分散処理におけるアプリケーションと実行環境に対する適応的なスケジューリング

宮崎 脩斗[†1]　Mahdillah[†1]　秋山勇樹[†1]　吉田 隆一[†2]

概要：分散処理において，アプリケーションやユーザの立場，実行環境によって求められる性能は異なり，求められる性能によって適切なスケジューリングも異なるので，スケジューリング方式を動的に入れ替える適応的なスケジューリングを考えた．本論文では，分散システムで頻発する実行環境の変化に自律的に対処するオブジェクトとして我々が開発している自己再構成可能オブジェクトにこの適応的なスケジューリングを導入した．自己再構成可能オブジェクトの計算は内包するメタオブジェクト同士のメッセージパッシングによって進むため，優先度別の処理をおこなうためにはそのメッセージパッシングも優先度別におこなわなければならない．この問題を解決するために，優先度別のポートを介したメッセージパッシングを導入した．また，ユーザのスケジューリングポリシーの表現方法を定め，それに従ったスケジューリング方式の入れ替えを実現した．

キーワード：スケジューリング，適応性，分散処理，自律型システム

Adaptive Scheduling to Application and Runtime Environment in Distributed Computing

Miyazaki Shuto[†1]　Mahdillah[†1]　Yuki Akiyama[†1]　Yoshida Takaichi[†2]

Abstract:
In distributed computing, the required criterion of performance differs depend on the applications, the standpoint of the users and runtime environments. So suitable scheduling differs depend on the required criterion of performance. Therefore, we propose adaptive scheduling that can dynamically change a scheduling algorithm to another one which is suitable for the criteria of performance required at each occasion. In this paper, we introduce adaptive scheduling into the self-reconfigurable object that can adapt for environmental change in distributed systems. Since computing of self-reconfigurable object is performed by message passing between the meta-objects which are internal objects of the self-reconfigurable object, the messages must be exchanged in priority order which is provided by a scheduling algorithm. In order to solve this problem, we introduced the message passing between meta-objects via a priority port. And we developed expression method of scheduling policy of users and realized the mechanism which can change scheduling method based on the policy.

Keywords: Scheduling, Adaptability, Distributed computing, Autonomous system

1. はじめに

一般に，アプリケーションソフトウェアの実行に求められる性能はそのアプリケーションの特性やユーザの立場，実行環境によって異なる．たとえば，対話型のアプリケーションは応答性を高めなければならないが，バッチ処理をおこなうアプリケーションであればスループットが重視される．また，エンドユーザにとっては自身の出した要求の応答性さえ高ければ良いが，システムの運用者にとっては

[†1] 現在，九州工業大学大学院情報工学府情報創成工学専攻
Presently with Kyushu Institute of Technology
[†2] 現在，九州工業大学大学院情報工学研究院情報創成工学研究系
Presently with Kyushu Institute of Technology

スループットを最大化することが重要になる．さらに，システムの負荷が小さい場合には応答性を高めることができるが，負荷が大きくなると個々の応答性を犠牲にしてでも仕事を捌くことに専念する必要がある．このような多様な性能はそれぞれに適したスケジューリングをおこなうことで満たすことができる．

求められる性能が異なると，採るべきスケジューリングアルゴリズムも異なる [9], [10]．たとえば，システムの平均的な応答性を高めるためには実行時間の短い仕事から実行するスケジューリングが有効であるが，オーバーヘッドを削減したければ単純な FCFS(First Come First Served) が有効となる．そこで，スケジューリングアルゴリズムをその時々において求められる性能を満たすものへ動的に入れ替えることで，アプリケーションや実行環境に対して適応的なスケジューリングをおこなうことができる．

ところで，現在我々は分散システムにおいて頻発する実行環境の変化に自律的に対処する機構として適応型分散オブジェクト指向環境 Juice[1], [2], [3] を開発している．Juice では従来のオブジェクトモデルを拡張し，実行環境の変化に自律的に対処する能力を個々のオブジェクトに持たせたオブジェクトモデルを用いている．このオブジェクトモデルにおいて，オブジェクトは複数の内部オブジェクトであるメタオブジェクトから構成される．個々のオブジェクトは実行環境の変化を検知すると，内包するメタオブジェクトを入れ替え自身を再構成することで，実行環境に対して随時適切に振る舞う．このため，このオブジェクトを自己再構成可能オブジェクトと呼ぶ．

現状の自己再構成可能オブジェクトは自身の再構成による実行環境変化への対処能力を有する一方で，対話型やバッチ処理などのアプリケーションの特性にあわせたスケジューリングや，時々刻々と変化するオブジェクトの負荷に応じたスケジューリング方式の変更などをおこなうことができない．そこで本研究では，この自己再構成可能オブジェクトに上述した適応的なスケジューリングを導入する．

以降の章構成は以下のとおりである．まず，自己再構成可能オブジェクトについて 2 章で説明し，3 章ではメタオブジェクト間通信について説明する．続いて 4 章と 5 章でスケジューリングポリシーの表現方法と適応的なスケジューリングの実現について述べる．そして，6 章で本研究の評価を述べ，最後に 7 章で本研究についてまとめる．

2. 自己再構成可能オブジェクト

本章では，適応型分散オブジェクト指向環境 Juice のオブジェクトモデルである自己再構成可能オブジェクトモデルについて問題領域の階層別に説明する．

2.1 自己再構成可能オブジェクトにおける問題領域の分離

自己再構成可能オブジェクトが取り扱う問題領域を以下の 3 つの概念階層に分類する．図 1 に階層構造を示し，その各階層について説明する

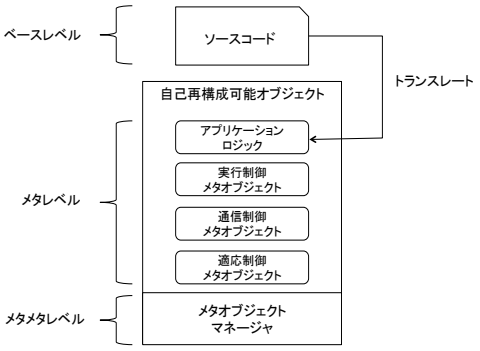

図 1 問題領域の分離

2.1.1 ベースレベル

ベースレベルはアプリケーションプログラマが担当する領域であり，我々はこの領域から非機能的要求を切り離すことで，アプリケーションプログラマに機能的要求の実現のみに専念させる．ベースレベルの成果物は，アプリケーションのコードとなる．

Juice のアプリケーションは複数の自己再構成可能オブジェクトによって構成され，各自己再構成可能オブジェクトが他の自己再構成可能オブジェクトに対しベースレベルのメソッドの実行を要求し合うことで計算が進む．本研究では，そのようなベースレベルの外部から要求されるメソッドに優先度を設定してスケジューリングをおこなう．

2.1.2 メタレベル

メタレベルでは自己再構成可能オブジェクトの非機能的な要求を満たす領域である．アプリケーション開発の現状として，アプリケーションプログラマがこの問題領域を扱っているが，本来はシステム運用者などが解くべき問題領域である．メタレベルの成果物は，メタオブジェクトの実装クラスであるメタレベルモジュールとなる．

以下にメタオブジェクトの例を示す．現在は実装の容易化のために実行制御メタオブジェクトと通信制御メタオブジェクトのみを対象とし，メタオブジェクトを実行環境に応じて適切なものへと入れ替えることによって環境適応を実現している．

- アプリケーションロジック
  アプリケーションのロジックそのものを実現する役割を担い，アプリケーションの状態とコードを保持する．アプリケーションプログラマが記述したソースコードは，我々が提供する専用のトランスレータによってコンパイル時にメタオブジェクトに変換される．従って，アプリケーションプログラマは通常の Java プログラミングを行うだけでよく，Juice に関する知識を必要としない．
- 実行制御メタオブジェクト
  実行制御メタオブジェクトは自己再構成可能オブジェ

クトにおいて，アプリケーションロジックの実行を制御する．実行制御メタオブジェクトの例として，並行実行制御をおこなうものが挙げられる．

- 通信制御メタオブジェクト
  通信制御メタオブジェクトは自己再構成可能オブジェクト同士のメッセージパッシングを実現する役割を担う．通信制御メタオブジェクトは通信プロトコルを実装し，他のオブジェクトから受信したメッセージを実行制御メタオブジェクトへ渡す．通信プロトコル毎に通信制御メタオブジェクトを用意することで同時に複数の通信プロトコルを実現することができる．
- 適応制御メタオブジェクト
  適応制御メタオブジェクトは自己再構成可能オブジェクトにおいて実行環境の情報収集，実行環境の同定，自己再構成可能オブジェクトが採るべき構成の決定をおこなう．

2.1.3 メタメタレベル

メタメタレベルはメタオブジェクトの実行を支援する領域であり，メタオブジェクトマネージャによって実現される．メタオブジェクトマネージャは，1つの自己再構成可能オブジェクトに1つ存在し，メタオブジェクト群の管理や入れ替えなどの役割を担う．

3. メタオブジェクト間通信

自己再構成可能オブジェクトの計算は内部のメタオブジェクトが互いに通信しながら協調的に動作することによって進む．本章ではそのメタオブジェクト間通信について説明する．

3.1 メタオブジェクト間通信

3.1.1 マイクロポート

メタオブジェクトは実行環境に応じて異なるメタオブジェクトへと入れ替えられる可能性があるため，メタオブジェクト間の関係を疎とする必要がある．そのために各メタオブジェクトは通信相手となるメタオブジェクトの参照を直接持たず，メッセージポートを介したメッセージパッシングをおこなう．このメッセージポートをマイクロポートと呼び，各メタオブジェクトはマイクロポートを介し，間接的かつ非同期に計算を進める．

再構成時には入れ替え後のメタオブジェクトが入れ替え対象となるメタオブジェクトに対応していたマイクロポートを引き継ぐことで処理の継続が可能となる．

3.1.2 メタオブジェクト間通信の流れ

各メタオブジェクトは生成された後，メタオブジェクトマネージャから自身のマイクロポートへの参照を取得する．その後，メタオブジェクト間通信を以下の順序に従っておこなう．

(1) 通信制御メタオブジェクトは外部からのメッセージを受信し，復号や伸長などのプロトコル処理をおこなった後，そのメッセージを実行制御メタオブジェクトに対応付けられたマイクロポートへと送信する．
(2) 実行制御メタオブジェクトは自身のマイクロポートからメッセージを受信し，要求されたアプリケーションロジックのメソッドを実行する．ここで必要であれば並行実行制御などの実行制御をおこなう．
(3) 実行制御メタオブジェクトはメソッドの実行結果をメッセージとして通信制御メタオブジェクトに対応付けられたマイクロポートへと返答する．
(4) 通信制御メタオブジェクトは実行制御メタオブジェクトから返答されたメッセージの暗号化や圧縮などのプロトコル処理をおこなった後，送信元へ返答する．

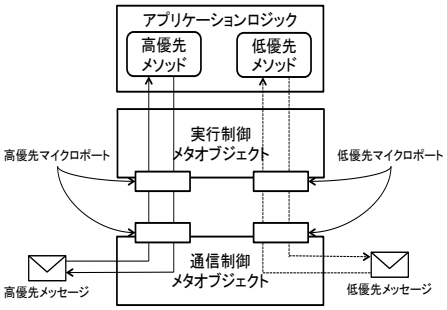

図 2 優先度別マイクロポート

3.2 スケジューリングとの関連

自己再構成可能オブジェクトにおいて，外部から実行を要求されるアプリケーションロジックのメソッドを優先度別に処理するためには，各メタオブジェクト間通信においてもその要求メッセージを優先度別に処理する必要がある．つまり，優先的に実行したいメソッドの実行要求メッセージは優先的にメタオブジェクト間を伝播させていかなければならない．そこで，メソッドの実行要求メッセージに優先度を付け，優先度別のマイクロポートを用いることで優先度別の処理を実現する．優先度別マイクロポートを用いた優先度別の処理の様子を図2に示す．

4. スケジューリングポリシー

第1章で述べたとおり，アプリケーションやユーザの立場，実行環境によって重視する性能は異なり，その多様な性能はそれぞれに適したスケジューリングをおこなうことで満たすことができる．そこで，スケジューリングアルゴリズムをその時々において求められる性能を満たすものへ動的に入れ替える適応的なスケジューリングを考えた．

本章では，適応的なスケジューリングを自己再構成可能オブジェクトに導入するにあたって，スケジューリングポリシーをいかにして自己再構成可能オブジェクトへ与えるかについて説明する．

4.1 スケジューリングポリシーとその抽象度

システム運用者がJuice内部を把握する必要はなく, Juiceにおけるスケジューリングの詳細について考慮させるべきではない．つまり，システム運用者がJuice外部からスケジューリングポリシーを与えるだけで，それを満たすスケジューリングがおこなわれるべきである．スケジューリングポリシーは，(1) ある実行環境において重視する性能，(2) その性能を満たすために採るべきスケジューリング，(3) そのスケジューリングにおいて各メソッドに付与する優先度，の3つの抽象度に分けられる．たとえば，対話型のアプリケーションであれば (1) オブジェクトの負荷が中程度以下の場合には外部への平均的な応答性を重視する，(2) 外部への平均的な応答性を重視するためのスケジューリングとしてSJF(Shortest Job First) を採用する，(3)SJFをおこなうために処理時間の短いメソッドを特定し，それに高優先度を与える，というスケジューリングポリシーが挙げられる．

4.2 スケジューリング対象の分類

優先度別スケジューリングの実現のために，前節の (3) の定義，つまり，外部から実行を要求されるアプリケーションロジックのメソッドへの優先度の付与をおこなわなければならないが，すべての対象メソッド一つひとつに対し，個別の優先度を付与することは現実的ではない．そこで，対象メソッドをその処理内容によっていくつかの種類に分類し，その分類に対して優先度を付与する．以降，この分類のことをメソッドタイプと呼ぶ．たとえば，平均応答時間を短縮するという性能要求は実行時間の短いメソッドを優先的に処理することで満たすことができることが知られているが，そのスケジューリングを実現するために「実行時間が非常に短い」というメソッドタイプや，「CPUを長時間使用する」というメソッドタイプなどが考えられる．ここでは，アプリケーションプログラマがメソッドの分類をおこない，ソースコードレベルで記述することとする．そして，トランスレータがトランスレートする際に，各メソッドとそのメソッドタイプの対を取得する．

4.3 スケジューリングポリシーの記述法

Juiceにおいて，スケジューリングポリシーはシステム運用者が適応ポリシーと呼ぶ外部ファイルに記述することで自己再構成可能オブジェクトへ与える．記述者が望む抽象度での記述を容易におこなうため，前節で挙げた (1) ある実行環境で重視する性能，(2) その性能を満たすために採るべきスケジューリング，(3) そのスケジューリングにおいて各メソッドタイプにどのような優先度を付与するかの3段階に分けて記述することとした．一般に，記述者は最も抽象度の高い (1) のみの記述を望むと考えられるため (2)，(3) は予めアプリケーションの特性に応じた初期ルールを設定する．初期ルールに基づいたスケジューリングをおこなうために，全ての外部から実行を要求されるアプリケーションロジックのメソッドを，ここでは一例として，CPU-Bound，I/O-Bound，処理がすぐに終わるもの，の3つのいずれかの基本メソッドタイプに排他的に分類する．その上で，特に適切な分類が存在するメソッドについては下位メソッドタイプを付与する．適応ポリシーの記述者は必要に応じて上記のタイプを用い (2)，(3) の編集をおこなうことができる．

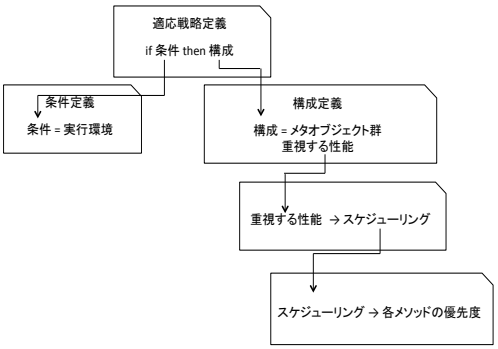

図 3 適応ポリシーの全体像

図3に適応ポリシーの全体像を示す．現在適応ポリシーは適応戦略定義，条件定義，構成定義の3つから構成され，XML[6] を用いて記述している．まず，適応戦略定義には実行環境がある「条件」を満たす場合の適応方法，つまり自己再構成可能オブジェクトの「構成」を定義する．例として，「通信方法が信頼できない」場合には「セキュアな通信をおこなう構成」を採るという定義が挙げられる．次に，条件定義には計算機のおかれている実行環境がどのようなときに適応戦略定義に記述した「条件」を満たすのかを定義する．例として，「通信方法が信頼できない」とは「無線通信をおこなっている」という定義が挙げられる．そして，構成定義には自己再構成可能オブジェクトの「構成」，つまり必要となるメタオブジェクト群の情報を定義する．例として，「セキュアな通信をおこなう構成」には「暗号化通信をおこなう通信制御メタオブジェクトが必要である」という定義が挙げられる．本研究ではスケジューリングポリシーの定義のために，上述した3要素に加え新たに2要素を追加した．以下に，各抽象度ごとの適応ポリシーへの記述法を示す．

(1) ある実行環境において重視する性能

本研究では，構成定義内にその実行環境において重視する性能を記述する項目を追加した．図4に新たな構成定義の例を示し，その要素について説明する．

- `Configuration`
  `Configuration` 要素内に自己再構成可能オブジェクトの構成名と構成とを定義する．
- `Name`
  `Name` 要素は構成の名前を示す．ここでは

```
<Configurations>
   <Configuration>
      <Name>DefaultConfiguration</Name>
      <Component>
         <Tree>
            <Module>Executor1</Module>
            <Tree>
               <Module>Communicator1</Module>
            </Tree>
         </Tree>
      </Component>
      <Performace>MeanResponseTime</Performace>
   </Configuration>
</Configurations>
```

図 4 構成定義

```
<SchedulingDefinition>
   <Solution>
      <Performance>MeanResponseTime</Performance>
      <Name>ShortestJobFirst</Name>
   </Solution>
</SchedulingDefinition>
```

図 5 スケジューリング定義

DefaultConfiguration という名前の構成を定義している．

- Component
  Component 要素はメタオブジェクトの構成を示す．ここでは，Executor1，および Communicator1 からなる構成を定義している．
- Performance
  Performance 要素が本研究で追加した要素であり，ここに重視する性能を記述する．例では MeanResponseTime，つまり平均応答時間を重視すると定義している．

(2) その性能を満たすために採るべきスケジューリング
(2) は現行の適応ポリシーとは別に新たな定義ファイルを用意し，重視する性能とそれを満たすために採用するスケジューリングの名前の対を記述する．図 5 に例を示し，その要素について説明する．

- Solution
  Solution 要素内に重視する性能とそれを満たすために採用するスケジューリングの名前の対として Performance 要素と Name 要素とを記述する．
- Performance
  Performance 要素内には重視する性能を記述する．これは (1) における Performance 要素と対応する．
- Name
  採用するスケジューリングの名前を記述する．ここでは，Shortest Job First の採用を定義している．

(3) そのスケジューリングにおいて各メソッドタイプに付与する優先度

```
<PriorityDefinition>
   <Scheduling>
      <Name>ShortestJobFirst</Name>
      <Priorities>
         <Priority>
            <MethodType>Short</MethodType>
            <Value>1</Value>
         </Priority>
         <Priority>
            <MethodType>I/O-Bound</MethodType>
            <Value>2</Value>
         </Priority>
         <Priority>
            <MethodType>CPU-Bound</MethodType>
            <Value>3</Value>
         </Priority>
      </Priorities>
   </Scheduling>
</PriorityDifinition>
```

図 6 優先度定義

(3) も (2) と同様に現行の適応ポリシーとは別に新たな定義ファイルを用意し，採用するスケジューリングの名前と，そのスケジューリングをおこなうために各メソッドタイプへ付与する優先度を記述する．図 6 に例を示し，その要素について説明する．

- Scheduling
  Scheduling 要素内に採用するスケジューリングの名前とそのスケジューリングにおいて各メソッドタイプに付与する優先度の対として Name 要素と Priorities 要素とを定義する．
- Name
  採用するスケジューリングの名前を記述する．これは (2) における Name 要素と対応する．
- Priority
  Priority 要素に各メソッドタイプの優先度を定義する．例では，Short タイプのメソッド，つまり処理がすぐに終わるメソッドに最も高い優先度である優先度 1 を，I/O-Bound タイプのメソッドに優先度 2 を，CPU-Bound タイプのメソッドに優先度 3 を付与している．

5. 適応的なスケジューリングの実現

本章では，自己再構成可能オブジェクトにおける適応的なスケジューリングの実現方法について説明する．

5.1 優先度別マイクロポート

実行環境と適応ポリシーから各メソッドの優先度が決まると，その優先度に基づいた順序でのメソッド実行の実現が次の課題となる．自己再構成可能オブジェクトにおいて，

高優先度のメソッドを優先的に処理するためには実行制御メタオブジェクトが対象のメソッドを優先的に実行するだけではなく，図 2 に示すように，外部から対象メソッドの実行要求メッセージを受け取ってから返答するまで一貫して優先的にメタオブジェクト間を伝播，処理する必要がある．そのため，実際にはメソッドの実行要求メッセージに対して優先度を付与し，対応する優先度のマイクロポートを介してメッセージ伝播させることで優先度別の処理をおこなう．そして，実行環境の変化に応じてメッセージの優先度を変更することで適応的なスケジューリングを実現する．なお，各メタオブジェクトが優先度別の複数のマイクロポートを保持するのではなく，一つのマイクロポートを保持し，そのマイクロポート内に複数の優先度別キューを持つ実装とした．この理由は，メタオブジェクトやメタオブジェクトプログラマがスケジューリングについて意識する必要がないことと，メタオブジェクト実装の効率化のための 2 点である．

5.2 スケジューリングの流れ

まず，コンパイル時にトランスレータがソースコードレベルで記述された各メソッドとメソッドタイプの対を解釈し，メタオブジェクトマネージャに与える．

Juice の起動時には，メタオブジェクトマネージャが各メソッドとメソッドタイプの対と各メソッドタイプとその優先度の対とを取得する．なお，各メソッドタイプとその優先度の対は自己再構成可能オブジェクトの再構成時にもメタオブジェクトマネージャが適応ポリシーから取得する．

その後，実行時に以下の順序でスケジューリングをおこなう．

(1) 外部からアプリケーションロジックのメソッドの実行要求メッセージが通信制御メタオブジェクトに届く．通信制御メタオブジェクトは，要求されたメソッドの名前から，そのメソッドの優先度をメタオブジェクトマネージャに問い合わせる．
(2) メタオブジェクトマネージャは自身が持つメソッドとメソッドタイプの対応，メソッドタイプと優先度の対応を参照し，問い合わせられたメソッドの優先度を返答する．
(3) 通信制御メタオブジェクトは返答された優先度をメッセージに付与し，実行制御メタオブジェクトのマイクロポートへメッセージを送信する．送信メッセージはマイクロポート内の対応する優先度を持つキューに格納される．
(4) 実行制御メタオブジェクトは自身のマイクロポートからメッセージを受信する．その際，メッセージはマイクロポート内の高優先キュー内のものから順に取り出される．

表 1 実験環境

OS	Mac OS X 10.11
CPU	Intel core i5 2.4GHz
Memory	8GB
Java VM	Apple Java 1.8.0_25

5.3 スケジューリングの入れ替えの流れ

実行環境が変化すると，以下の流れによってスケジューリングが入れ替わり，適応的なスケジューリングが実現できる．

(1) 適応制御メタオブジェクトは適応ポリシーを基にその実行環境において重視する性能を決定する．
(2) 適応制御メタオブジェクトは適応ポリシーを基に重視する性能を満たすために採るべきスケジューリングを決定する．
(3) 適応制御メタオブジェクトは適応ポリシーを基に各メソッドタイプの優先度を決定する．
(4) 適応制御メタオブジェクトがメタオブジェクトマネージャへ各メソッドタイプの優先度を送信し，現在の各メソッドタイプとその優先度の対応を更新する．
(5) 5.2 項の (2) において，メタオブジェクトマネージャが通信制御メタオブジェクトに返答する優先度が変更される．
(6) 5.2 項の (3)，(4) において通信制御メタオブジェクトが各メッセージに書き込む優先度とメッセージが伝播するマイクロポート内の優先度別キューが変化する．

6. 評価

この章では適応的なスケジューリングに対する評価について述べる．

6.1 評価項目

実装した適応的なスケジューリングの評価項目は以下の 2 点である．

(1) 優先度別マイクロポートにおいて，高優先メッセージが優先的に処理されているか
(2) 異なる性能要求に対して，スケジューリングを入れ替えることが有効であるか

6.2 実験

評価項目 (1)，(2) に対してそれぞれ以下の実験 1，2 をおこなった．実験環境を表 1 に示す．

6.2.1 実験 1

実験 1 高優先メッセージと低優先メッセージの平均応答時間の比較

擬似乱数を用いて高優先メッセージと低優先メッセージを 1:1 の割合でランダムに 10000 通発生させ，自己再構成可能オブジェクトに送信した．メッセージの発生率は平均 λ のポアソン分布に従い，サービス時間

表 2 実験 1 の結果

測定回数	応答時間 [ms](高優先)	応答時間 [ms](低優先)
1	13.39	42.68
2	13.48	56.57
3	12.28	48.96
4	11.65	41.34
5	12.22	44.72
平均	12.40	46.86

は平均 μ^{-1} の指数分布に従う．本実験では，$\lambda^{-1} = 6$ ms，$\mu^{-1} = 5$ms とした．なお，高優先メッセージ，低優先メッセージそれぞれでおこなう処理に差異はなく，応答時間は受信側の自己再構成可能オブジェクト内の通信制御メタオブジェクトがメッセージを受け取ってから送信元への返答を送信する直前までの時間とする．

結果 実験 1 の結果を表 2 に示す．高優先メッセージの方が応答時間が短く，優先的に処理されていることが確認できる．

6.2.2 実験 2

本実験のために 3 つのメソッドを用意し，それぞれのメソッドタイプを CPU-Bound，I/O-Bound，Short の 3 タイプとした．なお，メソッドとメソッドタイプの対応は事前にメタオブジェクトマネージャに与えた状態で実験をおこなった．以下にそれぞれのメソッドタイプを持つメソッドについて説明する．なお，すべてのメソッドのサービス時間は平均 μ^{-1} の指数分布に従う．

- CPU-Bound
  メソッドの実行中は CPU を使用する．本実験では $\mu^{-1} = 10$ms とした．
- I/O-Bound
  メソッドの実行中に数回 1ms 程度 CPU を使用するが，他のオブジェクトとの通信や入出力待ちをシミュレートするためにほとんどの時間は Sleep する．このメソッドの実行中に，他の CPU を使用するメソッドを実行することで処理効率の向上が見込まれる．本実験では $\mu^{-1} = 60$ms とした．
- Short
  メソッドの実行中は CPU を使用する．本実験では $\mu^{-1} = 1$ms とした．

異なる性能要求に対して，スケジューリングの入れ替えが有効であるか評価するために，以下の 2 つの性能要求を仮定した実験をおこなった．

実験 2-1 自己再構成可能オブジェクトの平均応答時間を短縮する
高優先から Short，I/O-Bound，CPU-Bound の順に優先度を設定する SJF スケジューリングと単純な FCFS とで，平均応答時間を比較した．

実験 2-2 CPU-Bound タイプを持つメソッドに対する応答時間を短縮する
CPU-Bound にのみ高優先度を設定するスケジューリングと実験 2-1 の SJF とで，CPU-Bound タイプのメソッドの応答時間を比較した．

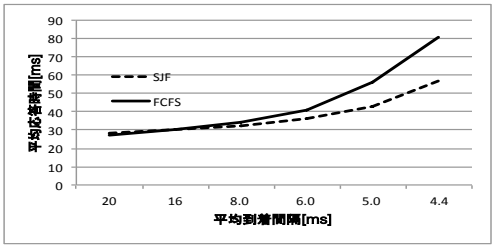

図 7 実験 2-1 の結果

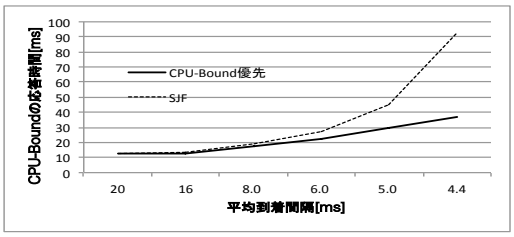

図 8 実験 2-2 の結果

実験内容 擬似乱数を用いて 3 タイプのメッセージを 1:1:1 の割合でランダムに 10000 通発生させ，自己再構成可能オブジェクトに送信した．メッセージの発生率は平均 λ のポアソン分布に従う．平均到着間隔 λ^{-1} を 20ms，16ms，8.0ms，6.0ms，5.0ms，4.4ms と変化させながら計測した．なお，ここにおいても応答時間は，受信側の自己再構成可能オブジェクト内の通信制御メタオブジェクトがメッセージを受け取ってから送信元への返答を送信する直前までの時間とする．

結果 実験 2-1 の結果を図 7 に，実験 2-2 の結果を図 8 に示す．図 7 より，FCFS と比較して SJF の方が負荷の増加に伴う平均応答時間の増加を抑えられていることが確認できる．また，図 8 から CPU-Bound 優先のスケジューリングでは SJF よりも CPU-Bound の応答時間を短縮できていることが確認できる．これらのことから，求められる性能によってスケジューリングを入れ替えることが有効であるといえる．さらに図 7 から，平均到着間隔が非常に大きい場合にはスケジューリングのオーバーヘッドが小さい FCFS の方が SJF よりも平均応答時間が短くなっていることが確認でき，同じ性能を満たしたい場合でもオブジェクト負荷などの実行環境に応じたスケジューリングの入れ替えによって実行環境への適応ができると見込まれる．

7. おわりに

7.1 まとめ

アプリケーションやユーザの立場，時々刻々と変化する実行環境によって異なる性能要求はスケジューリング方

法を適宜入れ替えることで満たすことができる．本論文では，分散システムにおいて頻発する実行環境の変化に自律的に対処する機構として我々が開発している適応型分散オブジェクト指向環境 Juice にこの適応的なスケジューリングを導入した．導入するにあたっての課題として，Juice のオブジェクトモデルである自己再構成可能オブジェクトは内包する複数のメタオブジェクトが互いに通信しあうことで処理が進むため，スケジューリング対象のメソッドの実行順序を優先度に応じて変更するだけでなく，メタオブジェクト間通信においても優先度別の処理をおこなう必要があった．そこで，実際には外部からのメソッドの実行要求メッセージに対して優先度を付与し，メタオブジェクト間通信に優先度別のマイクロポートを用いることでこの課題を解決した．

適応的なスケジューリングをおこなうためのスケジューリングポリシーは適応ポリシーと呼ぶ外部ファイルに記述することで自己再構成可能オブジェクトへ与える．この適応ポリシーには，(1) ある実行環境において重視する性能，(2) その性能を満たすために採るべきスケジューリング，(3) そのスケジューリングにおいて各メソッドタイプに付与する優先度，の 3 つの抽象度に分けて記述することとした．(3) を定義するために，外部から実行を要求されるアプリケーションロジックのメソッドをその処理の内容に応じて分類し，その分類に対し実行環境ごとに優先度を設ける．そして上述したとおり，各メソッドタイプを持つメソッドの実行要求メッセージに優先度を付与し，優先度別のマイクロポートによるメタオブジェクト間通信によって優先度別の処理を実現した．

適応ポリシーには実行環境ごとにメソッドタイプの優先度を定義しているため，実行環境が変化するとそれに伴いメソッドタイプとその優先度の対応も変化する．そしてその対応を基に，各メッセージに付与する優先度を変更することで動的にスケジューリングを入れ替える適応的なスケジューリングを実現した．

異なる性能要求に対して，スケジューリングを入れ替えることが有効であるかという評価項目に対して，求められる性能として自己再構成可能オブジェクトの平均応答時間と特定のメソッドの応答時間の 2 つを仮定し，FCFS，SJF，特定のメソッドを優先するスケジューリングの 3 つのスケジューリング間で比較した．その結果，平均応答時間に対しては SJF が，特定のメソッドの応答時間に対しては特定のメソッドを優先するスケジューリングがそれぞれ最も優れており，異なる性能要求に対してスケジューリングを変更することの有効性が確認できた．また，同じ性能を満たしたい場合でも，実行環境に応じたスケジューリングの入れ替えが有効であることがわかった．

7.2 今後の課題

メタレベルスケジューリングとして，メタオブジェクト間のスケジューリングをおこなうことができる．例として，複数のメタオブジェクトにおいて同一優先度のマイクロポートにメッセージが格納されている場合，どのメタオブジェクトを優先的に実行するかというメタレベルスケジューリングが挙げられる．従来の実装では各メタオブジェクトそれぞれがスレッドを保持しており，自己再構成可能オブジェクトにおけるスレッドスケジューリングは OS に頼らざるを得なかった．そこで，山本，布志原らの先行研究 [4], [5] によってメタオブジェクト間の並行実行にコルーチン [7], [8] が導入され，メタメタレベルによる明示的なスケジューリングが可能となった．現在はどのようなメタオブジェクト間のスケジューリングが効果的であるか調査を進めている．

また，実行制御メタオブジェクトによるスケジューリングも今後の課題として挙げられる．たとえば，並行実行制御をおこなう実行制御メタオブジェクトにおいて，サスペンドロックの解除の順序のスケジューリングや，ラウンドロビンスケジューリングによるアプリケーションロジックの実行などがここで取り扱う問題となる．

参考文献

[1] 小田謙太郎，適応型分散オブジェクト指向計算モデルとその応用に関する研究，九州工業大学博士学位論文，2008.
[2] Kentaro ODA,Shinobu IZUMI,Yoshihiro YASUTAKE,Takaichi YOSHIDA, A Simple Reconfigurable Object Model for a Ubiquitous Computing Environment, International Journal of Computer Science and Network Security(IJCSNS),Vol.7,No.5, pp8-16,2007.
[3] 加藤建士，小田健太郎，吉田隆一，適応型分散オブジェクト指向計算環境の実現，情報処理学会論文誌 44 巻 1 号 pp.39 47，2003
[4] Sho Yamamoto, Mahdillah, H.Saito and T.Yoshida，Introduction of Coroutines for the Self-Reconfigurable Objects，2013 International Workshop on ICT，2013.
[5] 布志原雅之，Mahdilha, 江本健斗，吉田隆一，自己再構成可能オブジェクトのためのタスクスケジューリングに関する研究，情報処理学会九州支部若手の会セミナー 2014, pp.45-49, (2014).
[6] Extensible Markup Language (XML) 1.0 (Fifth Edition): https://www.w3.org/TR/REC-xml/, (2017 年 1 月 18 日アクセス)
[7] Lukas Stadler, Coroutine in Java, Johannes Kepler University Linz, 2011.
[8] Apache Project Javaflow: http://commons.apache.org/sandbox/commons-javaflow/, (2017 年 1 月 23 日アクセス)
[9] Kento Aida, Effect of Job Size Characteristics on Job Scheduling Performance, IPDPS '00/JSSPP '00 Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing, pp.1-17, 2000.
[10] HSIU-JY HO, WEI-MING LIN, Task Scheduling for Multiprocessor Systems with Autonomous Performance-Optimizing Control, Journal of Information Science and Engineering 26, pp.347-361, 2010.