Java スレッドの永続化、移送と移動エージェントへの応用

早稲田大学 理工学部
首藤一幸 村岡洋一

従来研究
応用
手法と実装
実行コンテクストを扱う際の問題
システム
性能評価
まとめ

スレッド移送システム

- 実行コンテクストを含めて移送
- スレッドのディスク等への保存が可能
- 移送スレッドのクラス定義を自動転送
  - 移送先へのインストールが不要
- 非均質環境で動作
  - プロセッサ, OS を選ばない
  - JVM の機種非依存性による

- 動作環境
  - Java Development Kit (JDK) 1.1
  - UNIX 互換 OS で動作

従来研究

- Object Migration
  - 実行コンテクストは移送しない
- プロセス移送
  - 同一のプロセッサ、OS 間で移送
  - OS のサポートが必要
- Java用移動エージェント
  - 移動先で実行を継続できない
  - 既存システム* は、移動の際に実行コンテクストを捨てる
    - * Voyager, Aglets Workbench, Kafka, odysssey

応用

スレッド移送の応用

- 移動エージェントの記述
  - ローカルに動作するコードとほぼ同じ記述法
- プログラムの耐故障性向上
  - 実行中の状態を保存するコードを書く必要がない
- 負荷分散
  - メカニズム (API, place の CUI) を提供
  - ポリシーは提供していない
- Disconnected operation

応用：移動エージェントの記述

プログラミングインタフェースを比較

- Voyager 1.0 - Agent enhanced ORB
  - ⊖ 移動： moveTo() メソッド

  移動先、移動先で呼び出したいメソッド名が引数

```
public class MobileAgent extends Agent {
  public void callback(Object arg) {
    System.out.println((String)arg);
  }
  public void go (String address) {
    moveTo(address, "callback", "Hello.");
    System.out.println("This statement is never executed.");
  }
}
```

実行が継続されないため

ローカルに動作するコードとは記述法が異なる

応用：移動エージェントの記述

- MOBA - スレッド移送システム
  - Thread クラスの代わりに MobaThread クラスを使用
  - スレッドの生成

```
Thread t = new MobaThread(...);
t.start();
```

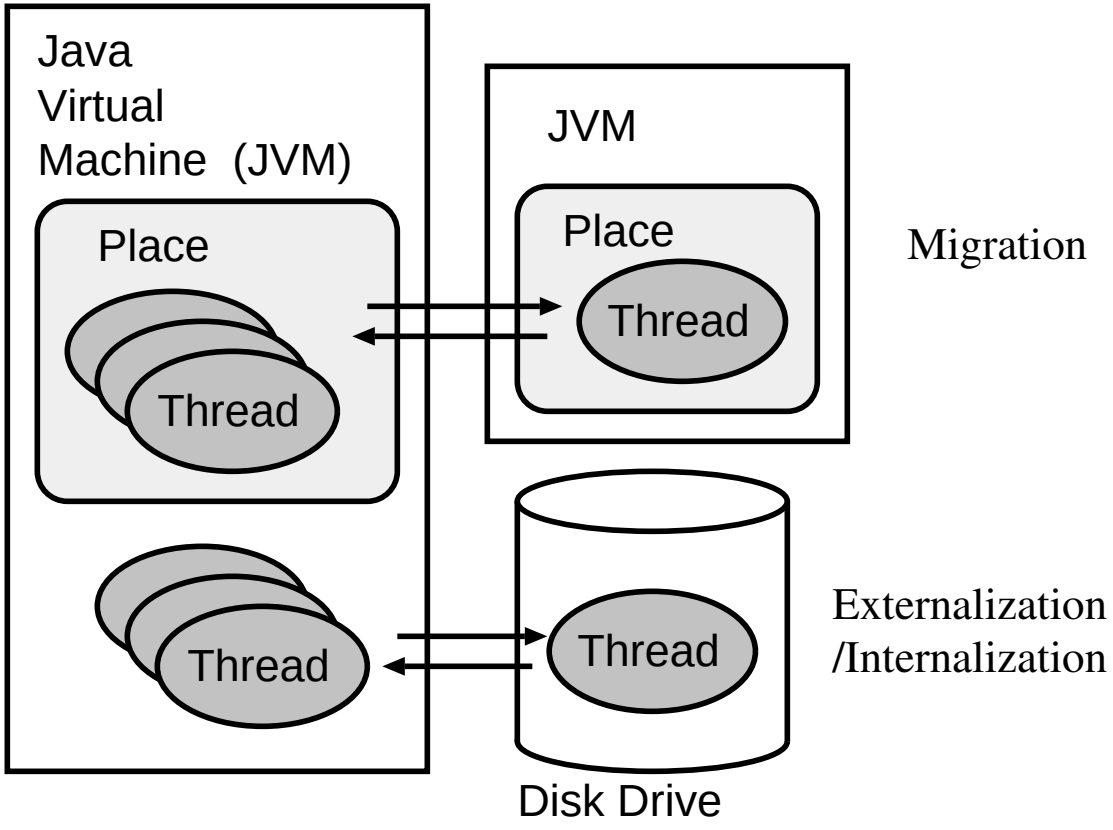
  - スレッド自身による移動

```
...
MobaThread.goTo(移動先計算機);
...
```

移動先で実行が継続されるため
移動を意識せずに記述できる

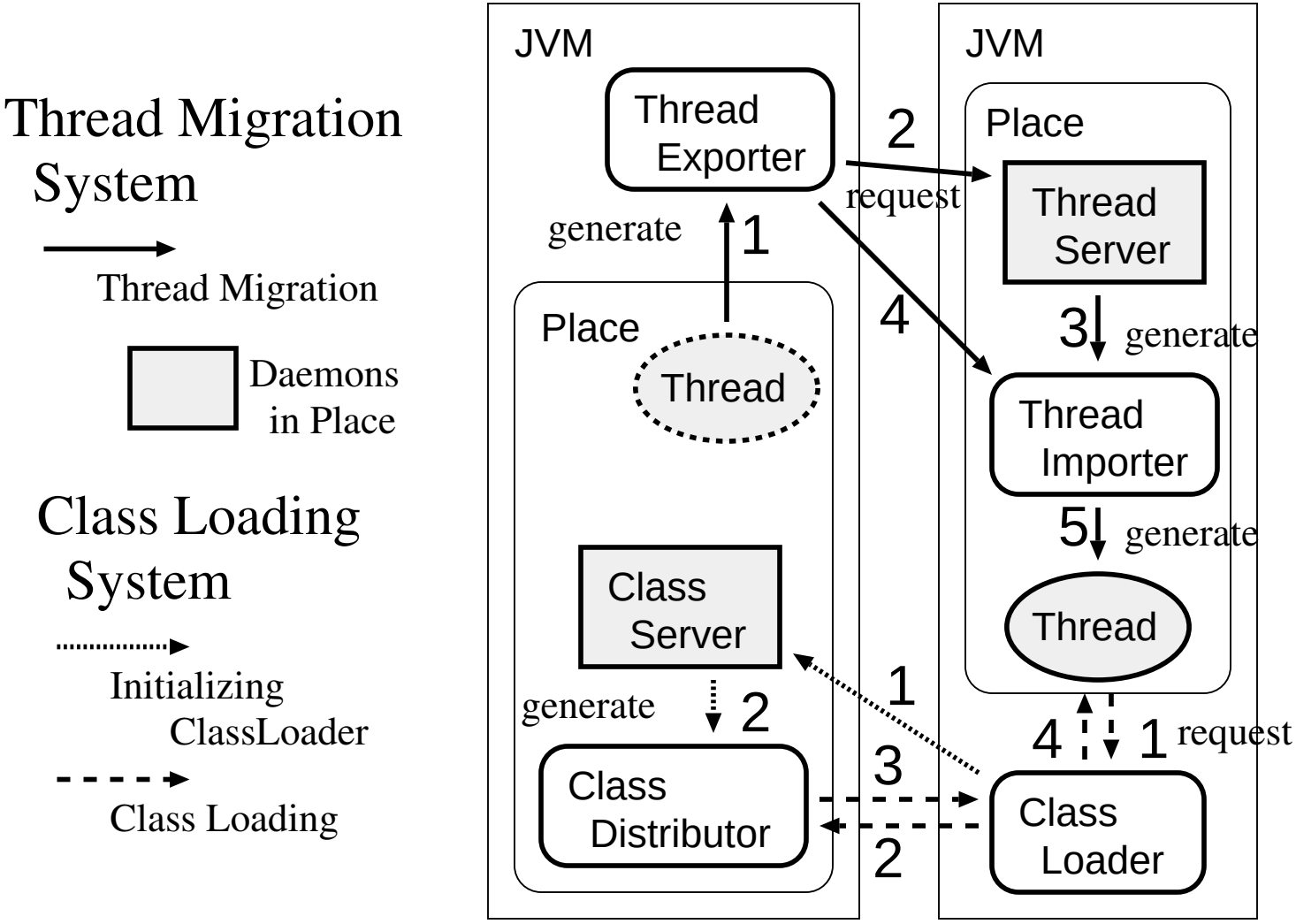
システム構成

クラスライブラリと place から成る

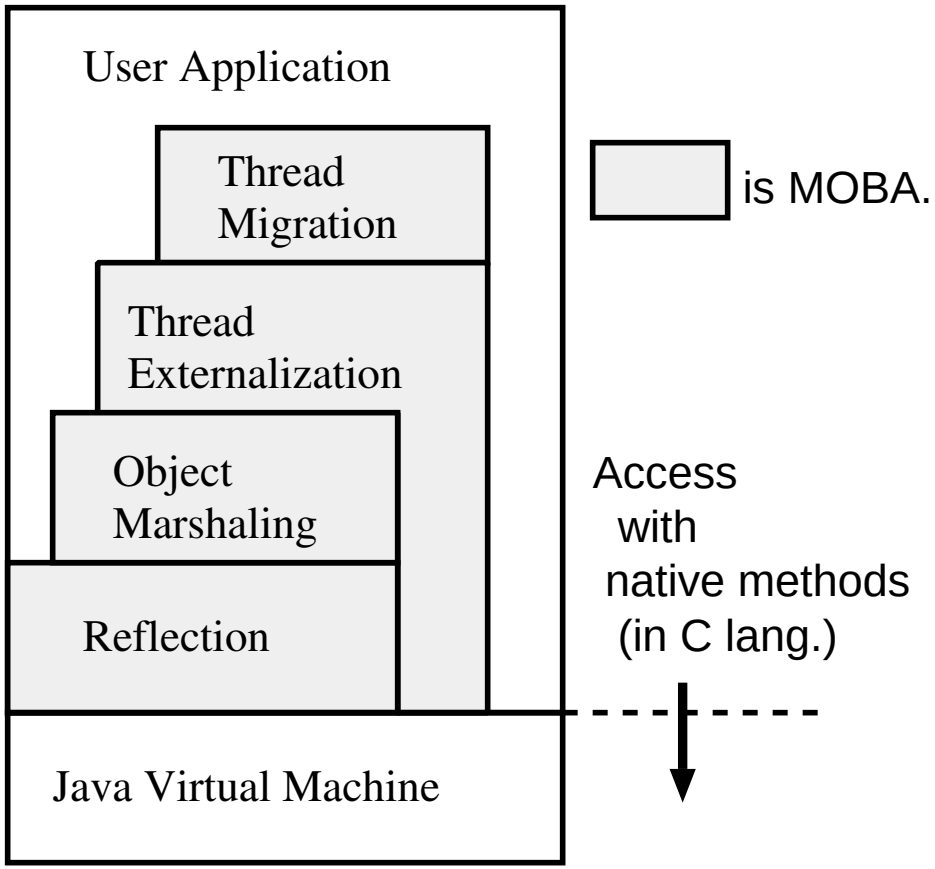


place のサブシステム

スレッド移送系
クラスロード系
コマンドシェル系 - 利用者からのコマンド(移送,永続化,...)を受ける



手法：要素技術


User Application
Thread Migration
Thread Externalization
Object Marshaling
Reflection
Java Virtual Machine
is MOBA.
Access with native methods (in C lang.)

スレッド移送

- スレッドの状態の転送
  - スレッドを外部化, 転送, 内部化
- 移送先へのクラス定義の転送
  - 移送先で実行を継続するためにコードの移動が必要

スレッド外部化

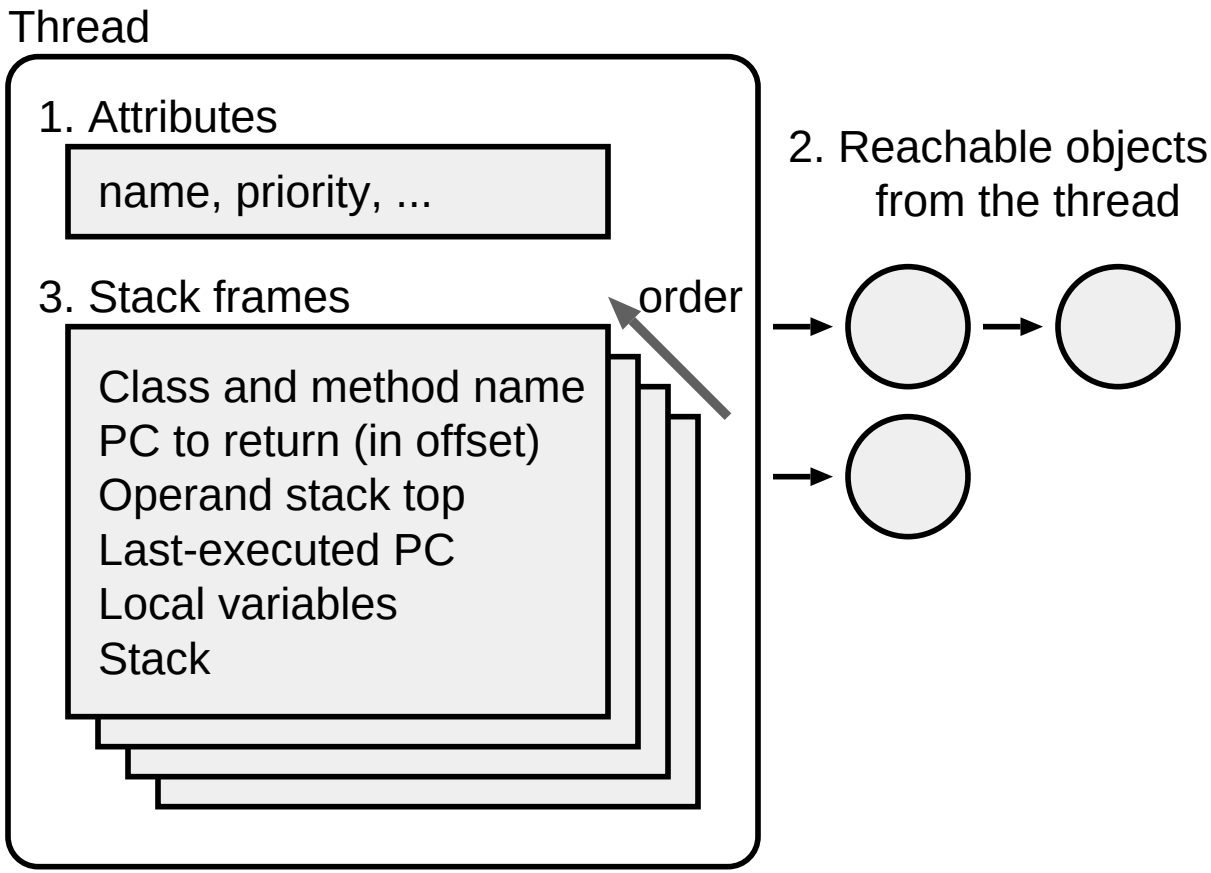
- 外部化 - スレッドの状態のバイト列化
- 内部化 - JVM内への再構成

手順

- 前処理
  - 自身の外部化のためには別スレッドを起動
- 排他制御, 一時停止
  - 対象スレッドのモニタを取得
- バイト列化

スレッド外部化

□バイト列化



Object Marshaling

オブジェクト群 (のグラフ) のバイト列化

- 問題： 位置依存のオブジェクト
  - ファイル記述子など、単純な外部化、内部化では問題あり
  - 無効化や修整が必要
- クラスの分類が必要
  - 位置非依存
  - 位置依存
    - 同一スレッドによる marshal, unmarshal では対処不要
    - 常に対処が必要

現在

一部のクラスに対する無効化処理が実装されている

クラス定義の転送

- 必要に応じて (動的に) 自動的に転送される
  - あらかじめ移送先へインストールしておく必要がない
  - 実行に必要なクラス定義が移送先でも得られることを保証
  - 移動エージェント向き

- 一貫したクラス名空間
  - クラス定義はスレッド生成元からロード
  - クラス名空間が移送先に依存しない

今後

- 静的な転送による最適化

実行コンテクストを扱う際の問題

Java 仮想計算機 (JVM) 上の実行コンテクストを
永続化、移送する際の問題

- バイトコードから機械語への変換系
  - JIT コンパイラ, static コンパイラ, ...
- ネイティブメソッド
  - 実行コンテクストへのアクセスに必要

機械語への変換系

高速実行のために、仮想計算機のコードから
プロセッサの機械語へ変換する実行系が増えている
例)

- JIT コンパイラ ： クラスのロード時
- static コンパイラ： 実行前

JVM 中立な永続化のためには
実行コンテクストの機種非依存表現が必要

プロセッサネイティブなコードの実行中、
JVM のプログラムカウンタ(PC)は機械語の命令列を指す
↓
機種非依存表現を得られない

機械語への変換系

対策

- PC をバイトコード上のオフセットに変換する
  - JVM の JIT インタフェースとして用意されていない
  - バイトコードと機械語命令は 1対1 対応しない
- スレッドごとに変換の可否を制御
  - JVM に手を入れる必要がある
- JVM ごと変換を抑制する
  - 現状の解決策
  - 実行性能に難あり

ネイティブメソッド

Java ではなく C や C++ で記述されたメソッド

Java 言語では実行コンテクストにアクセスできない
JVM 内へのアクセスにネイティブメソッドが必要

利用のデメリット

- 機種依存するので実行形式の可搬性が損なわれる
- インストール時にコンパイルが必要
- アプレットから利用しづらくなる

既存の移動エージェント

- 既存システムはネイティブメソッドを使用していない
- 移動の際に実行コンテクストを捨てる

性能評価

- □移送遅延
  - ⊖ MOBA
  - ⊖ Voyager 1.0.1 - Agent enhanced ORB
- □スループット
  - ⊖ MOBA と ORB 2種
    - ⊖ RMI
    - ⊖ HORB 1.3.b1
      (RMI と同等のバッファリングをする unofficial 版)

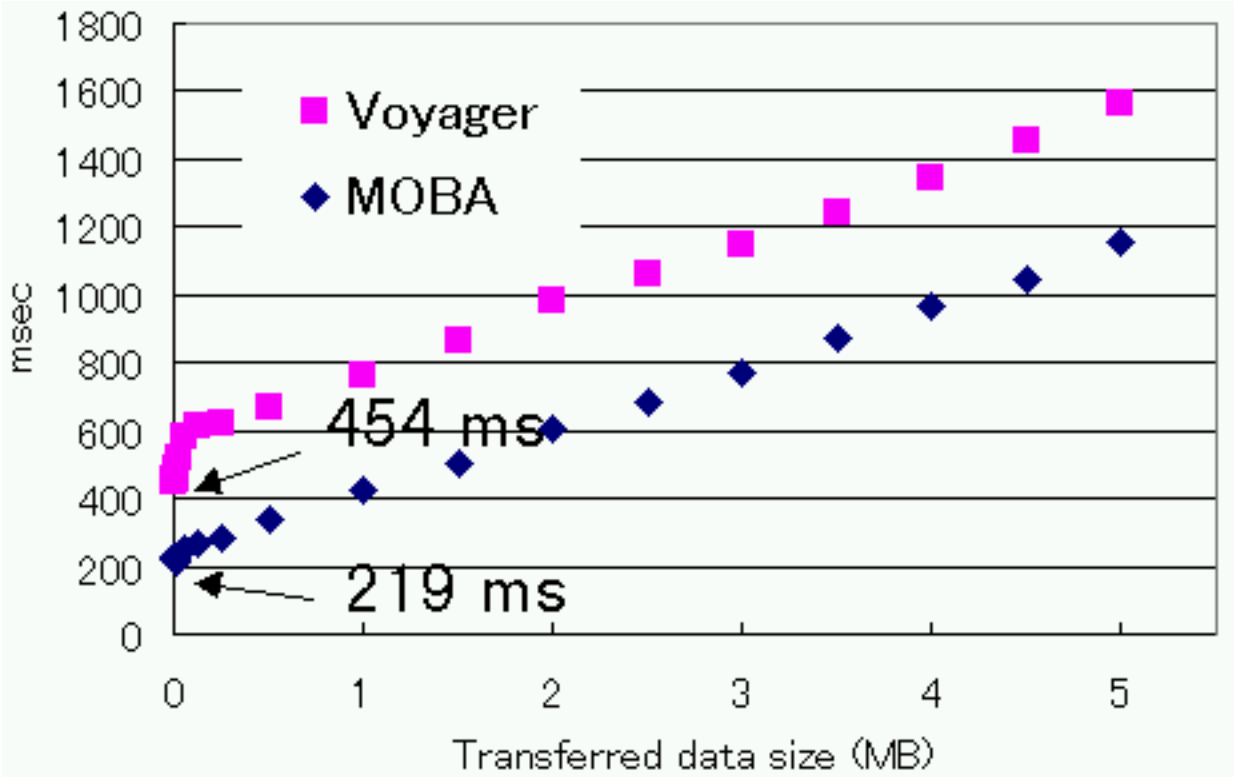
実験環境

- ⊖ マシン1: Sun (UltraSPARC 168MHz)
- ⊖ マシン2: Sun (UltraSPARC 167MHz)
- ⊖ JVM:　JDK 1.1.5, Sun の JIT コンパイラ
- ⊖ ネットワーク: 100Mbps Ethernet, リピータ1台経由

性能評価： 移送遅延

移動エージェントを往復させ、片道の移動時間を算出

- MOBA　: 219 msec
- Voyager : 454 msec

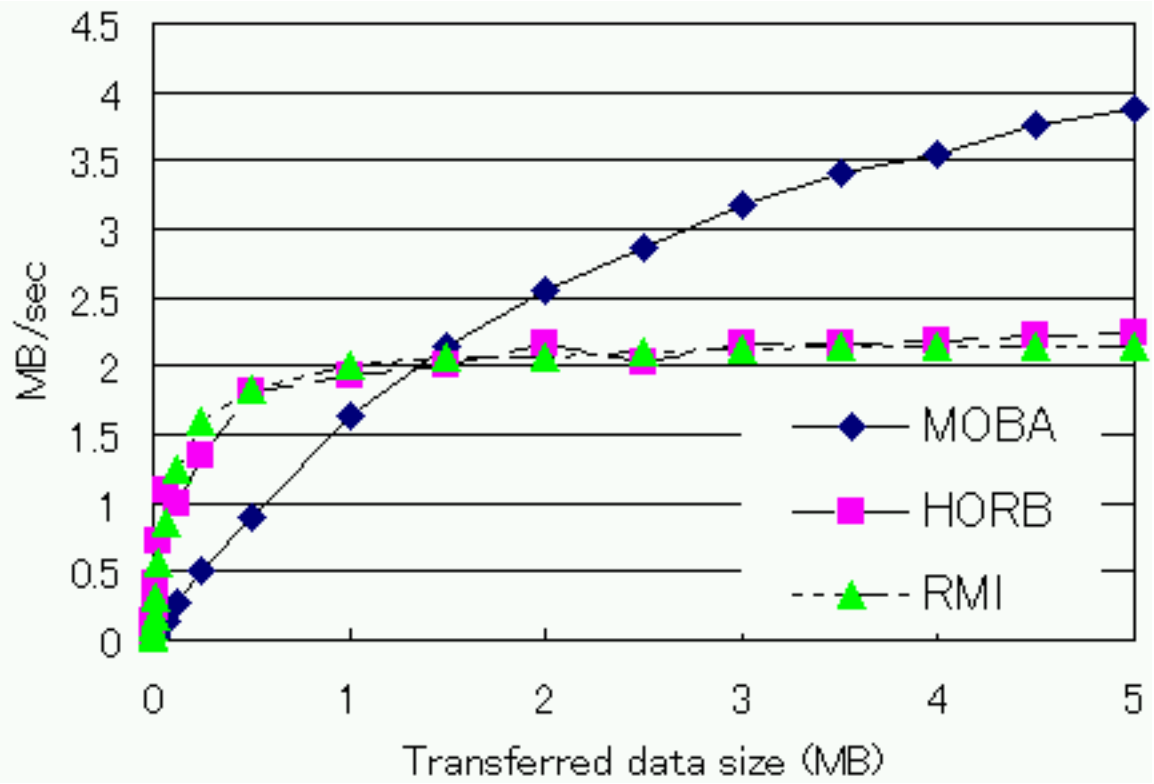


性能評価：スループット

□ORB 2種

　double型配列を引数に遠隔メソッド呼び出し

□MOBA　配列を保持して移動



まとめ

- 実行コンテクストを含めたJavaスレッドの移送を実現
- ローカルに動作するコードと変わらない
  移動エージェントの透過的な記述
- 各種応用
  移動エージェント, 負荷分散, 耐故障性向上
- 実行コンテクストを扱う際の問題を明らかにした
- 小さい移送遅延、高いスループット
- 公開済み http://www.shudo.net/moba/

- 課題
  - オブジェクトは単純にコピーしていることによる問題の解決
  - 機械語への変換系との同時動作