポートしている
  - GUI
  - マルチスレッド
  - 共有ライブラリのロード
- Linux用GUIプログラムのアプレット化を実現し,簡単なプログラムがWebブラウザ上で動作することを確認した.

24

今後の予定

- 完成度の向上
  ⇒ 命令エミュレーションのバグフィックス
- 性能向上
  ⇒ 既存研究(NestedVM)では2～8倍程度の性能低下で
  MIPSアプリのエミュレーションを実現
- エミュレータのPureJava化
  ⇒ Java標準ライブラリによる
  POSIXシステムコールのエミュレーション
- (将来的には)
  OpenOfficeをJavaアプレットして動作
  ⇒ Google Docs に対抗

25

システムソフトウェア教育支援環境「港」のシステムソフトウェア

早川栄一†　青山誠一†　西野洋介‡

1．はじめに

現在，教育界および産業界の両方で，組込みシステム教育の重要性が高まっている．中等教育においては，教科情報や技術家庭の科目の中にロボットを用いた演習が採り入れ始められている．また，高等教育においても，産業界での神座育成の必要性の高まりから，専門課程で組込みシステムコースの設置が行われはじめている．機器の高度化，ソフトウェア化が高まるにつれてソフトウェアエンジニアの不足は深刻であり，システム教育の役割も増大している．

その一方で，システムソフトウェアの学習には課題が多い．ソフトウェアそのものの問題として仮想化による内部動作の隠蔽や，多くの関連知識獲得の難しさ，また，教育コストの問題として，教材作成の手間や，学習者の学力の分散が広がっている点，授業に割り当てられる時間の限界などが問題として挙げられる．

本報告では，これらの問題を解決するシステムプラットフォームとして，統合型のシステムソフトウェア学習支援環境である「港」システムの開発，特に「港」を支えるシステムソフトウェアについて述べる．システムソフトウェアの動作原理の理解，および演習で用いることができる環境の構築，さらに学習者の利用や操作学習，教授者の管理が容易な環境の構築を目指す．

2．「港」システムの構成

「港」システムを図１に示す．港システムは次の四つの構成要素によって，従来の学習環境が抱える問題を解決する．

- ロボット

ロボットによって学習者の興味を持続させる．学習環境に配慮して，多様なロボットハードウェアおよびその上で動作する OS を対象とする．

- システム可視化環境

ロボットの動作動画と入出力データ，OS の内部状態とを連動して可視化することによって，学習者がシステムの挙動を理解しやすくする．

- 仮想環境

3D シミュレータおよび CPU シミュレータによるマシン上での動作検証を可能にした．ロボット台数や物理的な問題を回避し，プログラム単体での検証を容易にする．また，ロボットを仮想マシン環境上で動作させて，ハードウェアのモニタリングを容易にする．

- 統合システム

システム全体を統合環境として提供することで，学習者の操作学習の負担を軽減する．また，ベースに Linux を用いた 1CD/1USB によってシステム全体を構築することで，教授者の教材管理負荷を軽減する．学習教材テンプレートをサポートし，教材作成についても容易にしている．

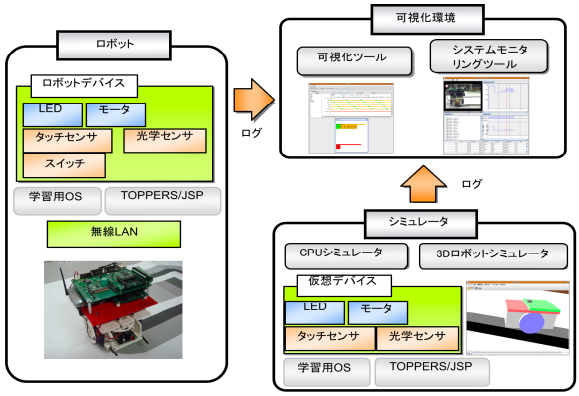


図１　港システムの全体構成

3．仮想マシンマネージャ

「港」システムのロボットは，仮想マシンマネージャ（VMM）上で，学習 OS を動作させる．これは，複雑なハードウェアに対応させることと，通信機構やストレージ，ファイルシステムなどを，学習 OS に組み込まずに利用できるようにする点，また，ロボットの暴走によるログの消失や制御不能に対する問題を解決するためである．

3．１　仮想マシンマネージャの特徴

本 VMM の特徴は，次の３点である．

- VMM によるユーザモードでの教材 OS の動作による安全な環境の提供
- デバイス管理 OS を特権モードで動作させることによるエミュレーションコストの低減
- VMM の GUI を提供することによる操作環境の改善

VMM を含むロボット上のソフトウェアの全体構成を図２に示す．

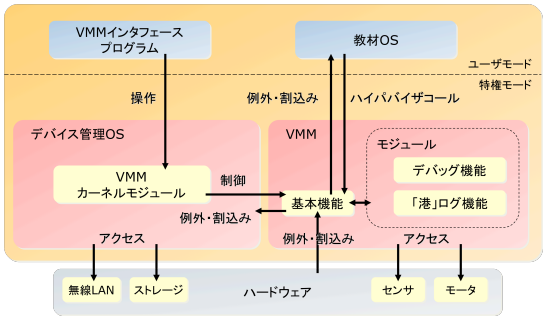


図２　仮想マシンの全体構成

本システムは，デバイス管理用 OS と教材 OS およびそれら

†拓殖大学　工学部　情報工学科
‡東京都立は当時桑志高等学校

システムソフトウェア教育支援環境「港」のシステムソフトウェア

拓殖大学　工学部　情報工学科
早川 栄一　青山 誠一
都立八王子桑志高等学校
西野 洋介

システムソフトウェア教育の重要性

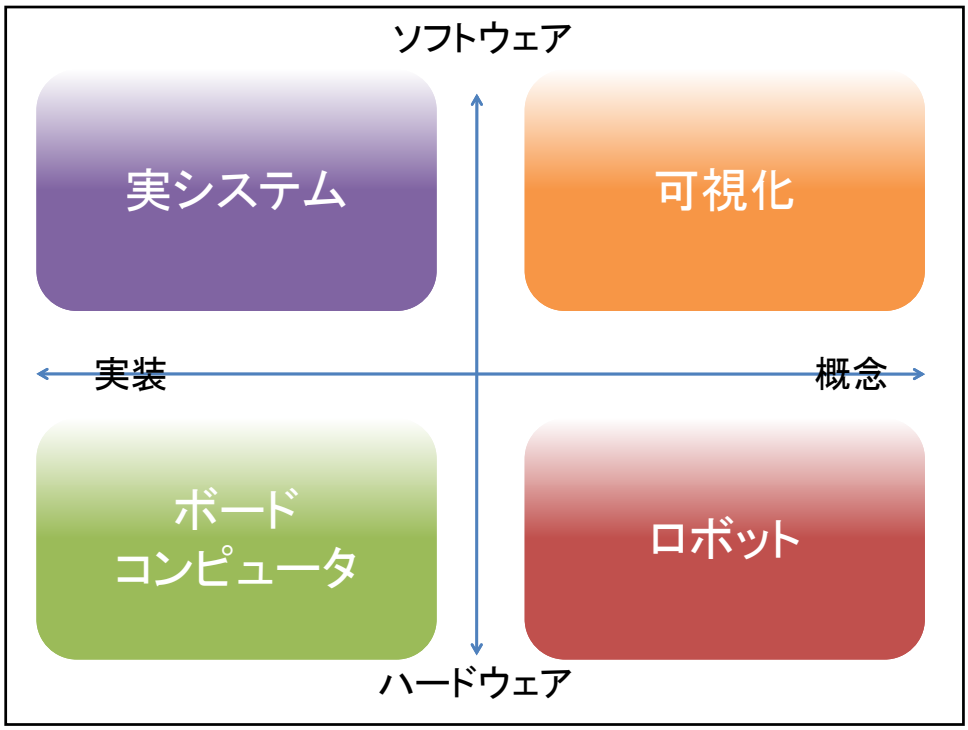
- 教育界の現状
  - 「ものづくり」教育
  - 教科「情報」，技術家庭
  - ロボットを利用した演習
  - J90,J97,J07でも重要な科目
- 産業界の現状
  - 組込み機器の高度化，ソフトウェア化
  - ソフトウェアエンジニアの不足
    - 42万人の不足（経団連）
  - システムソフトウェアの役割の増大

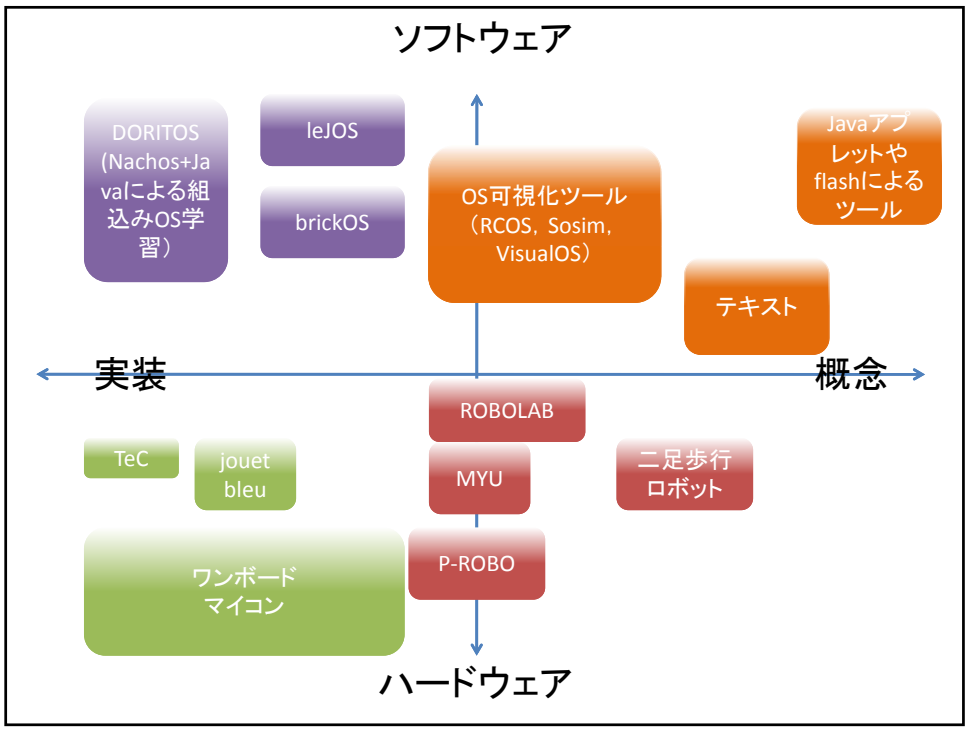
問題分析

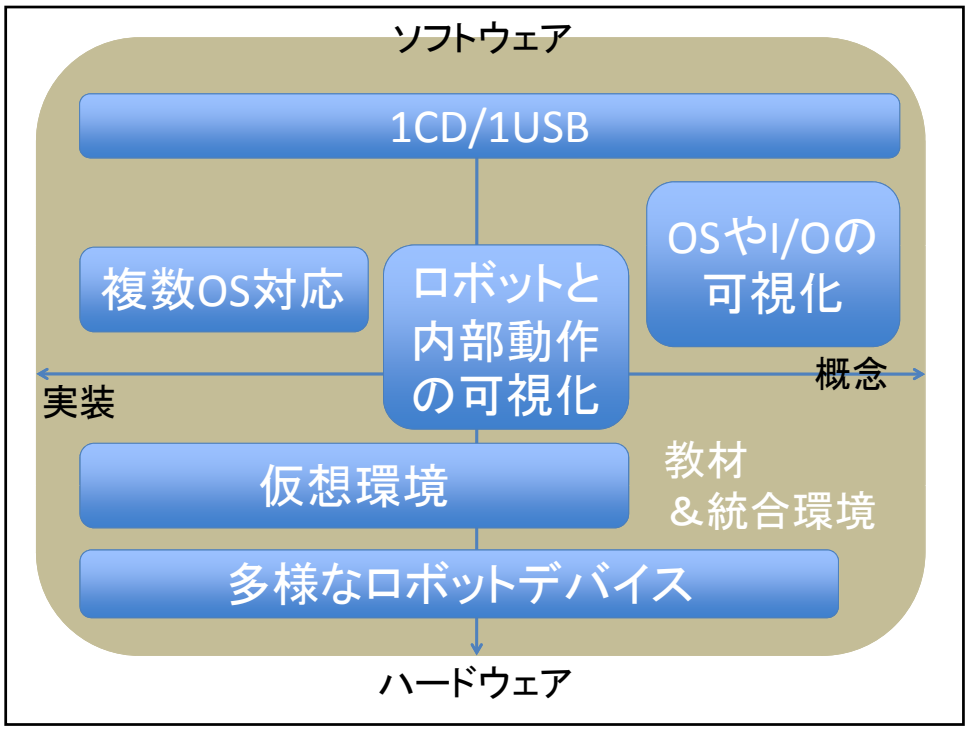
- システムソフトウェアに起因する原因
  - 仮想化によるブラックボックス化
  - 多くの関連知識取得のコスト
- 教育コストに起因する原因
  - 教材作成の手間
  - 学習者の学力の分散
  - 限られた時間

目的

- システムソフトウェアに対する要求
  - 動作原理の理解
  - 演習を中心とした利用スキルの習得
  - 利用や管理，操作学習が容易であること
- 目的
  - 統合型のシステムソフトウェア学習支援環境の開発
  - 「港」システム

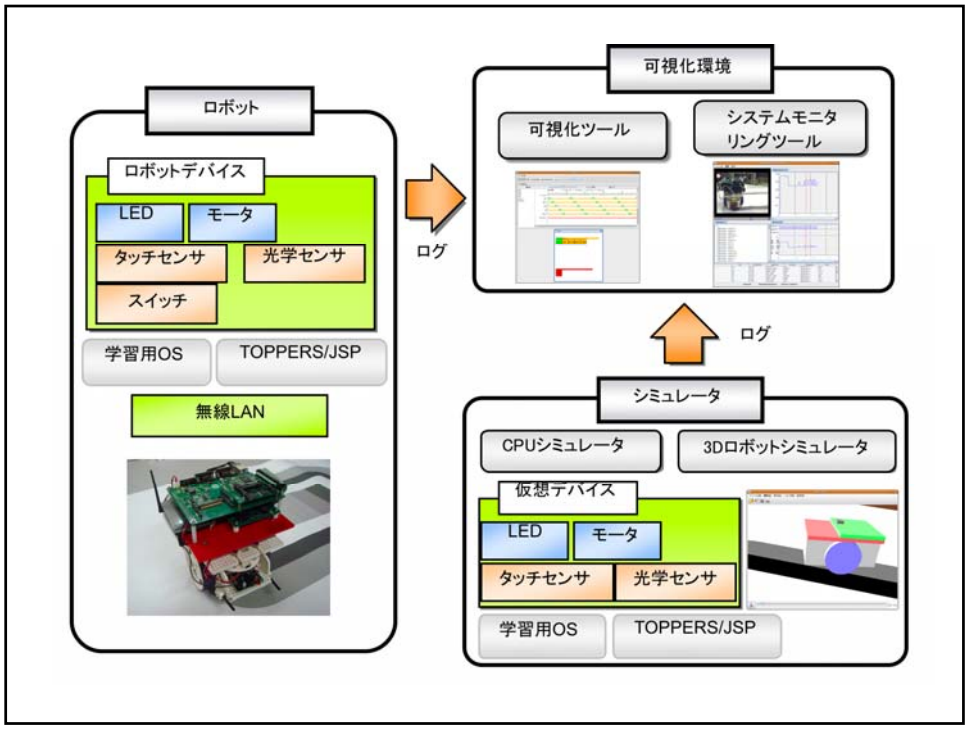

ソフトウェア
実システム
可視化
実装
概念
ボード
コンピュータ
ロボット
ハードウェア


ソフトウェア
DORITOS
(Nachos+Ja
vaによる組
込みOS学
習)
leJOS
brickOS
OS可視化ツール
(RCOS, Sosim,
VisualOS)
Javaアプ
レットや
flashによる
ツール
テキスト
実装
概念
ROBOLAB
TeC
jouet
bleu
MYU
二足歩行
ロボット
ワンボード
マイコン
P-ROBO
ハードウェア



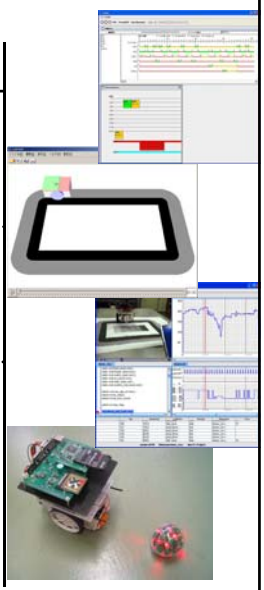
「港」システム

- ロボット
  - 多様なロボットのサポート
- 可視化
  - ロボットの動作と入出力データとを連動して可視化
  - リアルタイムOSや入出力の動作を可視化する環境
- 仮想環境
  - 3Dシミュレータ，CPUシミュレータによる動作検証を簡易化
  - 仮想マシンによるハードウェアのモニタリング
- 統合システム
  - エディタやロボット操作を含む統合環境による操作学習負荷の低減
  - 1CD/1USBブート環境による実行や管理の容易化
  - 学習教材テンプレートのサポート



コースウェア

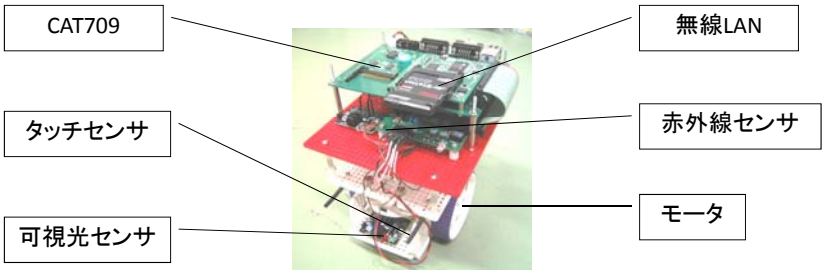
	学習者	学習支援ツール
概念	直観的な概念学習	可視化
	OS，制御の実践学習	シミュレータ＋可視化
実装	ハードウェアによる実践学習	ロボット＋シミュレータ＋可視化

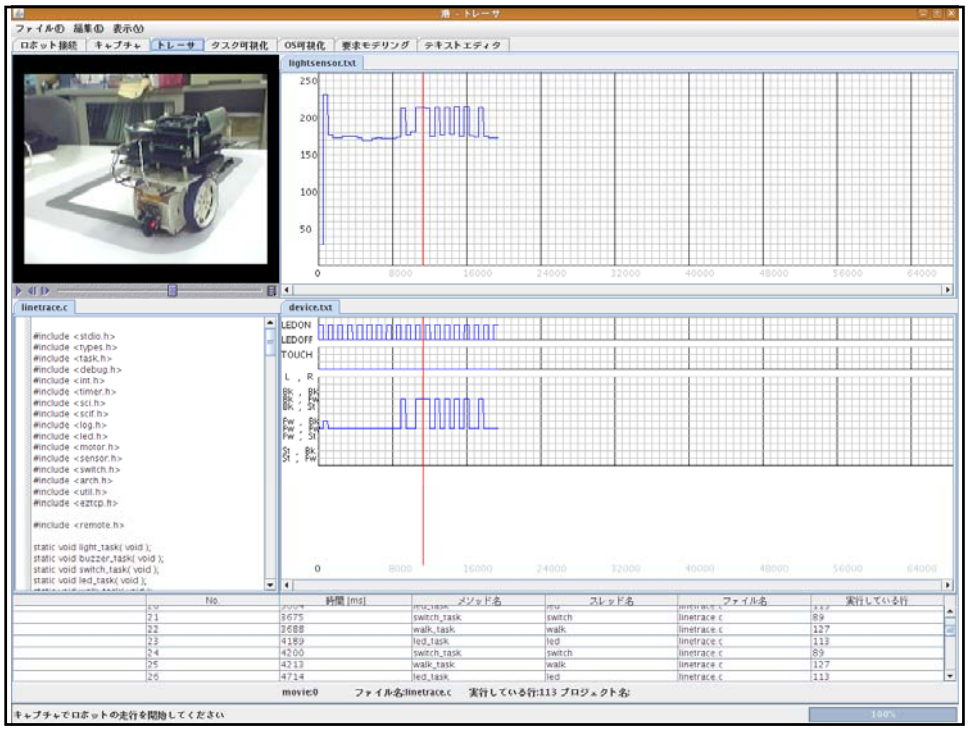


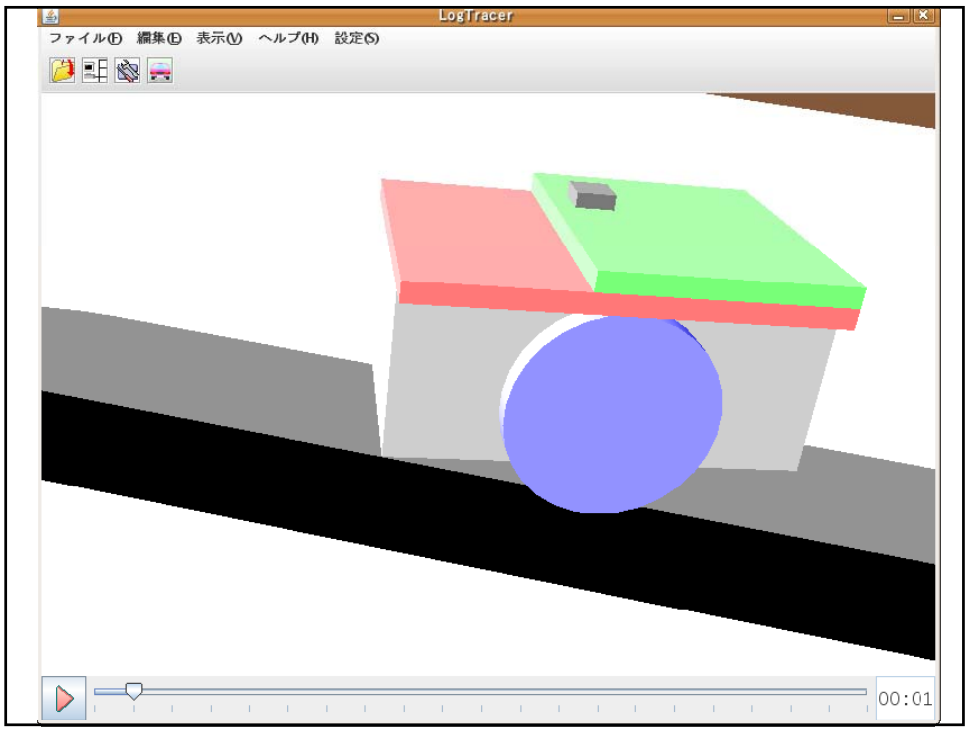
単一の環境で概念から実装までを段階的に学習

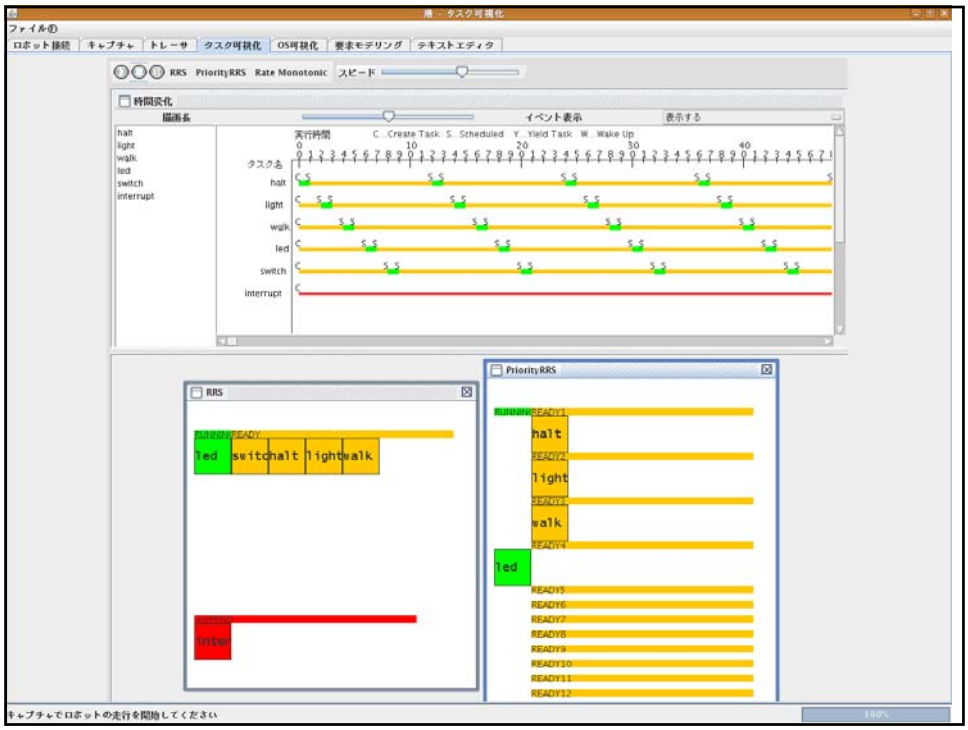
対象ハードウェア

- 教育用ロボット
  - 組込みシステム学習環境「港」で利用
  - SH7709S（SH-3）CPU を搭載の CAT709 ボード
  - 豊富なデバイスと拡張性




ファイル(F) 編集(E) 表示(V)
ロボット接続
キャプチャ
トレーサ
タスク可視化
OS可視化
要求モデリング
テキストエディタ
lightsensor.txt
linetrace.c
device.txt
LEDON
LEDOFF
TOUCH
No.
時間 [ms]
メソッド名
スレッド名
ファイル名
実行している行
キャプチャでロボットの走行を開始してください


LogTracer
ファイル(F) 編集(E) 表示(V) ヘルプ(H) 設定(S)
00:01



港の仮想マシンマネージャ

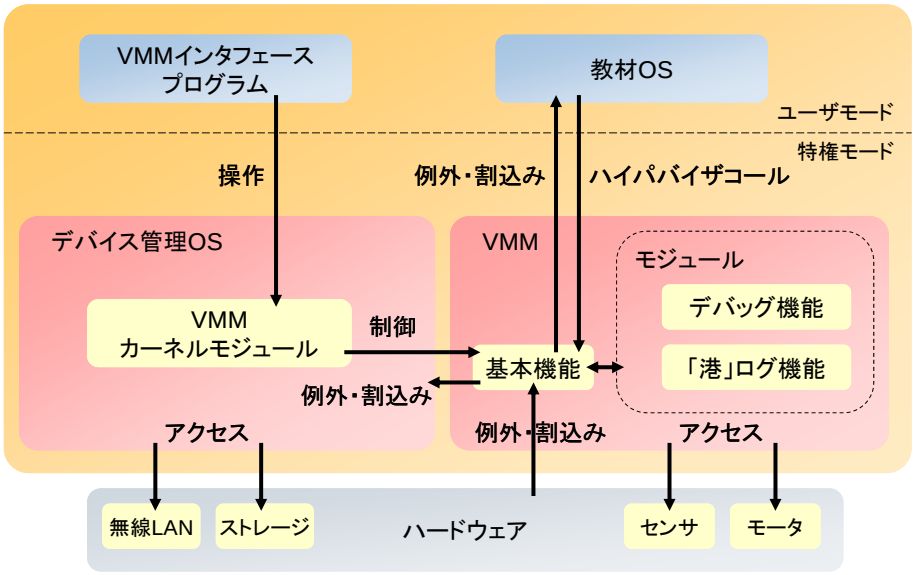
- ハードウェアの複雑化への対応
  - 学習者が理解しやすい単純な設
    - 教材OSを単純にできる
  - ハードウェアの複雑化に対応したい
    - 通信機構やストレージ，ファイルシステム
- ロボットデバイスの安定稼働
  - 暴走時でもログを残したい
    - ストレージや通信帯域の問題
  - 安定した制御機構

組込みシステム指向仮想マシンマネージャ（VMM）の開発

VMMの特徴

- VMMによるユーザモードでの教材OSの動作
  - 複雑なデバイスの管理を担う部分を別のOSとして分離
  - 暴走時のログの取得を実現
- デバイス管理部分を特権モードで動作
  - エミュレーションオーバヘッドの低下
  - 準仮想化によるハイパーバイザインタフェース
  - タイマ管理オーバヘッドの低下
- 学習環境の改良
  - ユーザインタフェースの改善による操作性の向上
  - 学習者がGUIからVMMを透過的に利用

VMM全体構成

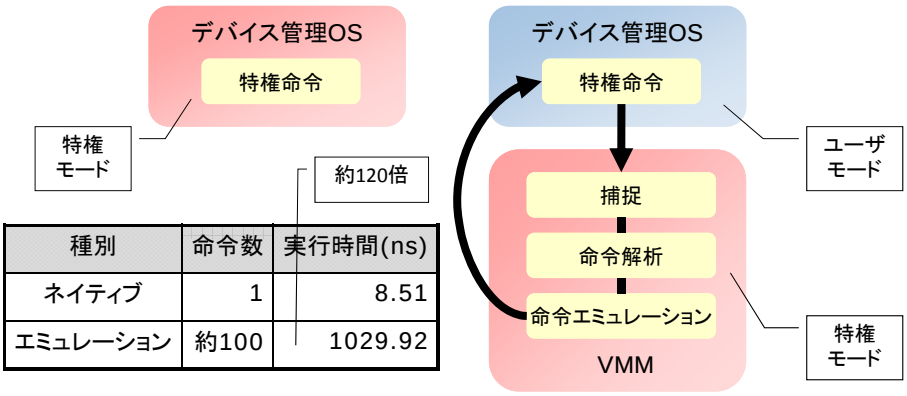


仮想マシンマネージャ

- 提供する機能
  - 特権命令の捕捉・エミュレーション
    - ユーザモードでのOS動作を可能にする
  - ハイパバイザコール
    - 教材OSからVMMに対するシステムコール
    - ポート通信や仮想デバイスI/Oを実現
  - メモリ保護
    - メモリ管理ユニット（MMU）によって各OSのメモリ領域を保護
  - デバッグ機能
    - ユーザモードのOSデバッグを支援
  - ログ機能
    - デバイス情報などのログを保存

オーバヘッドの削減（1）

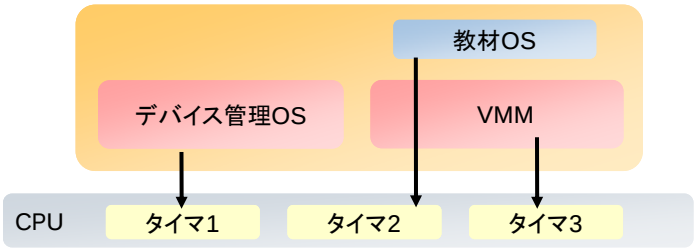
- デバイス管理OSを特権モードで動作
  - デバイス管理におけるオーバヘッドを減らす



種別	命令数	実行時間(ns)
ネイティブ	1	8.51
エミュレーション	約100	1029.92

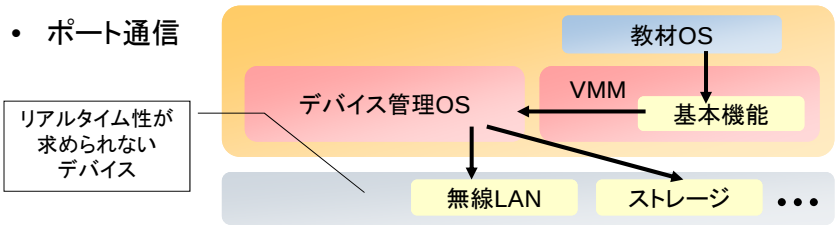
オーバヘッドの削減(2)

- 独立した3つのタイマを利用
  - 短い間隔で呼び出されるため実行コストが問題
    - 平均4μ秒のコストが掛かる
  - SH7709Sは3つのタイマを持つ
  - タイマデバイスのエミュレーションが不要

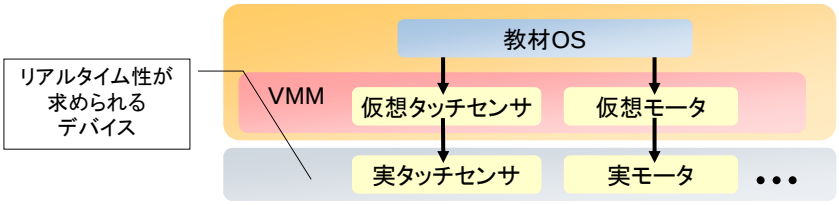


オーバヘッドの削減(3)

- ポート通信



- 仮想デバイスI/O

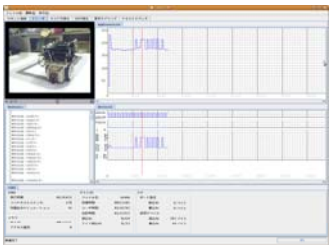


デバッグ機能

- 教材OSのデバッグに必要な機能を提供
  - 指定したメモリアドレスの監視
  - レジスタの監視
  - ブレークポイントなどでの一時停止・再開
- 学習環境との統合
  - 「港」のGUIから教材OSの実行などを操作
  - 監視している各情報の表示

ログ機能

- 教材OSだけでは取得の難しい情報
  - 各デバイスの状態
  - 割込み禁止時のログ
  - プログラム暴走時のログ



- 学習環境との統合
  - 各種ログを「港」ツールで表示
  - 時系列でそれぞれの情報を比較

デバイス管理OS

- デバイス管理用の独立したOS
  - 教材OSを複雑化せずに様々なデバイスを利用
- Linuxを採用
  - 豊富なデバイスドライバをそのまま利用できる
- VMMとの協調動作
  - 同じ特権モードで動作

教材OS

- VMM上で動作するOS
  - 学習用OS
    - 2000行程度の小型のRTOS
  - TOPPERS/JSP
- VMM上で動作させるための修正
  - 利用メモリアドレス空間
  - I/Oアクセス
  - 例外処理

独自OSの場合

項目	行数（行）
アドレス空間	4
I/O	60
例外処理	6

OS教材の修正

- 利用メモリアドレス空間
  - ユーザモードからアクセス可能な範囲に変更
- I/Oアクセス
  - inbやoutbを用いてアクセスしている部分
  - vd_inbやvd_outbに置き換えて仮想デバイスI/Oを利用
- 例外処理
  - 要因を特定のアドレスを読み出すことにより取得
  - レジスタで渡された要因を利用

例：OSビルド用スクリプト

```
INCDIR = $(TOPDIR)/sys/inc
LDSCRIPT = $(TOPDIR)/build/ldx/CAT709.x
STARTADDR = 0x8c300200
OBJCOPYFLAGS = -O srec --srec-forceS3 --
set-start $(STARTADDR)
```

ユーザモードからアクセス可能な
0x00000000～0x7FFFFFFF
の範囲に変更する

OS教材の修正

- 利用メモリアドレス空間
  - ユーザモードからアクセス可能な範囲に変更
- I/Oアクセス
  - inbやoutbを用いてアクセスしている部分
  - vd_inbやvd_outbに置き換えて仮想デバイスI/Oを利用
- 例外処理
  - 要因を特定のアドレスを読み出すことにより取得
  - レジスタで渡された要因を利用

例：inb，outbの置き換え

```
// 入力命令
#define    inb   vd_inb
#define    inw   vd_inw
#define    inl   vd_inl

// 出力命令
#define    outb vd_outb
#define    outw vd_outw
#define    outl  vd_outl
```

マクロで置き換え

OS教材の修正

- 利用メモリアドレス空間
  - ユーザモードからアクセス可能な範囲に変更
- I/Oアクセス
  - inbやoutbを用いてアクセスしている部分
  - vd_inbやvd_outbに置き換えて仮想デバイスI/Oを利用
- 例外処理
  - 要因を特定のアドレスを読み出すことにより取得
  - レジスタで渡された要因を利用

例：例外処理

```
exception_handler:
   ! 例外要因レジスタのアドレスを
   ! r2 レジスタへ代入
   ! r4レジスタで渡された例外要因を
   ! r2 レジスタへ代入
   mov.l        r4, r2
   mov.l    @r2,   r2
   ! all_exception に無条件分岐
   bra    all_exception
    nop
   .align     2
1: .long      EXPEVT
```

評価

- VMMの性能評価
  - 仮想デバイスI/Oに掛かる時間の測定
- 対象デバイス
  - LED（CPUの外部I/Oポートに接続）
- 方法
  - LEDに対して1億回入力命令を呼び出し，1命令当たりの実行時間を求める

評価結果

種別	時間（マイクロ秒）	ネイティブとの差
ネイティブ	374.55	-
仮想デバイスI/O	374.68	0.13
エミュレーション	376.17	1.62

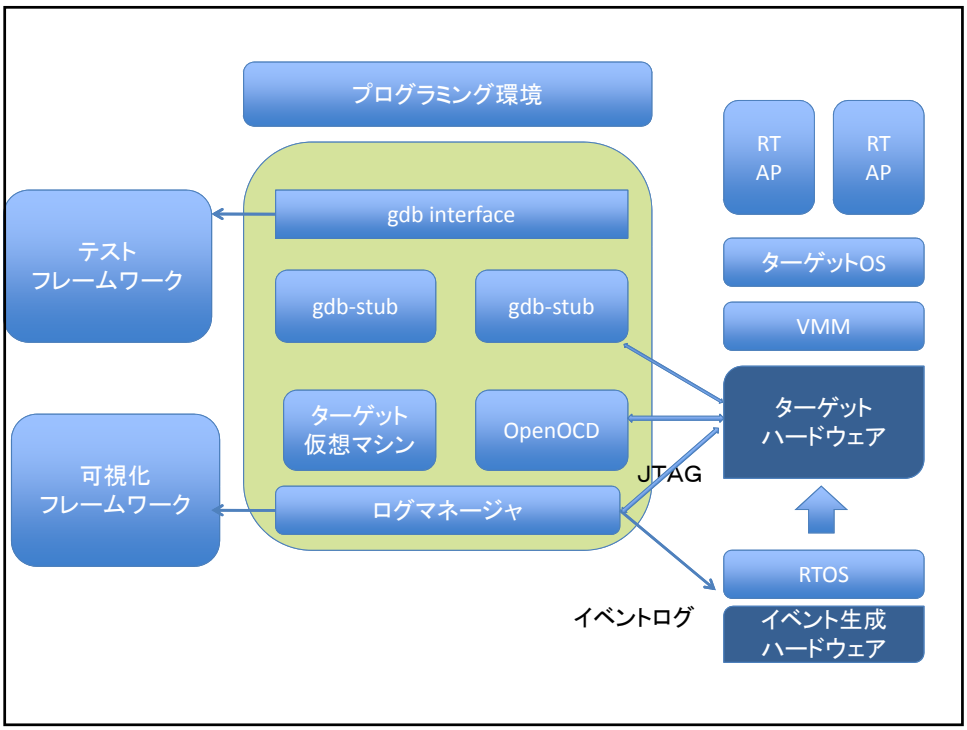
- ネイティブ
  - 直接アクセス
- 仮想デバイスI/O
  - ハイパバイザコールによるVMMを用いたアクセス
- エミュレーション
  - I/O操作のエミュレーションを行うアクセス

次の「港」へ向けて

- VMと実機インタフェースとの統合
- OSの多様化に対応した可視化フレームワーク
  - OSの多様化に対応
  - ユーザが自由に改良できる
- ポータブルなロギングインタフェース
  - データ自身に情報を持たせる
  - 省サイズ
- ポータブルなビデオインタフェース
  - Java Media Frameworkに依存しない

VMと実機とのインタフェース統合

- 組込みシステムでのシミュレータ利用は一般的になりつつある
  - 実機を用意しなくてもできる
  - QEMUでも各種CPUに対応
- 実機はまったく異なるインタフェース
  - 実機とはJTAGやシリアル通信
  - とても面倒で使いにくい
- デバッグ/テストのインタフェース
  - 実機での組込みシステムでのテスト
  - イベントを外部から与える仕組みの導入



おわりに

- システムソフトウェア学習支援環境「港」の紹介
- 「港」が提供する仮想マシンマネージャ
- 1CD/1USB環境で現在利用中

- 課題
  - 次の「港」へのシステム再構築
  - ユーザインタフェースの改良
  - ドキュメント
  - 継続した利用評価

Linux Security Module を用いた Privacy-aware OS Salvia の構築

鍛治 輝行 †
† 立命館大学大学院理工学研究科

毛利 公一 ††
†† 立命館大学情報理工学部

1 はじめに

近年，プライバシ情報が電子化され，その情報が計算機で管理されている．このプライバシ情報が，記憶媒体やネットワークを通じて漏洩する事件が多発している．個人情報の保護は，2005 年 4 月 1 日から施行された個人情報保護法により，保護が義務付けられている．暗号化，認証，侵入検知といった技術により，セキュリティ侵害における情報漏洩を防ぐ事が可能である．しかし，日本ネットワークセキュリティ協会の情報セキュリティインシデントに関する調査報告書 [1] における，情報漏洩比率によると，セキュリティ侵害といった外部からの攻撃による漏洩の比率は低い．漏洩原因の多くを占めるものが，ユーザの管理ミス，紛失，誤操作といった正当なアクセス権限を持つものによる漏洩である．暗号化，認証，侵入検知といった技術では，このような正当なアクセス権限を持つ者が引き起こす漏洩を防ぐことが困難である．以上の背景により，我々は，正当なアクセス権限を持つ者による情報漏洩を防止するために，Privacy-aware OS *Salvia*[2] の開発を行っている．*Salvia* を用いることにより，正当な権限を持つユーザによる情報の漏洩を防止する事が可能となる．

これまで，*Salvia* は Linux カーネルを改変する形で開発してきたが，より移植性を高くし，ユーザによる *Salvia* の導入を容易にすることを目的として，Linux Security Module[3](以下，LSM と略す) に基づいて開発を行っている．以下，本稿では，2 章で *Salvia* の概要について述べ，3 章で LSM の概要について述べる．4 章で LSM に対応した *Salvia* の構成について述べ，5 章で本稿のまとめとする．

2 *Salvia* の概要

Salvia は，アクセス制御を行い，情報の漏洩を防止する OS である．図 1 に，*Salvia* のデータ保護モデルを示す．*Salvia* は，ファイルを保護の単位としており，保護対象ファイルをオープンしたプロセスを制御の対象としている．データ作成者は，保護対象ファイルごとに保護ポリシを作成する．保護ポリシには，対応する保護対象ファイルをオープンしたプロセスに課す制約と，その制約を課す条件を記述する．*Salvia* では，プロセスを制御するための条件として，コンテキストの記述を可能としている．プロセスを制御する際に用いるコンテキストには，端末の位置，現在の時刻，ユーザ ID，過去の動作履歴などがある．*Salvia* は，この保護ポリシに基づき，制御対象プロセスに対し，ファイル，ソケット，パイプなどの計算機資源へのアクセスを制御することにより，データの漏洩を防止する．

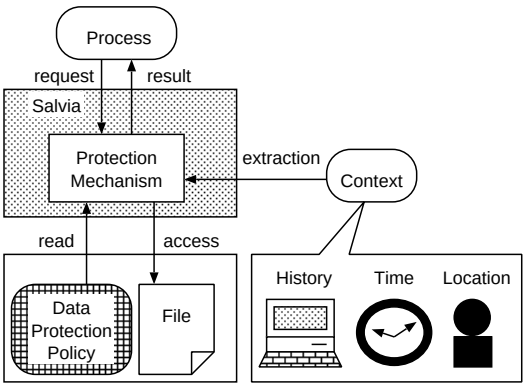


図 1 *Salvia* のデータ保護モデル

3 LSM の概要

LSM は，カーネルにセキュリティ機能を拡張するためのフレームワークであり，Linux カーネル 2.6 シリーズ以降，正式に採用されている．LSM が採用されているカーネルには，ソースコード中のファイルシステム操作の可否などのセキュリティ・チェックポイントで，コールバック関数呼び出しが挿入されている．対応するコールバック関数を用意することで，ファイルシステム操作の可否の判断に，より詳細なチェックを加える，といったことが可能となる．

LSM に基づき作成したセキュリティモジュールは，LSM が採用されている Linux2.6 シリーズ以降のカーネルであれば，移植が可能となる．現在の *Salvia* は，Linux カーネル 2.6.8 を基に実装を行っており，ソースコードに変更を加えることにより開発を行っている．LSM に基づき *Salvia* を構築することで，LSM が採用されているカーネルへの移植が可能となる．

4 LSM に対応した *Salvia* の構築

4.1 現在の *Salvia* の処理の流れ

現在の Salvia において，プロセスのアクセスを制御する処理の流れを，以下に述べる．

- プロセスがファイルをオープンする際に発行する open システムコールのフックを行う．
- open 対象ファイルに対応する保護ポリシを取得し，プロセスの動作の監視を開始する．

- プロセスが計算機資源に対し，アクセスを行う際に発行するシステムコールをフックし，保護ポリシに基づき計算機資源へのアクセスを制御する．

以上の処理を行うことにより，保護ポリシに基づき，ファイルの保護が可能となる．*Salvia* では，アクセスの種類を，以下に述べる 4 つに分類を行い，この分類に基づきアクセスの制御を行っている．

- read グループ ファイルからのデータの読み出し
- write グループ ファイルへのデータの書き込み
- send_local グループ 同一計算機上の他のプロセスのデータ領域への書き込み
- send_remote グループ 他の計算機上のプロセスのデータ領域への書き込み

4.2 LSM を用いた *Salvia* の構築

LSM を用い *Salvia* を構築し，移植性を向上させるためには，カーネルソースに変更を加えずに開発する必要がある．しかし，現在の *Salvia* では，ソースコードに変更を加え，システムコールテーブルを書き換えることにより，システムコールのフックを行っている．そのため，LSM のフック関数を用い，カーネルに変更を加えることなく，前節で述べたような *Salvia* のアクセス制御を実現する必要がある．

LSM フックは，1 つのシステムコールで複数呼ばれる可能性があり，処理によって呼ばれない場合もある．また，複数のシステムコールが呼び出す LSM フックも存在する．そのため，各処理関数に対応する適切な LSM フックを選ぶ必要がある．

以下に，LSM を用いた *Salvia* において，フックが必要な処理と，その際に用いるフック関数について述べる．

はじめに，ファイルのオープン時，その処理をフックする．この際，security_inode_permission 関数を用いる．ファイルをオープンする際は，そのファイルに対応する i-node へのアクセスが行われる．security_inode_permission 関数は，その i-node へのアクセスの可否を判別する箇所に挿入されている．この関数は，ファイルがオープンされる際，必ず一度呼び出される．この時点で，オープン対象ファイルが，保護対象ファイルであるか調べる．保護対象であるか否かは，オープン対象ファイルに対応する保護ポリシの有無で判別を行う．現在，この保護ポリシは，保護対象ファイルの i-node 拡張属性に格納している．i-node へのアクセスの可否を判別するとともに，i-node 拡張属性に格納されている保護ポリシを取得することが，security_inode_permission 関数の時点で可能である．保護ポリシが存在した場合，その情報を取得し，プロセスへの制御を開始する．

次に，監視対象プロセスが計算機資源に対するアクセスをフックを行う．この際には，アクセスの種類ごとに異なるフック関数を用いる必要がある．アクセスの種類は，上記した *Salvia* における分類に従う．現在，read グループ，write グループのフックの実装が完了している．read グループ，write グループのフックには，同一のフック関数 file_permission 関数を用いる．read グループ，write グループは，共にファイルへのアクセスであるため，そのファイルへのアクセスの前に，パーミッションチェックを行う箇所のフック関数を用いることにより，アクセスのフックが可能となる．read グループと write グループは，file_permission 関数の引数である mask により区別が可能である．アクセスのフック後，ファイルのオープン時に取得した保護ポリシに基づき，アクセスの可否を決定する．アクセス可否の判定を行う際，時刻，ユーザ ID，グループ ID をコンテキストとして用いる実装が完了している．位置や RFID といったコンテキストについては，未実装である．send_local グループ，send_remote グループのフックに関しては，現在調査中であり，今後の課題となる．現在は，send_local グループの中でも，特にカーネルが読み書きを検知できない共有メモリについて実装を行っている．

以上の処理を行うことにより，*Salvia* のアクセス制御と同様の処理が実現可能となる．

5 おわりに

本論文では，*Salvia* におけるアクセス制御を，LSM に対応する形で構築を行う手法について述べた．LSM のフック関数を利用し，処理のフックを行う箇所は，保護対象ファイルのオープンを行う箇所と，監視対象プロセスが計算機資源に対しアクセスを行う箇所である．プロセスが計算機資源へのアクセスをする際，適宜，LSM のフック関数を用いる事により，アクセスのフックが可能となる事を述べた．本手法により，移植性が向上し，導入が容易となる．今後の課題として，send_local グループ，send_remote グループのフックをし，アクセスの制御を行う必要がある．

参考文献

[1] NPO 日本ネットワークセキュリティ協会: JNSA 2007 年情報セキュリティインシデントに関する調査報告書，2008.

[2] 鈴木 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, Vol. 47, No. SIG3, pp. 1-15, 2006.

[3] Chris Wright, Crispin Cowan, Stephen Smalley, James Morris, Greg Kroah-Hartman : Linux Security Modules:General Security Support for the Linux Kernel, In Proc. of the 11th USENIX Security Symposium, pp. 17-31, 2002.

Mouri
Lab

Linux Security Moduleを用いた Privacy-aware OS Salviaの構築

立命館大学大学院
毛利研究室
鍛治　輝行

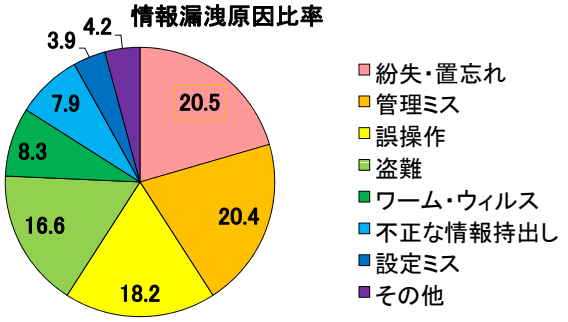
はじめに

- 研究背景
- Privacy-aware OS *Salvia* の概要
- Linux Security Module の概要
- LSMを用いたアクセス制御
- 実装済みの機能
- 実装中の機能
- 今後の課題

Mouri Lab

研究背景(1/2)

- **電子化されたプライバシ情報が漏洩する事件の多発**
  - 情報通信技術の発達や記憶媒体の大容量化により漏洩の規模・頻度の増大
- **情報漏洩原因の多くが、正当な権限を持つ者による漏洩**



日本ネットワークセキュリティ協会　情報セキュリティインシデントに関する調査報告書

2

Mouri Lab

研究背景(2/2)

- **既存の暗号化，認証，侵入検知などの技術では正当な権限を持つ者による情報漏洩を防ぐ事が困難**

- **正当な権限を持つ者による情報漏洩を防ぐ為，Privacy-aware OS *Salvia* の開発**

3

Mouri Lab

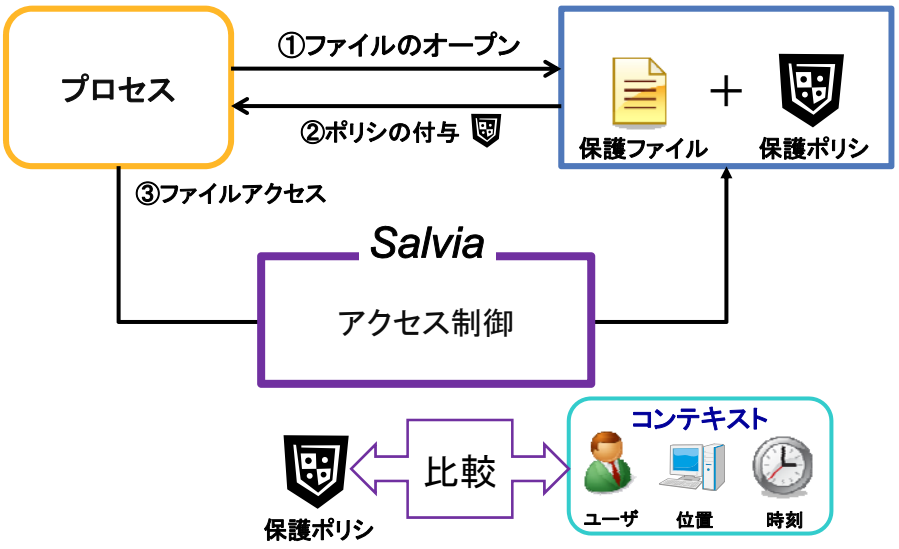
Privacy-aware OS *Salvia*

- **アクセス制御機構を備えたOS**
  - アプリケーションの信頼度に関わらず統一的に制御可能
- **保護方式**
  - 保護単位：ファイル
    - 保護したいファイルと保護ポリシを一組にして管理
  - 制御対象：プロセス
    - プロセスがデータを漏洩させる動作を制御
    - プロセスが実行するシステムコールをチェックすることで制御
  - コンテキストの利用
    - ユーザや計算機の状況のこと
    - ユーザ，時間，計算機の場所，IPアドレス　など

4

Mouri Lab

*Salvia*におけるアクセス制御



5

Mouri Lab

*Salvia*の課題

- Linuxカーネルを基に，カーネルソースに変更を加え開発
- *Salvia*導入時の制約
  - カーネルの再構築が必要
  - カーネルが開発を行っているバージョンに限定される

- 制約を解消し，
  ユーザによる*Salvia*の導入を容易にするため，
  *Salvia*をLinux Security Moduleに対応させる

6

Mouri Lab

Linux Security Module(LSM)

- カーネルにセキュリティ機能を拡張するための
  フレームワーク
- Linuxカーネル2.6以降，正式に採用
- カーネルの再構築無しにセキュリティ機能を拡張可能
- カーネルコードには，セキュリティモジュールへの
  コールバック関数呼出しが挿入されている

- I/Oポートへのアクセスの可否
- プロセス生成／終了の可否
- 各種ファイルシステム操作の可否
- 各種ソケット操作の可否

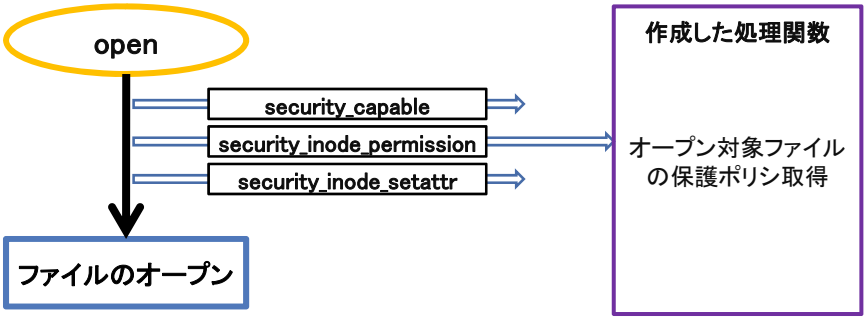
7

Mouri Lab

LSMを用いたモジュール作成法

- 適切なLSMフックに対応する制御関数を作成し，登録
- LSMフックの引数から得られる情報を基に，制御

- 例）openシステムコール

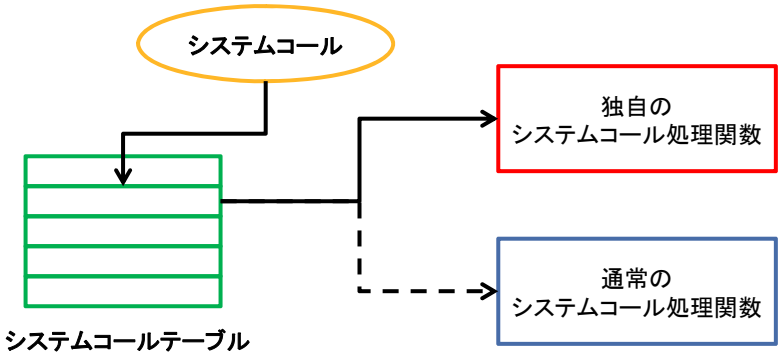


8

Mouri Lab

*Salvia*でのシステムコールフック

- システムコールテーブルを変更し，独自の処理関数へ
- カーネルに変更を加えることにより実現

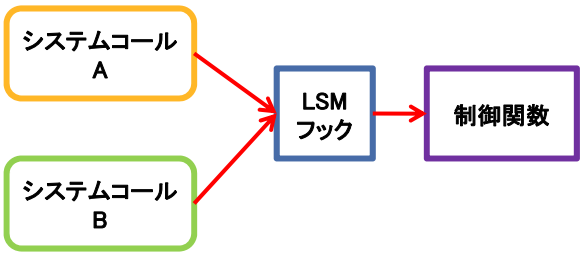


9

Mouri Lab

システムコール フック法

- *Salvia*は，プロセスが発行するシステムコールをフックし，チェックすることで制御を行う
- システムコール処理関数が呼び出すLSMフックを用いる
  - 複数のシステムコール処理関数が呼び出すLSMフックが存在
  - LSMフックの引数を基に制御→制御する為に十分な情報が必要



10

Mouri Lab

openシステムコールのフック

- 処理内容
  - オープン対象ファイルが保護ファイルであるかチェックする
  - 保護ファイルであれば，保護ポリシを取得　プロセスの制御開始
- LSMフックを選ぶ基準
  - openシステムコールが呼び出したと分かる
  - openシステムコールが呼ばれた際，必ず呼び出される
  - オープン対象ファイルを特定できる情報が得れる

11

Mouri Lab

open処理関数が呼び出すLSMフック

- **int security_inode_permission**
  - struct inode *inode, struct dentry *dentry, struct nameidata *nd
  - i ノードへのアクセスの前にパーミッションのチェックを行う
- **int security_inode_setattr**
  - struct dentry *dentry, struct iattr *attr
  - ファイルの属性をセットする前にパーミッションのチェックを行う
- **int security_inode_create**
  - struct inode *dir, struct dentry *dentry, int mode
  - 通常のファイルを生成する為に，パーミッションのチェックを行う
- **int security_capable**
  - struct task struct *tsk, int cap
  - プロセスtsk が，ケーパビリティcap を持っているかをチェックする.

12

Mouri Lab

read/writeシステムコールのフック

- **処理内容**
  - read/writeを行うプロセスが制御対象かチェック
  - 制御対象であれば，ポリシとコンテキストを基に制御
- **LSMフックを選ぶ基準**
  - read/writeシステムコールが呼び出したと分かる(区別可能
  - read/writeシステムコールが呼ばれた際，必ず呼び出される
  - アクセス対象ファイルを特定できる情報が得れる

13

Mouri Lab

read/writeが呼び出すLSMフック

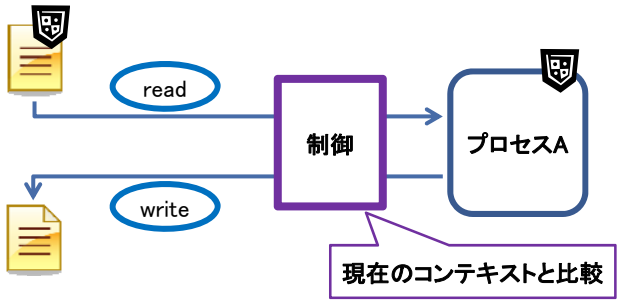
- **int security_file_permission**
  - struct file *file, int type（MAY_READ,MAY_WRITE）
  - ファイルにアクセスする際，パーミッションのチェックを行う
- **int security_file_free**
  - struct file *file
  - file->f_securityに保存されているsecurity構造体の開放を行う

14

Mouri Lab

実装済みの機能

- **保護対象ファイルの保護ポリシの読み込み**
- **保護ポリシとコンテキストを比較し，ファイルへの読み書きを制御**
- **複数のプロセスを制御可能**

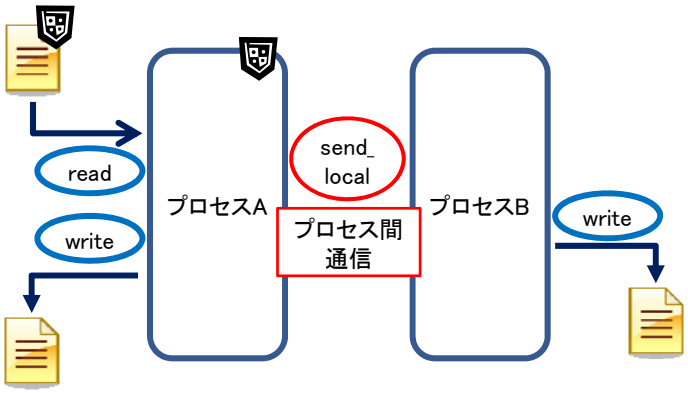


15

Mouri Lab

実装中の機能

- **同一計算機内のプロセス間通信の制御処理**
  - ファイルの読み書き以外に，情報漏洩が発生する可能性がある



16

Mouri Lab

今後の課題

- **共有メモリを介したプロセス間通信の制御**
  - 共有メモリの書き込み，読み出しはOSが検知できない
- **システムコールの実行履歴の取得法**
  - システムコールの実行結果や実行回数を基に制御を行う必要
  - LSMでは，システムコールのフックができない

17

おわりに

- **Salvia**
  - 情報漏洩を防ぐ　アクセス制御を行うOS　導入時に制約
- **LSM**
  - セキュリティモジュール作成のフレームワーク
  - LSMを用いて作成することで，移植性が向上し，導入が容易に
- **LSMを用いたモジュール作成法**
  - 適切なLSMに対応する関数を作成し，登録
- **今後の課題**
  - 同一計算機内のプロセス間通信（共有メモリ）の制御
  - システムコールのログ取得法

Privacy-aware OS *Salvia* における データフローを主体としたアクセス制御手法

井田 章三
立命館大学大学院理工学研究科

1 はじめに

近年，計算機が普及し，個人情報は電子データとして保存，管理されている．それに伴ない，記憶媒体やネットワークを通じて個人情報が流出する事件が頻繁に起きている．情報漏洩事件は，デジタルデータを劣化なく高速にコピーできるという特徴に加え，記憶媒体の大容量化や情報通信の発達により，事件の発生頻度，規模ともに増大し，深刻な社会問題となっている．

文献 [1] では，2007 年に報道された個人情報漏洩事件の調査・分析を行った．文献によると，情報漏洩の原因として「紛失・置き忘れ」,「管理ミス」,「盗難」,「誤操作」の比率が最も高い．特に「管理ミス」は前年と比較するとその比率は大幅に高くなっている．つまり，企業や自治体は情報漏洩対策を行っているが，その現状は対策が不十分で，情報漏洩を発生させてしまっている．

また，情報漏洩事件の事例として,「紛失・置き忘れ」,「管理ミス」,「盗難」では，顧客などの個人情報を記憶媒体に保存し，外部に持ち出したことが原因となっていることが最も多い.「誤操作」では，顧客に他の個人の情報を電子メールに添付して送信してしまったという事例が報告されている．これらの事例は，正当なアクセス権限を持つユーザによる情報漏洩ということができる．暗号化などの既存のセキュリティ技術は，外部からの攻撃を防ぐことを目的とした技術であって，このような情報漏洩を防ぐことはできない．よって，このような情報漏洩を防止する技術の開発は急務だといえる．

以上の背景より，我々は，このような正当なアクセス権限を持つユーザによる情報漏洩を防ぐことを目的とした Privacy-aware OS *Salvia*(以下 *Salvia*) を開発している [2]．*Salvia* は，プロセスによるファイルやソケットなどの情報漏洩の可能性がある計算機資源に対する操作を制御することにより，情報漏洩を防ぐ．すなわち，計算機資源に対する操作の制御をプロセス単位としている．しかし，プロセス単位の制御では情報漏洩とは関係のない操作まで制御してしまい，プログラムの動作を必要以上に制限してしまう可能性があることが判明している．そこで解決策として，現在，データフローを主体としたアクセス制御手法の開発を行っている [3].

以下本稿では，2 章で今まで開発を行ってきた *Salvia* の概要と課題を説明し，3 章でその課題の解決策として提案するデータフローを主体としたアクセス制御手法について述べ，4 章で現在判明している課題を説明し，5 章で本稿のまとめを述べる．

2 Privacy-aware OS Salvia

2.1 概要

データ保護ポリシ プライバシデータは，そのデータ提供者の意志により利用目的や提供範囲が異なる．そのため，プライバシデータの漏洩を防止するためには，各データごとに異なるアクセス制御を行う必要があり，かつそのアクセス制御にはデータ提供者の意志が反映されなければならない．よって *Salvia* では，ユーザは，ファイルごとにデータ保護ポリシを設定可能である．データ保護ポリシには，当該ファイルにアクセスしたプロセスに課す各計算機資源に対するアクセス制御と，その制御が行われる条件を記述する．制御条件には，時刻や IP アドレス，電波強度などのプロセスや計算機の状況を使用でき，これらは *Salvia* が，コンテキストとして抽出する．コンテキストにより *Salvia* は，特定の時間，場所のみ閲覧，変更，転送可能なファイルを設定できる．

データ保護方式 *Salvia* は，ファイルアクセスは，プロセスの open システムコールにより開始する点に着目し，ファイルオープンの前処理としてファイルに設定されているデータ保護ポリシを読み出し，ファイルをオープンしたプロセスをアクセス制御の対象とする．アクセス制御の対象となったプロセスでは，各計算機資源へのアクセスが制限される．計算機資源へのアクセスは，システムコールによって行われるため，*Salvia* は，システムコールの実行の可否をコンテキストとデータ保護ポリシから判断することにより，アクセス制御を実現する．

2.2 Salvia の課題

前述の通り *Salvia* は，保護ファイルオープン以降，プロセスにその保護ファイルのデータ保護ポリシに基づくアクセス制御を課す．しかし，この手法では，保護ファイルの関わらない処理までアクセス制御の対象としてしまう可能性がある．例えば，図 4 では，保護ファイルの open 以降に非保護ファイルに write をしようとしても，その write システムコールも，保護ファイルへの書き込みでないにも関わらず，禁止されてしまう．この動

An access control method based on data flow in Privacy-aware OS *Salvia*

†Shozo Ida
†Graduate School of Science and Engineering, Ritsumeikan University

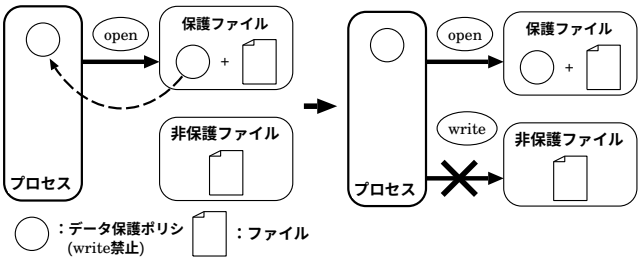


図 1　*Salvia* における問題点

作により，長命なプロセスや扱うファイル数が多いプログラムでは，ユーザの意図した動作とならない可能性がある．例えば，一時ファイルを作成するプログラムやログファイルや設定を保存するプログラムが挙げられる．

この問題は，アクセス制御の主体がプロセスなので，プロセスが行う処理を，保護ファイルを扱うものとそうでないものとの区別がつけられないために発生する．よって問題を解決するには，プロセスが行う各処理を区別し，より粒度の細かいアクセス制御を行う必要がある．プロセスの各処理の区別は，プログラム中のデータの流れを解析することによって実現させることを考えている．データの流れの解析は，コンパイラのデータフロー解析を利用する．次章より，コンパイラのデータフロー解析を利用したアクセス制御について説明する．

3　提案手法

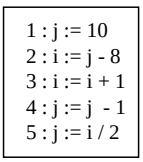
3.1　データフロー解析

コンパイラは，プログラマが作成した原始言語プログラムを解析し，より簡単なアルゴリズムを使い，コードを速く，サイズの小さいものに変換した後，目的プログラムを生成する．このコード変換をコード最適化という．コード最適化には，プログラム中の変数定義の流れ，つまりデータフローを把握することが必要不可欠であり，そのために行うことがデータフロー解析である．データフロー解析については，文献 [4][5] が詳しい．コンパイラが解析できるデータフローは様々であるが，本提案手法では，定義-使用連鎖を作成するデータフロー解析を利用する．

定義-使用連鎖は，各変数定義について，そこで定義された値がどこの文で使用されるかをその変数の使用文の集合で表したものである．定義-使用連鎖の簡単な例を図 2 に示す．プログラム例は，3 番地コードとなっている．定義-使用連鎖により，保護ファイルのデータを読み込む命令文とそのデータを使用する文の間の流れを可視化できる．

3.2　アクセス制御方式

保護ファイルが格納された変数定義の定義-使用連鎖と，それ以外の変数定義の定義-使用連鎖は，別のものとして扱うことができる．よって，定義-使用連鎖をアクセス制御の主体とすれば，プログラム中の各処理を区別



定義文		使用文
1行目	j	2行目
		4行目
2行目	i	3行目
3行目	i	5行目
4行目	j	無し
5行目	i	無し

図 2　プログラムと定義-使用連鎖

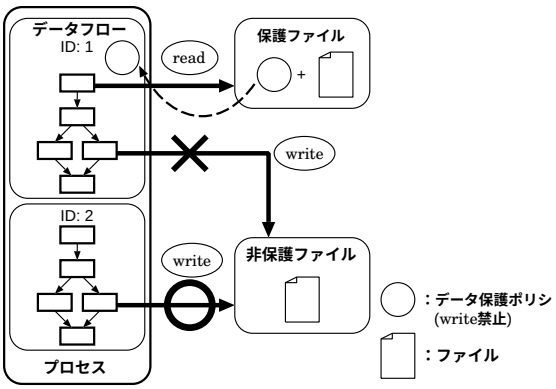


図 3　データフローを利用した *Salvia*

することができる．その様子を図 3 に示す．プロセスが保護ファイルのデータを読み込むと，読み込んだ定義-使用連鎖に保護ポリシに基づくシステムコールの制御が課せられ，非保護ファイルの定義-使用連鎖はアクセス制御の対象とならないため，write を実行できる．データフローに基づくアクセス制御方式を以下に示す．

1. コンパイラが生成した定義-使用連鎖に対し，データフロー ID を割り当てる．
2. 保護データ読み込み時，保護データが格納されている変数が属するデータフロー ID に対して，データ保護ポリシを適用する．
3. 監視対象のシステムコールが実行された場合，データが格納された変数が属するデータフロー ID に対して適用されたデータ保護ポリシの有無を確認する．
4. データ保護ポリシが適用されている場合は，保護データが含まれている可能性があるため，ポリシに従ってシステムコールの実行の可否を判断する．

OS が実行時に，上記の処理を行うには手順 2，3 において，システムコールが発行された時点で使用されている変数が属するデータフロー ID を特定できなければならない．解決方法として，システムコールの発行元の命令アドレスからデータフロー ID を求める．

具体的には，コンパイラとリンカでシステムコール・データフロー対応表を作成する．システムコール・デー

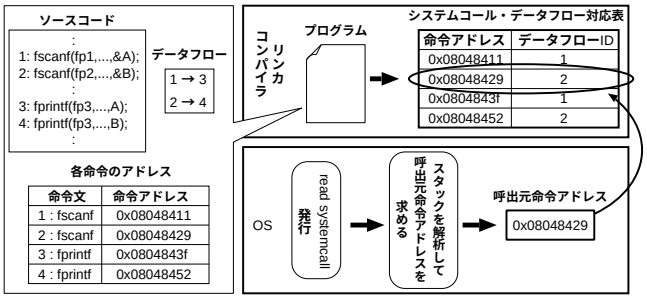


図 4　データフロー ID 特定の流れ

タフロー対応表とは，システムコール呼出しの大元となるユーザプログラムの命令アドレスとデータフロー ID を対応付けさせたものである．OS 実行時のシステムコールの発行元の命令アドレスを検出できれば，その命令アドレスとシステムコール・データフロー対応表に記録してあるシステムコールの呼出元命令アドレスを比較し，そのシステムコールで使われる変数が属するデータフロー ID を求めることができる．OS 実行時のシステムコール発行元命令アドレスの検出は，システムコール実行時にプロセスのスタックを解析することにより求める．以上のデータフロー ID 特定の流れを図 4 に示す．

4　データフロー ID の作成に関する考察

コンパイラのデータフロー解析は，ある変数定義が到達する可能性のある全ての点を解析する．また，ポインタの定義文は，そのポインタが指す可能性のある変数全ての定義文として解析する．このように，コンパイラが解析するデータフローは，プログラム実行時の処理のフローより大きくなる．これにより，提案手法におけるデータ保護ポリシの適応範囲が肥大化しすぎて，意図した通りにアクセス制御の主体を分割できない可能性がある．その例の 1 つとして，システムコールを発行しない定義文が挙げられる．定義-使用連鎖は，1 つの定義点と複数の使用点の繋がりを表すが，定義-使用連鎖とその定義-使用連鎖の使用点で定義された変数の定義-使用連鎖を別のものとして解析する．しかし，この 2 つの定義-使用連鎖が読み込んだデータは同じものだが，データフロー ID が違うものとなってしまい，実際には保護データを読み込んでいるデータフロー ID にデータ保護ポリシを適用できない．図 5 は，1 行目で buf1 にファイルから読み込んだデータを格納し，2 行目で buf2 に buf1 のデータをコピーし，3 行目で buf2 を別ファイルに出力するプログラムの定義-使用連鎖である．見ての通り，2 つのデータフロー ID が作成されるが，buf1 と buf2 のデータは同じものであるにも関わらず，それぞれが所属しているデータフロー ID は，別のものとなっている．よって，fp1 が保護ファイルだった場合，データフロー ID1 にしか，データ保護ポリシが適用されず，3 行目の write を制御することができない．

この問題を解決するには，2 行目のデータコピー時に

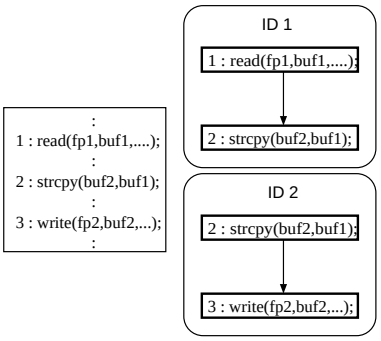


図 5　システムコールを発行しない定義文

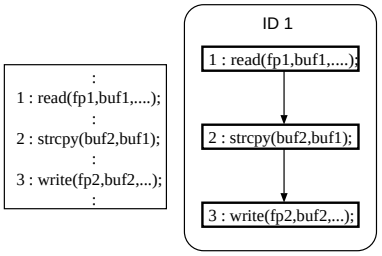


図 6　2 つの定義-使用連鎖の連結

データフロー ID2 にデータフロー ID1 のデータ保護ポリシを伝搬させるか，1 行目の保護データ読み込み時点でデータフロー ID2 にもデータ保護ポリシを適用させる必要がある．前者は，代入文などのシステムコールが発行されない命令文においてデータ保護ポリシをデータフロー ID 間に伝搬させなければならないが，*Salvia* のアクセス制御処理のトリガーはシステムコールの発行時のみなので，代入命令実行時に動的にデータ保護ポリシの伝搬処理を行うのは，現在の *Salvia* の仕様では不可能である．後者は，定義-使用連鎖の作成方法次第で実現可能である．具体的には，ある定義-使用連鎖の使用文で定義された定義-使用連鎖を，その使用文の属する定義-使用連鎖と連結させる．図 6 に例を示す．図 5 のデータフロー ID1 の使用文かつデータフロー ID2 の定義文である 2 行目を連結させて，1 つのデータフロー ID としている．これならば，1 行目で保護データを読み込んだ場合でも，3 行目の write にデータ保護ポリシを適用でき，アクセス制御対象とすることができる．

しかし，この場合，前述したデータ保護ポリシの適応範囲の肥大化が起こる可能性がある．図 7 は，2 つのファイルからデータを読み込み，そのどちらかを buf3 にコピーして別のファイルに buf3 を出力するプログラムを，図 6 のデータフロー ID 作成法でデータフロー ID を作成した例である．5 行目の命令文が 2 つのデータフロー ID に属していることが分かる．これにより，以下に示す処理手順を踏むと過剰なアクセス制御が発生する．

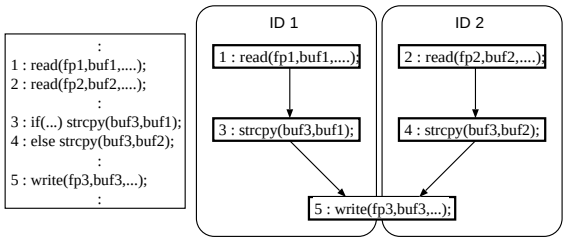


図 7　2 つのデータフロー ID に属する命令文

1. 1 行目で保護データを buf1 に読み込み，データ保護ポリシをデータフロー ID1 に適用する．
2. 2 行目で非保護データを buf2 に読み込む．
3. if 文による場合分けで 4 行目の命令文を実行し，buf2 を bu3 にコピーする．
4. 5 行目の write 時に，手順 1 で読み込んだデータ保護ポリシに基づくアクセス制御を受ける．

この手順では，buf3 は保護データではないにも関わらず，アクセス制御の対象となってしまっている．この動作は，1 行目実行時に 3 行目が実行されるものとしてデータ保護ポリシを適用してしまうために発生する．このようなデータ保護ポリシの適応範囲の肥大化は，厳密な解析が行えない関数間のデータフローにおいても起こると考えられる．

以上より，プロセスの処理をデータ毎に分割し OS に認識させることは，コンパイラのデータフロー解析のみでは，実現が困難であると考えられる．よって，他のプログラム解析技術の利用・併用を検討する必要があると考えられる．

5　おわりに

本稿では，現在我々が開発している Privacy-aware OS *Salvia* の概要とユーザの意図通りに動作しないプログラムが存在する可能性について述べ，その解決案としてデータフローを主体としたアクセス制御手法を提案し，現時点で判明している課題について説明を行った．

今後は，システムコール・データフロー対応表を作成するコンパイラの開発や，4 章で述べた課題の解決のため別のプログラム解析技術の調査を行っていく．

参考文献

[1] NPO 日本ネットワークセキュリティ協会: JNSA 2007 年情報セキュリティインシデントに関する調査報告書，2008.

[2] 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，Vol.47, pp.1-15，2006.

[3] 毛利 公一: “システムコール間のデータフローに基づくデータ漏洩防止手法,” コンピュータセキュリティシンポジウム 2007(CSS2007) 論文集，pp.631-636，2007.

[4] Alfred V.Aho, Ravi Sethi, Jeffrey D.Ullman 著, 原田 賢一 訳: “コンパイラ I/II 原理・技報・ツール,” サイエンス社，1990.

[5] 中田 育男 著: “コンパイラの構成と最適化,” 朝倉書店，1999.

Privacy-aware OS *Salvia* におけるデータフローを主体としたアクセス制御手法

立命館大学大学院
毛利研究室
井田 章三

発表内容

- はじめに
- Privacy-aware OS *Salvia*
- *Salvia* の課題
- 提案手法
- 課題
- おわりに

はじめに(1/2)

- 近年，個人情報は電子データとして計算機で管理されている
- 個人情報の漏洩問題が深刻化している
  - 電子データは，劣化なく高速にコピー可能
  - 情報通信技術の発達や記憶媒体の大容量化により漏洩の頻度，規模が増大
  - 個人情報漏洩は，プライバシー問題，企業のイメージダウン，補償問題を引き起こす

はじめに(2/2)

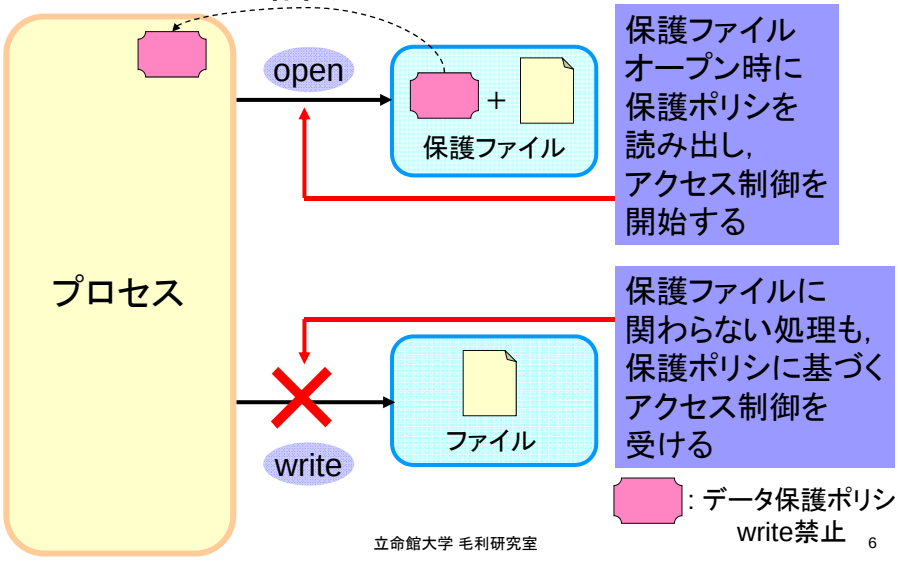
- 個人情報漏洩の原因
  1. アプリケーションの操作ミス
  2. 従業員による機密情報の持ち出し
  3. ウィルス感染による流出
  4. クラッカーによる侵入，攻撃
  5. セキュリティの設定ミス
- 1，2のような正当なアクセス権限を持つユーザによる情報漏洩の頻度が最も高い
- 既存のセキュリティ技術は，外部攻撃の防止が目的

Privacy-aware OS *Salvia*

Privacy-aware OS *Salvia*

- アクセス制御機構を備えたOS
  - アプリケーションの信頼度に関わらず統一的に制御可能
- 保護方式
  - 保護単位：ファイル
    - 保護したいファイルと保護ポリシを一組にして管理
  - 制御対象：プロセス
    - プロセスがデータを漏洩させる動作を制御
    - プロセスが実行するシステムコールをチェックすることで制御
  - コンテキストの利用
    - ユーザや計算機の状況のこと
    - ユーザ，時間，計算機の場所，IPアドレス　など

Salvia の課題



問題が発生するプログラム

- 一時ファイルを作成するプログラム
  保護ファイルオープン以降，一時ファイルを作成できなくなる可能性
- ログを保存するプログラム
  ログを保存できなくなる可能性
- 複数のファイルを同時にオープンするプログラム
  保護ファイルオープン以降，他のファイルを
  保存できなくなる可能性
- スレッドを用いたプログラム
  1つのスレッドの保護ファイルオープン以降，他のスレッドも
  アクセス制御を受ける可能性

原因

- アクセス制御の主体がプロセス
  - 保護ファイルの関わる処理とそうでない処理を区別する
    ことなく，アクセス制御の対象としてしまう

より粒度の細かいアクセス制御の実現の為に，アクセス制御の主体をプロセスより細かいものへ

_Salvia_とコンパイラを提携させたシステムの提案
データフローをアクセス制御の主体に

データフロー解析

- プログラム中の変数定義の流れを解析
- 定義-使用連鎖を作成するデータフロー解析を用いる
  - 各定義文で定義された変数がどこで使用されるかを その変数の使用文の集合で表したもの

```
1:  j := 10
2:  i := j - 8
3:  i := i + 1
4:  j := j - 1
5:  i := i / 2
```

定義文	使用文
1行目　j	2行目 4行目
2行目　i	3行目
3行目　i	5行目
4行目　j	なし
5行目　i	なし

定義-使用連鎖

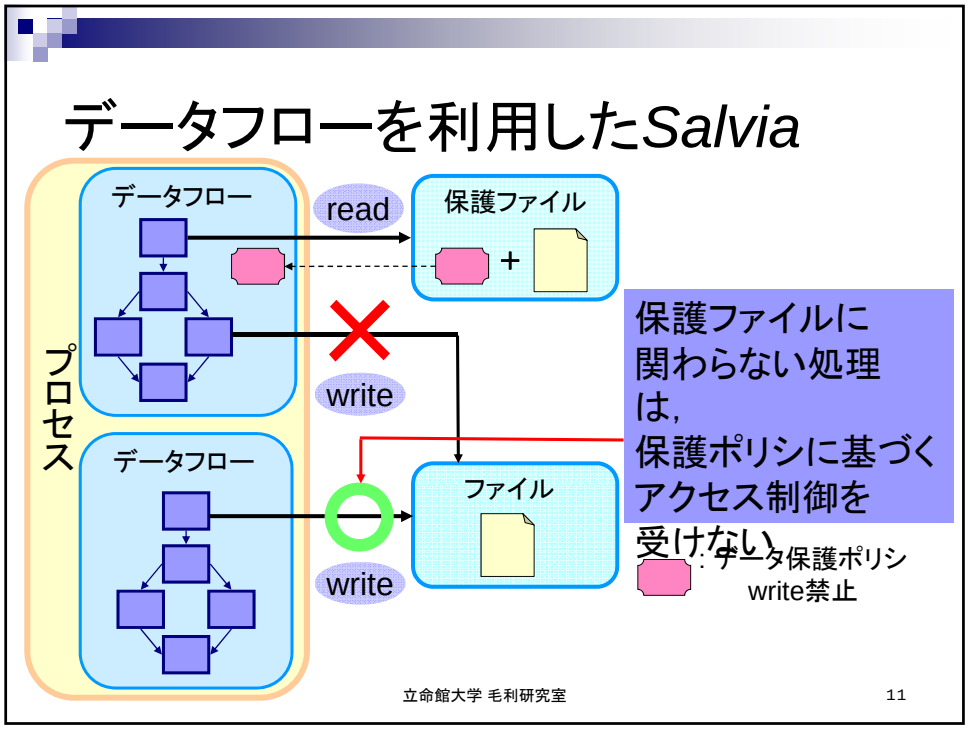
- 機密ファイルのデータの読み込み以降，データ漏洩する可能性がある
- readによる定義文と，そこで定義された変数の使用文の間の流れに着目する

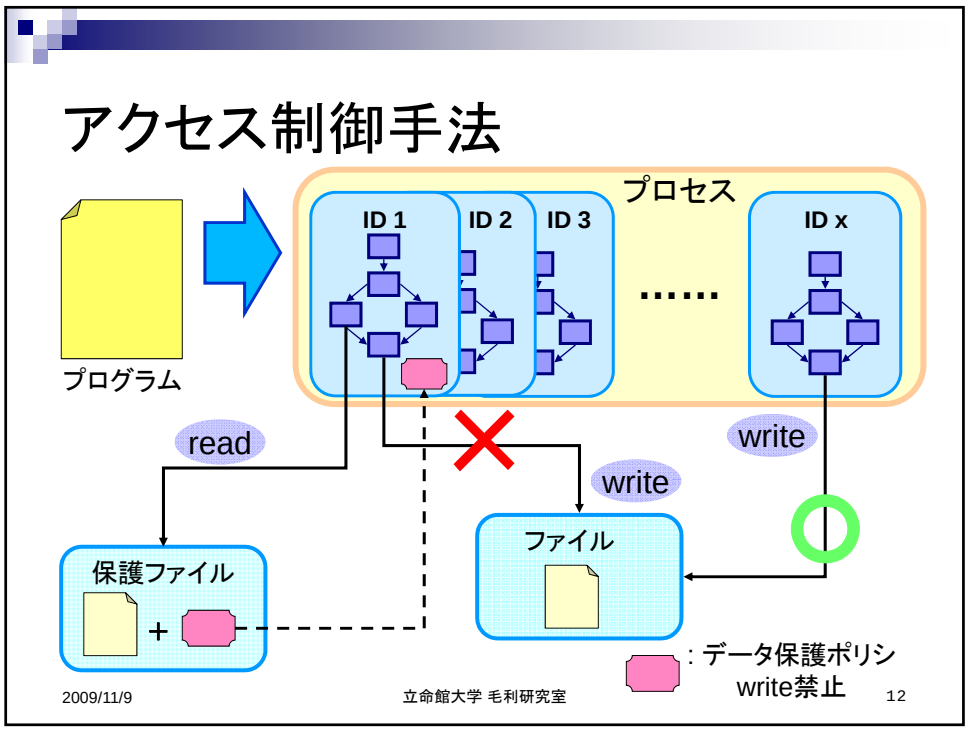
```
:
fp1 = fopen(.....);
:
:
fscanf(fp1,...,&A);
:
:
fprintf(fp2,...,A);
:
```

機密ファイルをオープン　open

機密ファイルのデータを読み込む　read

データを書き込む　write


データフローを利用したSalvia
プロセス
データフロー
read
保護ファイル
+
write
データフロー
write
ファイル
保護ファイルに
関わらない処理
は,
保護ポリシに基づく
アクセス制御を
受けない
: データ保護ポリシ
write禁止
立命館大学 毛利研究室
11


アクセス制御手法
プログラム
プロセス
ID 1
ID 2
ID 3
.....
ID x
read
write
write
保護ファイル
+
ファイル
: データ保護ポリシ
write禁止
2009/11/9
立命館大学 毛利研究室
12

データフローIDの特定

システムコールの呼び出し元命令アドレスから
データフローIDを求める

- スタックバックトレーサ
  - プロセスのスタックを解析し，システムコールの呼び出し元命令アドレスを求める
- システムコール・データフロー対応表
  - システムコールの呼び出し元命令アドレスとデータフローIDを対応付けした表
  - コンパイラとリンカで事前に作成

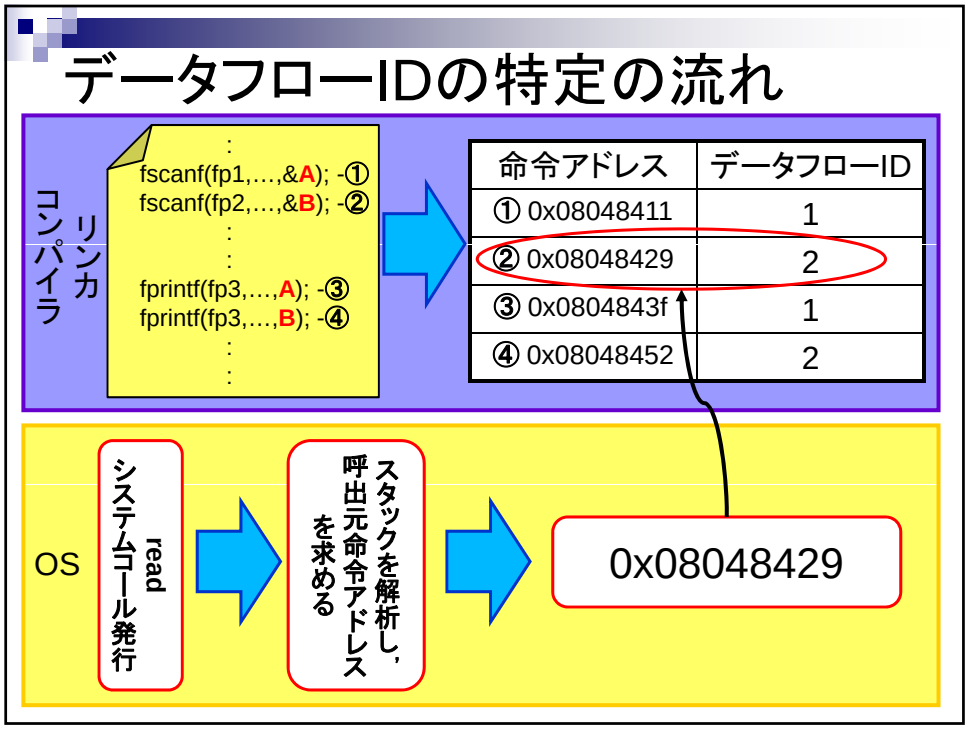
システムコール・データフロー対応表

```
:
fscanf(fp1,…,&A); -①
fscanf(fp2,…,&B); -②
:
:
fprintf(fp3,…,A); -③
fprintf(fp3,…,B); -④
:
:
```

定義文	使用文	ID
① fscanf	③ fprintf	1
② fscanf	④ fprintf	2

命令文	命令アドレス
① fscanf	0x08048411
② fscanf	0x08048429
③ fprintf	0x0804843f
④ fprintf	0x08048452

命令アドレス	データフローID
0x08048411	1
0x08048429	2
0x0804843f	1
0x08048452	2

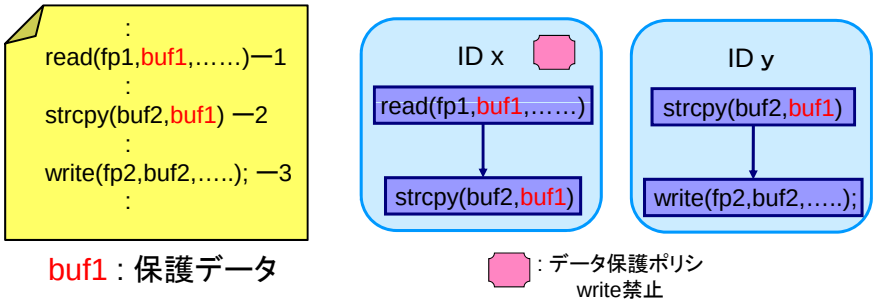


課題

- システムコール・データフロー対応表を作成するコンパイラ，リンカの開発
- スタックバックトレーサの*Salvia*への実装
- データフローIDの作成・特定・管理
  - 定義-使用連鎖の作成
  - 関数間のデータフロー

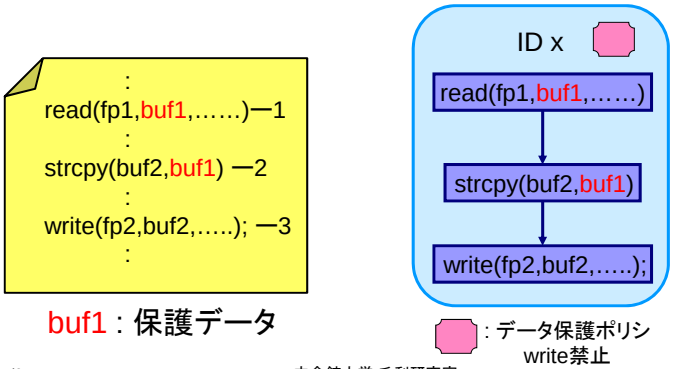
システムコールを発行しない定義文(1/2)

- 定義文に保護データが使用されていた場合，保護ポリシを伝播させる必要がある
- システムコールが発行されないと*Salvia*は動的な処理を行うことができない



システムコールを発行しない定義文(2/2)

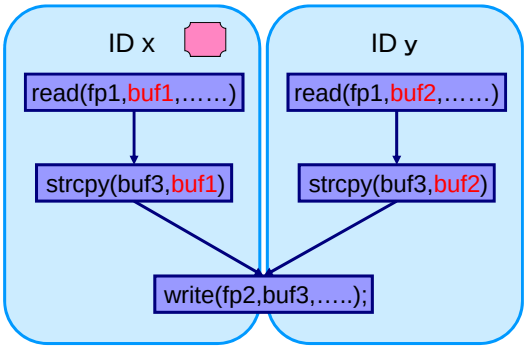
- buf1とbuf2のデータフローIDを同じものにする必要がある
- データフローIDを使用される変数の属するデータフローIDと同じものにする



問題点

- 分岐を挟むと必要以上のポリシを適応してしまう可能性
- この例だと，5行目のbuf3使用時にデータフローID1と2両方の制限がかかる

```
        :
1 : read(fp1,buf1,……);
2 : read(fp2,buf2,……);
        :
3 : if(……)  strcpy(buf3,buf1);
4 : else  strcpy(buf3,buf2);
        :
5 : write(fp3,buf3,…..);
        :
```



buf1：保護データ

関数間のデータフロー

- コンパイラは，各関数毎にデータフロー解析を行う
  厳密な関数間のデータフローは，解析不能
- 無理やりデータフローを繋げると大域的になる可能性
- インライン展開は，展開しきれないコードが存在する

origin解析

- 従来のデータの格納領域に特別な領域を追加し，もとのデータに関する情報を保持する
- 代入が行われる場合，代入する値の情報も拡張領域に代入

```
1: int a, b, c;
2: a = 1;
3: b = a + 2;
4: c = b;
```

2行目: a に定数 1 に関する情報を代入
3行目: b に a と定数 2に関する情報を代入
4行目: c に b に関する情報を代入

拡張領域には，その変数が持つ値の起源となるデータの情報が常に保存される

おわりに

- Privacy-aware OS *Salvia* 概要
- *Salvia*の課題
- データフローを主体としたアクセス制御手法
  - アクセス制御の主体を定義-使用連鎖に変更
    - 各定義-使用連鎖をデータフローIDで管理
  - システムコールが属するデータフローIDの特定方法
    - システムコール・データフロー対応表の利用
    - スタックバックトレーサの作成
  - データフローIDの作成・特定・管理に関する問題
    - 定義-使用連鎖の作成
    - 関数間のデータフロー

機密情報の拡散追跡機能におけるプロセス間通信経路の監視手法

植村 晋一郎† 田端 利宏† 谷口 秀夫†
†岡山大学大学院自然科学研究科

1. はじめに

近年，計算機で機密性の高い情報を扱う機会の増加ともに,機密情報の漏えい事例が増加している．そこで,我々は機密情報が計算機内に拡散する状況を追跡し，機密情報を有する資源を把握する機能（以降，機密情報の拡散追跡機能）を提案し，本機能を用いた情報漏えいの防止機構を提案している．本論文では，機密情報の拡散追跡機能におけるプロセス間通信経路の監視手法について述べる．

2. 機密情報の拡散追跡機能の概要

機密情報はファイルの形式で存在し，どれが機密情報に相当するかは利用者が指定していると仮定する．また,機密情報を有する可能性があるため，機密情報の拡散追跡機能によって監視されているファイル，プロセスおよびプロセス間通信用資源を管理対象と呼ぶ．

機密情報の拡散は，ファイル形式で存在する機密情報をプロセスがオープンしてその内容を読み込み，さらに他のプロセスやファイルなどへその内容を伝えることで起こる．プロセスが情報を伝搬する経路は(1)ファイル操作，(2)子プロセス生成，(3)プロセス間通信である．これらの処理を監視し，機密情報が拡散した可能性のあるプロセス，ファイルおよびプロセス間通信を仲介する通信用資源を管理対象とすることで，機密情報の拡散を追跡する．

3. プロセス間通信経路の監視手法

Linux におけるプロセス間通信のパイプ，FIFO,メッセージキュー，および共有メモリについて通信の経路を監視し，機密情報の拡散を追跡する手法について述べる．

パイプの場合，送信側のプロセスがパイプへ write システムコールによりデータを書き込むことで，パイプ内に機密情報が拡散する．次に，受信側のプロセスが read システムコールによりパイプからデータを読み出すことで，受信側プロセスに機密情報が拡散する．このことから，write と read のシステムコールが拡散の契機となる．そこで，これらのパイプに対してデータを読み書きするシステムコールを契機とし，パイプ全体を管理対象とすることで監視を行う．

メッセージキューの場合，送信側のプロセスがメッセージキューへ msgsnd システムコールによりデータを送信することで，キュー内部のメッセージに機密情報が拡散する．次に，受信側のプロセスが msgrcv システムコールによりメッセージキューからデータを受信することで，受信側プロセスに機密情報が拡散する．このことから，msgsnd と msgrcv のシステムコールが拡散の契機となる．そこで，これらのメッセージキューへデータを送受信するシステムコールを契機とし，メッセージキュー内のメッセージを管理対象とすることで監視を行う．

共有メモリの場合，データのやり取りを行う前に，両プロセスとも shmat システムコールにより共有メモリをアタッチする．次に，送信側のプロセスが共有メモリにデータを書き込むことで，共有メモリ内に機密情報が拡散する．次に，受信側のプロセスが共有メモリからデータを読み出すことで，受信側プロセスに機密情報が拡散する．このことから，shmat システムコールによる共有メモリアタッチ後の読み書きが拡散の契機となる．共有メモリの読み書きをシステムコール単位で捕捉するのは困難である．そこで，共有メモリをアタッチする shmat システムコールを契機とし，共有メモリを管理対象とすることで監視を行う．

4. オーバヘッドの評価

本機能の実装によりオーバヘッドが発生する,プロセス間通信関連のシステムコールの実行時間を測定した．オーバヘッドは，最大でパイプを破棄する close の場合の 38%であり，システムコール処理時間へ極端に大きな影響はない．

5. おわりに

機密情報の拡散追跡機能において，プロセス間通信を監視する手法について述べた．残された課題として，実 AP を用いた事例の評価と共有メモリにおけるアタッチ操作以外による監視方法の検討が挙げられる．

The Monitoring Method of Interprocess Communication Route on the Diffusion Tracing Function of Classified Information
Shinichiro Uemura†, Toshihiro Tabata† and Hideo Taniguchi†
†Graduate School of Natural Science and Technology, Okayama University

機密情報の拡散追跡機能における プロセス間通信経路の監視手法

植村晋一郎†, 田端 利宏†, 谷口 秀夫†
† 岡山大学大学院自然科学研究科

はじめに

- 計算機で機密性の高い情報を扱う機会の増加
  - ⇨機密情報漏えい事例の増加
- 情報漏えいの主な原因は，紛失や盗難に次いで計算機の操作ミスや管理ミス
- 漏えいの経路は，Winnyなどのファイル交換ソフトウェアを原因としたネットワーク経由が増加
- 情報漏えい防止のためには，機密情報がシステム上のどこにあるかを把握することが重要

⇩

- OSレベルで機密情報の拡散を追跡し，計算機外部への漏えいを検知する機能（機密情報の拡散追跡機能）の実現
- 本機能におけるプロセス間通信経路の監視方法を提案

機密情報の拡散追跡機能の概要

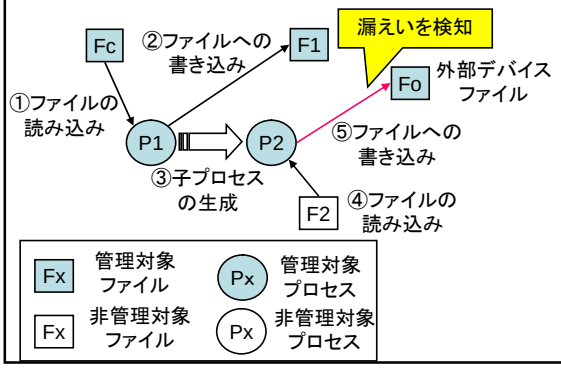
- 機密情報を持つ可能性のあるプロセスとファイルを把握
  - ファイル操作（read, write），子プロセス生成(fork)およびプロセス間通信に着目し，機密情報の拡散を追跡
  - 機密情報の伝播された可能性のあるプロセスとファイルを管理対象として登録

⇩

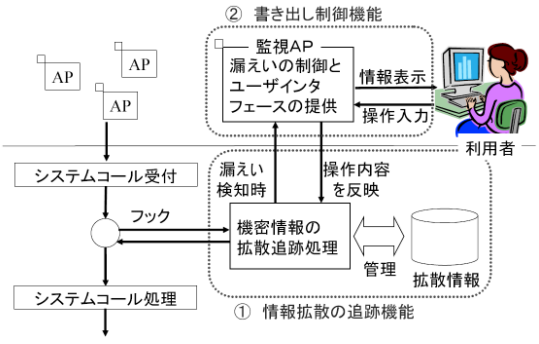
- 管理対象プロセスから外部への書き出しを漏えいとして検知
- 漏えい検知時の動作として，利用者へ警告を表示

管理対象：機密情報を所持している可能性があるとして機密情報の拡散追跡機能によって監視されている状態

機密情報の拡散追跡機能の動作



情報漏えいの防止機構の概要



プロセス間通信の監視における課題

＜監視の対象とするプロセス間通信の種類＞
(1) パイプとFIFO
(2) メッセージキュー
(3) 共有メモリ

⇩

機密情報を計算機内部の他プロセスに伝播させる可能性有り

⇩

プロセス間通信の経路を監視し，機密情報の拡散を追跡

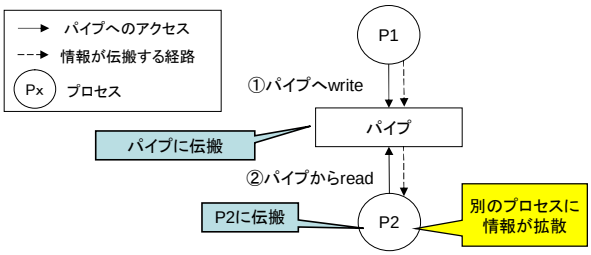
＜監視における課題＞
(1)プロセス間の機密情報の拡散の契機となるシステムコール
(2)機密情報の拡散追跡機能へ適用させる手法

関連システムコール一覧表

プロセス間通信の種類	システムコール	情報拡散の契機
パイプ，FIFO	pipe	×
	mknod	×
	write	○
	read	○
	close	×
メッセージキュー	msgget	×
	msgsnd	○
	msgrcv	○
	msgctl	×
共有メモリ	shmget	×
	shmat	△
	shmdt	×
	shmctl	×

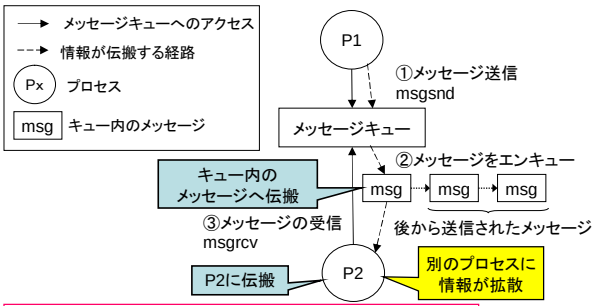
＜凡例＞
○：情報の拡散に強く関係
△：情報の拡散に一部関係
×：関係なし

機密情報の拡散経路（パイプ）



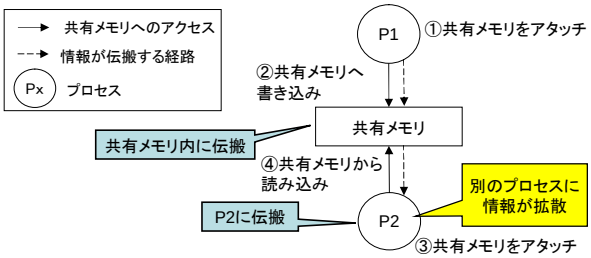
(1)writeとreadのシステムコールが拡散の契機
(2)ファイルと同様にパイプを管理対象として監視

機密情報の拡散経路（メッセージキュー）



(1)msgsndとmsgrcvのシステムコールが拡散の契機
(2)メッセージを管理対象として機密情報の拡散を監視

機密情報の拡散経路（共有メモリ）



共有メモリの読み書きをシステムコール単位で捕捉するのは困難

(1) 共有メモリのアタッチ操作を監視に利用
(2) 共有メモリの領域を管理対象として監視

設計方針

＜プロセス間通信の監視における要件＞

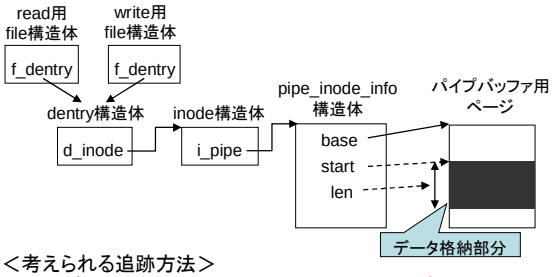
(要件1) プロセス間通信の監視に漏れがないこと
⇨ 見逃しのない確実な情報漏えいの検知
(要件2) プロセス間で拡散する機密情報を即座に追跡できること
⇨ 情報漏えい防止のための警告出力の確実な制御

＜機能への要望＞

(要望1) 機密情報の拡散追跡が的確であること
⇨ 情報漏えいの誤検知の防止
(要望2) 処理のオーバヘッドが小さいこと
⇨ プロセス間通信の処理時間への影響の抑制

パイプの監視方法

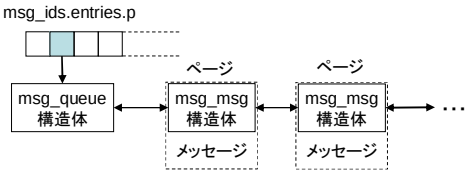
＜パイプの構造＞



＜考えられる追跡方法＞
(1)パイプに関連付けられるinodeを利用してパイプ全体を監視
(2)パイプバッファ内の機密情報が格納されている部分を監視
⇨ 方法(1)を採用

メッセージキューの監視方法

＜メッセージキューの構造＞



＜考えられる追跡方法＞
(1)識別IDを利用してメッセージキュー全体を監視
(2)キュー内の機密情報が格納されているメッセージを監視

いずれの方法もオーバヘッドは同等
⇨方法(2)を採用

共有メモリの監視の契機

＜共有メモリにおける機密情報の拡散の契機＞
共有メモリをアタッチしたプロセスを監視する3つの契機を定義
(1) アタッチした共有メモリが管理対象の場合
　アタッチを実行したプロセスを管理対象に追加
(2) 管理対象のプロセスが共有メモリをアタッチした場合
　共有メモリを利用している全てのプロセスを管理対象に追加
(3) 共有メモリをアタッチしているプロセスが管理対象になった場合
　共有メモリを利用している全てのプロセスを管理対象に追加

⇩

＜実現に必要な機能＞
- 共有メモリをアタッチしているプロセスの把握
- 管理対象となっている共有メモリの把握

共有メモリの利用状況の管理

shmid1	PID1	PID2	NULL	NULL
shmid2	PID2	PID3	PID4	…
shmid3	…			

…

⋮

＜共有メモリ利用状態テーブル＞
- どの共有メモリをどのプロセスが使用しているか把握
- 共有メモリ識別IDとプロセスIDで管理
- 共有メモリのアタッチのたびに，対応する共有メモリ識別IDのエントリにアタッチしたプロセスのIDを追加

＜管理対象の共有メモリの把握＞
- 共有メモリ識別IDを用いて管理対象の共有メモリを把握

評価

＜評価内容＞
(1)システムコールのオーバヘッドの評価
　フック対象となるシステムコールのオーバヘッドを測定
⇨システムコール処理時間への影響の評価
(2) 評価事例の実装機能への適用
　コピー＆ペーストにより共有メモリを利用する作業を想定した事例を実装機能に適用
⇨プロセス間通信の監視の有効性と他プロセスへの影響の評価

＜評価環境＞
CPU: Celeron D 2.8 GHz
Memory: 768 MB
OS: Linux-2.6.0

オーバヘッドの評価

	機能実装前	機能実装後		オーバヘッド
		非管理対象プロセスの通信	管理対象プロセスの通信	
write (pipe)	5.9	6.1	6.9	1.1
read (pipe)	5.2	5.4	5.4	0.2
close(pipe)	5.8	6.5	7.1	1.3
msgsnd	8.2	8.2	9.2	1.1
msgrcv	6.3	6.6	7.4	1.1
msgctl	5.0	6.2	6.3	1.3
shmat	14.1	17.2	16.5	2.3
shmdt	7.0	7.6	7.4	0.4
shmctl	11.0	12.5	12.6	1.5

＜測定対象＞
データの送受信，および通信資源を破棄するシステムコール
最大でパイプを破棄するcloseの場合の1.1μs(38%)
⇨システムコール処理時間へ極端に大きな影響なし

おわりに

＜本発表のまとめ＞
(1) 機密情報の拡散追跡機能におけるプロセス間通信の監視方法を提案
(2) 管理対象プロセスがデータを送信したパイプ，FIFO，メッセージキューを監視
(3) 共有メモリについてアタッチ操作を契機として監視
(4) システムコールのオーバヘッドへの影響はほとんどなし

＜残された課題＞
(1) 実際のAPを利用した更なる事例の評価
(2) 共有メモリのアタッチ操作以外による機密情報の拡散追跡の方法の検討

編集後記

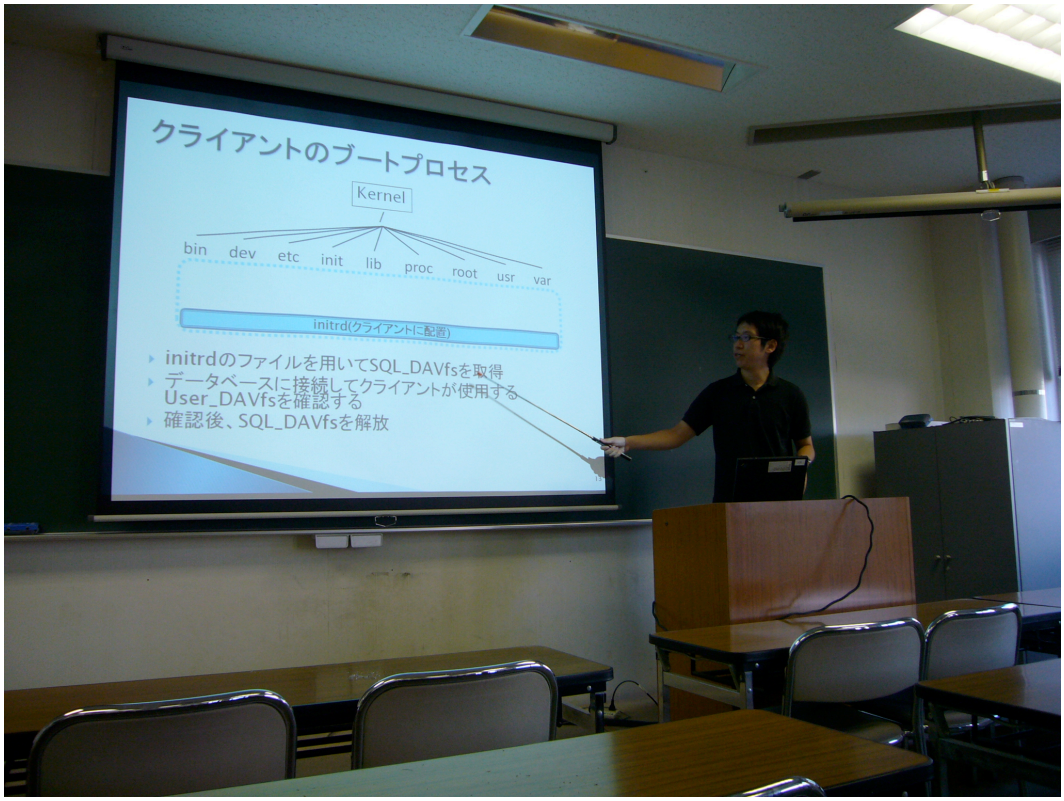
私自身の JSASS への参加は，昨年に引き続き今年で 2 回目となります．今回は幹事の一人として参加させていただくことになりましたが，ちょっとした手伝い程度で終わってしまい，毛利先生にはいろいろご負担をおかけしてしまったように思います．

今回，私が所属する研究室からは，大学院学生が 4 名発表させていただきました．普段，研究室内で繰り返される議論からついつい考え方が近視眼的になりがちなところを，様々な方から異なる立場，異なる視点によるご意見をいただくことのできる，貴重な機会になったと思います．

今回のプログラムからいくつか特徴的なキーワードをひろってみると，Cell，シンクライアント，組み込み，クラスタ，省電力，QoS，エミュレータ，ソフトウェア教育，匿名通信，アクセス制御，情報漏洩，MANET，DHT，センサネットワークといったものが目に止まります．これらのキーワードを眺めていると，研究分野の広がりや，最近の研究の動向などが垣間見え，興味深く感じられます．議論の接点が無くなるほど個々の発表のトピックが発散してしまうと大変ですが，発表内容の勢いや新鮮さを失わないよう，JSASS ではこれからも柔軟に様々なテーマを取り上げてもらえたらと思います．

今回は残念ながら準備不足のため，JSASS 前後に岡山名物を味わうなどのイベントを実行できなかったのが個人的な心残りですが，会場の岡山大学はキャンパスも広く，落ち着いた雰囲気でよいところでした．ローカルアレンジメントをご担当された田端先生，どうもありがとうございました．余談ですがキャンパスの裏にあるキッチン稲穂は，たたずまいもメニューの構成も学生向け定食屋の王道という感じで感銘を受けました．

立命館大学　横田裕介



今年の JSASS は 2 日間に渡り岡山大学で行った．研究室で連れだって参加してくれたところもあり，多くの方に参加してもらうことができた．今回の研究発表はいずれもが，その成果に到達するために多くの努力が行われてきたものであると思う．研究室内で普段から協力して研究を進め，発表でもお互いにサポートしあっていることがうかがえた．JSASS が，このような協力が研究室を越えて行われるようになるきっかけになれば嬉しく思う．

本シンポジウムはシステムソフトウェアを扱うものであるが，今年も様々なシステムや手法の発表があった．特に，オペレーティングシステムの機能に関する研究など，構築に手間がかかり，多くの試行錯誤が行われたであろう発表もあり，頼もしく感じた．また，実用されるシステムを定め，その要求を満たすよう，目的が明確になっている発表もあった．何が課題であるかが意識され目標がはっきりしているため，非常に興味深く聞くことができた．これとは逆に，具体的な応用よりは興味あることを突き詰めてユニークなシステムを構築したという発表もあった．このような発表は，実装における細かな工夫や苦労話を聞くことができたいへん面白い．このように，今年の JSASS も様々な種類の発表があり，それぞれが研究室内だけは難しいような意見交換をできたのではないかと思う．今後も JSASS がシステムソフトウェアに関する研究のために有用な場であることを願っている．

龍谷大学　芝 公仁

先進的基盤ソフトウェア
Joint Symposium for Advanced System Software 2009 (JSASS2009)

3 巻 1 号（通号 3 号）オンライン版　2009 年 11 月 30 日発行

編　　集　毛利 公一，横田 裕介，芝 公仁，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