システムソフトウェアと
オペレーティング・システム 81－21
（１９９８．５．７）

動的分散ソフトウェアツールキット “Meta Space” の設計と実装

藤村 浩士[1] 中澤 仁[1] 大越 匡[1] 徳田 英幸[1 2]
[1] 慶應義塾大学大学院 政策・メディア研究科 [2] 慶應義塾大学 環境情報学部

本研究は，従来の分散ソフトウェアモデルの問題点を解決する新たなモデルとして，動的分散ソフトウェアモデルを提案し，同モデルを実現するためのソフトウェアツールキットを提供する．本研究で提案する動的分散ソフトウェアモデルは，プログラムのモジュールが計算機の制約を受けず，自律的に計算機間を移動する．これにより，動的負荷分散の実現，耐故障性の向上，大規模分散性の向上，計算環境の最適化，小型携帯端末の処理支援などを実現できる．本モデルを実現するために，動的分散ソフトウェアの構築を支援するツールキットであるMeta Spaceを設計，実装，評価した．

Design and Implementation of Dynamic Distributed Software Toolkit “Meta Space”

Hiroshi Fujimura[1] Jin Nakazawa[1] Tadashi Okoshi[1] Hideyuki Tokuda[1 2]

[1]Graduate School of Media and Governance, Keio University
[2]Faculty of Environmental Information, Keio University

In this research, we propose a software model for the dynamic distribution of software components across a wide-area network, and we provide a software environment to realize the proposed model. In the dynamic distributed software model, software modules can autonomously migrate among computers. This feature allows dynamic distributed software to have the advantage of dynamic load balancing, fault tolerance, scalability, optimization of computing environment, and the processing support for small-size portable devices. Moreover, we designed and implemented a dynamic distributed software toolkit named ”Meta Space”.

1 はじめに

分散ソフトウェアは，異なる計算機に常駐するサブシステム間の協調動作によって，全体の機能を提供するソフトウェアである．クライアント・サーバモデルに代表される従来の分散ソフトウェアモデルは，サブシステムが計算機に常駐し，計算機間を移動しない．このような分散ソフトウェアモデルには，いくつか問題点が存在する．第一の問題点は，耐故障性の低さである．ネットワークの障害や計算機自体の障害などにより，一部のサブシステムが利用不能になると，分散ソフトウェア全体の機能が損なわれる．第二の問題点は，実行ファイルの保守が困難な点である．システムの起動に先立って，システム管理者が実行ファイルを対象の計算機に配置せねばならない．

本論文では，これらの問題点を解決する新たな分散ソフトウェアモデルとして動的分散ソフトウェアモデルを提案し，同モデルを実現するためのツールキットである “Meta Space” について述べる．

2 動的分散ソフトウェアモデル

動的分散ソフトウェアモデルとは，プログラム中の各モジュールが計算機に常駐せず，最適な実行環境を求めて自律的に計算機間を移動するモデルである．したがってプログラムは，必要に応じて各モジュールに分割され，そのモジュールにとって最適な計算機上で実行される．動的分散ソフトウェアモデルの利点を以下に示す．また，動的分散ソフトウェアモデルの概念図を図1に示す．

動的負荷分散の実現

計算資源を必要とするモジュールをプログラムから分離して，異なる計算機上で実行させることにより，動的に負荷分散を実現できる．更に，分離したモジュール（以下分散モジュールと呼ぶ．）は実行中であっても計算機間を自由に移動できるため，最適な実行環境の計算機を求めて移動を繰り返すことができる．

耐故障性の向上

ネットワークの断絶や，計算機自体の障害により，接続先のモジュールが利用できない場合，新たなモジュールを異なる計算機に移送して処理を継続できる．これにより，モジュールの移送先計算機が存在する限り，システムの運用を継続できる．

大規模分散性の向上

モジュールの実行に必要なプログラムを動的に移送することで，モジュールが分散する計算機の台数が増加しても，システム管理者の負担は増大しない．したがって，インターネット上の不特定多数の計算機で分散並列実行されるソフトウェアを作成しても，少人数で運用できる．

計算環境の最適化

マルチメディア処理モジュール，浮動小数点演算モジュールなどの特色を持った処理を，それぞれの処理をより高速に実行可能なCPUへ移動させ，高速に処理できる．

小型携帯端末の処理支援

各モジュールを異なる計算機で分散実行することで，移送元のプログラムのメモリ消費量を抑制できる．これにより，実メモリの少ない計算機でも高度なプログラムを実行できる．

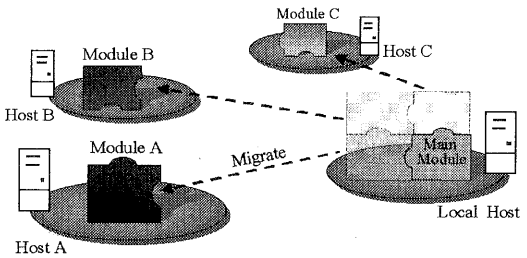

図 1: 動的分散ソフトウェアモデルの概念図

3 動的分散ソフトウェアツールキットの設計

本節では，動的分散ソフトウェアツールキットであるMeta Spaceの設計について述べる．

3.1 Meta Spaceの概要

Meta Spaceは，動的分散ソフトウェアの構築を支援するためのライブラリ及び，分散モジュールの実行環境から成るソフトウェア群である．Meta Spaceを用いることにより，プログラマは高機能な動的分散ソフトウェアを効率的に開発できる．

Meta Spaceは，インターネット上の多数の計算機を共有計算資源として，提供，管理，分配する機構を提供している．提供された計算資源（計算機）は分散モジュールの実行環境として利用可能なため，大規模分散実行を必要とする分散ソフトウェアを構築できる．計算資源の提供は，計算機の所有者が共有計算資源として，計算資源管理サーバであるMPoolへ登録することで行なわれる．MPoolは登録された計算資源を管理し，計算資源要求に応じてアプリケーションへ分配する．提供された共有計算資源は，Meta Spaceが提供するライブラリを用いて実装された分散モジュールによって利用される．アプリケーションが実行されると，MPoolから分散モジュールに対して共有計算資源が割り当てられる．アプリケーションは，割り当てられた共有計算資源へ分散モジュールの実行に必要なプログラムを移送する．移送完了後，分散モジュールはアプリケーションの要求に応じて，移送先の計算機で実行される．

3.2 サブシステムの構成

Meta Spaceは，MPool部，MSupporter部，MClient部，MModule部，Ticket部の5つのサブシステムで構成される．

MPool部

計算資源を集約するサーバプログラム．MPoolに蓄積された資源は，必要に応じてMClient（後述）に分配される．本システムでは，資源を提供する計算機をサポートホストと呼ぶ．

MSupporter部

サポートホスト上に常駐し，自らの計算機を共有計算資源としてMPoolへ登録するサーバプログラム．登録はサポートホストの所有者が決定する計算資源提供ポリシに応じて行なわれる．

MClient部

MPoolに蓄積された計算資源を利用するために，アプリケーション内部でインスタンス化されるクラス．MClientはMPoolに対して計算資源要求を行ない，分配された計算資源(サポートホスト)に対してMModule(後述)オブジェクトを送り込む．

MModule部

Meta Spaceを利用するアプリケーションプログラマによって拡張され，分散処理する内容が実装されるクラス．MModuleはMClientによってサポートホスト上に転送され，アプリケーションからのリモートメソッド呼び出しによって制御される．

Ticket部

MSupporter内で発行され，サポートホストの情報(ホスト名，CPUタイプ，メモリ容量など)および，提供条件(最大提供時間，セキュリティ制限など)が格納されるクラス．MPoolへの登録時にMSupporterから渡される．Ticketは複数発行でき，それぞれに対して別々の提供条件を記述可能である．

3.3 動作手順

本システムの動作手順は「サポートホストの登録と計算資源要求のフェーズ」,「計算資源共有利用のフェーズ」の2つに分けられる.

サポートホストの登録と計算資源要求のフェーズ

はじめに,サポートホストの登録と計算資源の要求の手順を図2に示す.

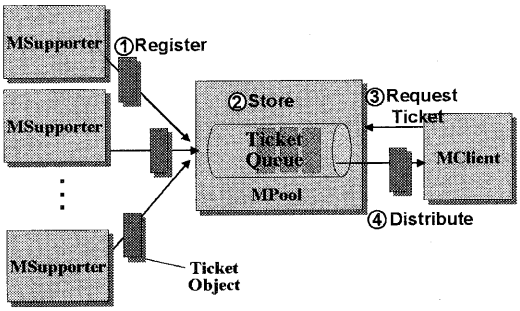

図 2: サポートホストの登録と計算資源要求の手順

1: MSupporterは自らの計算機が計算資源として提供可能であることを示すTicketオブジェクトを作成して,あらかじめ指定されたMPoolへTicket オブジェクトを発行する.

2: MPoolはTicketオブジェクトを受け取り,MPool内部のキューに追加して登録を完了する.

3: 新たな計算資源を必要とするMClientはMPoolに対して,Ticket オブジェクトを要求する.

4: MPoolは,MClientへTicketオブジェクトを分配する.

これ以降はMPoolを介さずに,MClientとMSupporterで直接通信を行なう.MClientはTicketオブジェクトから取得した情報を元にMSupporterに接続し,MModuleを送信する.

計算資源共有利用のフェーズ

次に,計算資源共有利用の大まかな手順を図3に示す.

1: MModuleオブジェクトはMeta Spaceを利用するアプリケーション内部でインスタンス化される.

2: MModuleオブジェクトはMClient内の移送待機キューに追加され,MSupporterへ移送されるまで待機する.

3: MClientは移送待機キュー内にMModuleオブジェクトが存在すれば,先に述べたサポートホストの登録と計算資源要求の手順3にしたがって,MSupporterに接続し,分散モジュールを移送する.

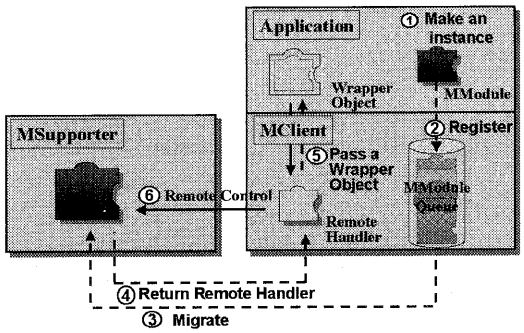

図 3: 計算資源共有利用の手順

4: MSupporterはMModuleオブジェクトを受け入れると,MModuleオブジェクトを遠隔から操作できるリモート参照をMClientに返す.

5: MClientは,リモート参照を内部に保持したラッパオブジェクトを生成してユーザアプリケーションに送る.

6: ユーザアプリケーションがラッパオブジェクトのメソッドを呼び出すと,ラッパオブジェクトの内部で対応するリモート参照のリモートメソッドが呼ばれ,サポートホスト上のMModuleオブジェクトが操作される.

例外発生時

MModuleオブジェクトの実行中に,サポートホストやネットワークの障害が発生する,あるいはMSupporterから退去命令が出されるなどの例外が発生する可能性がある.例外が発生した場合は,後述するチェックポインティング機構によってバックアップされたMModuleが,移送待機キューへ再登録される.その後,MModuleは手順3,4にしたがって他のMSupporterへ移送され,計算を再開する.

4 動的分散ソフトウェアツールキットの実装

本節では,動的分散ソフトウェアツールキットであるMeta Spaceの実装について述べる.

4.1 実装環境

Meta Spaceの実装はJava言語[1]を用いて行なった.実装言語としてJavaを選択した理由を以下に示す.

安全性

JavaはJava Virtual Machine (JVM)[2]と呼ばれるインタプリタによって実行される.JVMには,Javaプログラムを安全に実行するための様々な機構が導入されている.これらの機構を利用することにより,外部から侵入を試みる有害な分散モジュールか

らサポートホストを防衛できる．

マルチプラットフォーム性

JavaプログラムはJVMが動作する計算機であれば，プログラムを再コンパイルすることなく実行できる．小型携帯端末の計算処理支援を研究の成果として掲げている本研究にとって，この特性は有益である．この特性により，本システムを用いて作成された小型携帯端末用ソフトウェアの分散モジュールは，高性能計算機上へ移動して，豊富な計算資源を利用できる．

ネットワーク指向型

Javaのオブジェクトは，ネットワークを通じて遠隔計算機上へ移送できる．また，Java RMI[3] を利用することでリモートオブジェクトに対して透過的にアクセスできる．これらの特性および機構を利用することで，計算機間を移動する分散モジュールを容易に実装できる．

4.2 分散モジュールの動的転送

MSupporterは，受け入れた分散モジュールの実行に必要なクラスを，MClient やWebサーバからネットワーク経由で動的にロード可能である．これにより，分散モジュールの実行に必要なクラスをサポートホスト上のファイルシステムに配置するというシステム管理者の作業を省略できる．また，分散モジュールが分散する計算機の台数が増加した場合も，システム管理者の負担は増大しないため，大規模分散実行されるソフトウェアを少人数で運用可能である．

4.3 分散モジュールの位置透過的制御

分散モジュールが計算機間を移動した場合，分散モジュールのラッパオブジェクトはリモート参照を自動的に変更する．これにより，アプリケーションは分散モジュールに対して継続的にアクセスできる．

この機能はラッパクラスによって実現される．ラッパクラスは本システムが提供するラッパジェネレータコマンドによって生成できる．アプリケーションがラッパオブジェクトのメソッドを呼び出すと，内部でリモート参照の対応するリモートメソッドが呼び出される．分散モジュールが異なる計算機へ移動すると，ラッパオブジェクト内部のリモート参照が自動的に切り替わるため，アプリケーションから位置透過的に分散モジュールを操作できる．

4.4 分散モジュールのガーベッジコレクタ

MSupporterは，不要になったラッパオブジェクトに対応する分散モジュールを終了できる．これにより，MSupporterは計算資源を新たな分散モジュールへ提供可能となる．本システムでは，分散モジュールを終了する方法として，プログラマが分散モジュールの終了手続きを明示的に呼び出す方法と，システムがラッパオブジェクトへの参照が消えるタイミングを検知することで，分散モジュールを自動的に終了する方法を提供している．

4.5 分散モジュールの制御インタフェース

本システムでは，分散モジュールの制御インターフェースとしてコールバックメソッドと，リモートメソッドを提供している．

コールバックメソッドは，分散モジュールの移送直後，強制退去直前，終了直前などにMSupporterによって呼ばれるメソッドである．プログラマは，サブクラスでこれらのメソッドの内部処理を実装する．コールバックメソッドの利点は，メソッドの内容を実装するだけで良いためプログラマの負担が少ないこと，移送後MSupporterから直ちにメソッドが呼ばれるため効率的であること，メソッド呼び出し中はネットワークが断絶していても支障がないことなどがあげられる．しかし，プログラマが分散モジュールを操作するメソッドを独自に定義できない，分散モジュールをインタラクティブに操作できないなどの問題点がある．

一方，リモートメソッドは，プログラマが独自に定義可能であり，アプリケーションから必要に応じて呼び出すことができる．しかし，リモートメソッド呼び出しと実際のメソッド呼び出しの間でネットワークによる遅延が発生すること，ネットワークが断絶すると分散モジュールを操作できないなどの問題点がある．

両者の方法はそれぞれの欠点をうまく補完し合うため，プログラマは分散モジュールの性質に応じて，分散モジュールの操作方法を使い分けることができる．

4.6 分散モジュールの自動復旧機能

計算機の異常など何らかの理由により，分散モジュールが実行途中で中止された場合，分散モジュールを異なる計算機に移送し，中止された地点から計算を再開できる．

本システムでは，プログラマが必要に応じて，プログラム内に明示的にチェックポインティングメソッドを挿入する方法を採用している．実行スレッドが計算ルーチン中に挿入されたこのメソッドに遭遇すると，それまでの実行結果を含んだ分散モジュールのオブジェクトをラッパクラス内へ待避する．この方法の利点は，プログラマの計画に沿ってチェックポインティングが行なえることである．しかし，プログラマはこの分散モジュールの実行が再開した際に，計算ルーチン中の既に済んでいる部分をスキップするように，アルゴリズムを工夫しておかねばならないという問題点がある．

4.7 分散モジュールの強制退去機能

MSupporterは，現在受け入れている分散モジュールに対して強制退去命令を発行できる．強制退去命令が発行されるのは，設定されていた提供時間を越えた場合，分散モジュールが不正な挙動をした場合などである．加えて，提供者の都合による明示的な強制退去命令の発行も可能である．これは，提供者が計算機を利用したい際に，ただちに分散モジュールが計算資源を明け渡すことを可能にするためである．

4.8 動的分散ソフトウェア構築の実際

分散モジュールの実装はリモートメソッドの定義，リモートメソッドの実装，ラッパジェネレータによるラッパクラスの生成の3ステップを必要とする．それぞれJava RMIのリモートメソッド定義，リモートメソッド実装，rmicコマンドによるスタブ及びスケルトンクラスの生成に対応しているため，Java RMIの利用経験があるユーザなら，分散モジュールの実装も比較的容易に行なえる．

5 動的分散ソフトウェアツールキットの評価

本節では，動的分散ソフトウェアツールキットの定量的及び，定性的評価について述べる．

5.1 定量的評価

今回は，Meta Spaceを用いて構築された並列計算型アプリケーションのパフォーマンスについて定量的評価を行なった．本測定により，並列計算型アプリケーションはMeta Spaceを用いることでパフォーマンスが向上することを実証する．また同時に，Meta Spaceを用いた並列計算型アプリケーションのパフォーマンスは，MPoolへ追加される計算資源の量に比例することも実証する．

5.2 測定方法

測定用アプリケーションには素数探索を行なうアプリケーションを利用した．素数探索を行なう分散モジュールを64個作成し，各分散モジュールに0から50000000の範囲に存在する素数を探索させた．MPoolへ追加する計算機台数を1台，2台，4台，8台，16台，32台，64台と増加させて，それぞれの場合で探索開始から全ての分散モジュールの素数探索が完了するまでの時間を計測した．

5.3 測定環境

測定には100Base-Tで相互接続された，Sun Microsystems社のワークステーションであるUltra 30を64台使用した．図4に測定環境を示す．また表1にUltra 30の性能を示す．

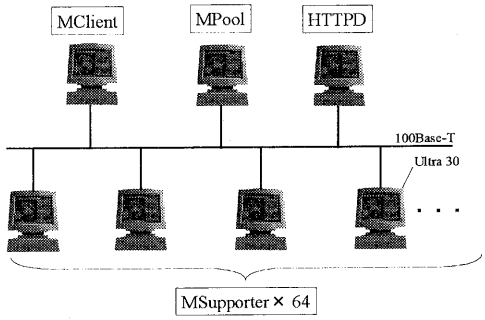

図 4: 測定環境

表 1: 測定環境

CPU	UltraSPARC-II 248MHz × 1
メモリ	128MB
OS	Solaris2.6

5.4 測定結果と考察

図5にMeta Spaceを用いたアプリケーションの利用計算機台数における速度比のグラフを示す．これより利用計算機台数に対して速度比が直線的に向上することがわかる．

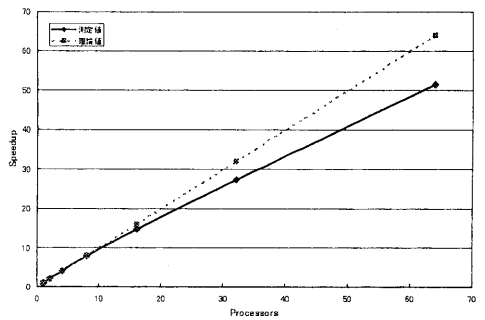

図 5: 測定結果のグラフ

図5には測定値の他に，通信遅延や計算機間の計算速度のばらつきが全くなかった場合の値を，傾き1の理論値としてプロットしている．これより測定値がMPoolへ追加する計算機の台数が多くなるにつれて，理論値からはずれることがわかる．この原因として，いくつか考えられる．

ネットワーククラスローディングのオーバヘッド

MSupporterは受け入れた分散モジュールの実行に必要なクラスをネットワーク経由でロードする．ただし，ネットワーク経由でのロードは初回だけで，2回目以降はキャッシング機構によってクラスファイルを再利用する．したがって分散率(x個の

表 2: 関連システムとの機能比較

	PVM	Javelin	Ninflet	MetaSpace
分散モジュールの動的移送	×	○	○	○
分散モジュールの自動復旧	×	×	○	○
ユーザによる遠隔手続きの定義	×	×	×	○
分散モジュールの位置透過的な記述	×	×	○	○
プラットフォーム非依存性	×	○	○	○

MModuleをn台の計算機に分散させた場合の分散率を$\frac{n}{x}$とする．)の増加に伴って，全実行時間に占めるクラスローディングオーバヘッドの相対的割合が増加する．

5.5 ネットワークの輻輳

x個のMModuleを分散率dで分散させた場合，xd回のクラスファイル要求がWebサーバに対して同時発生する．したがって分散率が増加するほどWebサーバの負荷とネットワークの輻輳が発生するため，ロード時間が大きくなる傾向にある．

6 関連システムとの機能比較

分散モジュールの応用例として，複数の計算機を用いた大規模分散計算が挙げられる．本節では，大規模分散計算を目的とする関連システムとしてPVM, Javelin, Ninfletを取り上げ，Meta Spaceと機能比較を行なった．評価項目および評価結果を表2に示す．

PVM (Parallel Virtual Machine)[4]は，ネットワークに接続された異機種UNIXコンピュータ群を，並列コンピュータとして利用できるライブラリ及び，コマンドである．PVMはモジュールの動的移送機能を提供していないため，大規模分散計算には適さない．また，マルチプラットフォーム性を実現していないため，計算機のアーキテクチャごとに実行ファイルをメンテナンスするという煩わしさがある．

これに対し，Javaで記述されマルチプラットフォーム性を実現したシステムとしてJavelin[5]やNinflet[6]などがあげられる．

本システムのMPoolにあたるBrokerへの登録のために，Broker Appletのあるwebページを開くという方法を採用している．この方法は手軽に登録できるという利点はあるが，計算資源を提供している間は常時ブラウザを開いておかねばならないため常連ユーザを得られにくいという欠点を持つ．

Ninfletは，本システムと類似した構成を持つ．Ninfletは，本システムのMPool, MSupporter, MClient, MModuleがNinfletのDispatcher, Server, Client, Ninfletにそれぞれ対応する．分散するNinfletの記述はWorkerクラスを拡張して，Workerのあらかじめ定められたコールバックメソッドをオーバライドする．この方法の利点は定められたメソッドの内容を実装するだけで良いためプログラマの負担が少ないことがあげられる．しかし，ユーザがWorkerの制御インタフェースを独自に定義できない，Workerを必要に応じてインタラクティブに操作できないなどの問題点がある．

7 結論

本論文では，従来の分散ソフトウェアモデルの問題点を解決する新たな分散ソフトウェアモデルとして動的分散ソフトウェアモデルを提案し，同モデルを実現するためのツールキットである“Meta Space”の設計，実装，評価について述べた．今後は，より詳細な測定を行ない動的分散ソフトウェアモデルの有効性を検証すると共に，分散モジュールの自立性向上及び，セキュリティの向上を目指す．

参考文献

[1] James Gosling, Bill Joy, Guy Steele, “The JAVA Language Overview”, Sun Microsystems Inc. (1998).

[2] Tim Lindholm, Frank Yellin, “The Java Virtual Machine Specificati on”, Sun Microsystems Inc. (1996).

[3] Sun Microsystems Inc., “JAVA REMOTE METHOD INVOCATION”, http://www.java.sun.com/marketing/collateral/rmi_ds.html (1998).

[4] Al Geis, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, Vaidy Sunderam, “PVM 3 USER'S GUIDE AND REFERENCE MANUAL”, (1993).

[5] Bernd O.Christiansen, Peter Cappello, Mihai F.Ionescu, Michael O.Neary, Klaus E.Schauser, and Daniel Wu, “Internet-Based Parallel Computing Using Java”, ACM Workshop on Java for Science and Engineering Computation, (1997).

[6] 高木，松岡，中田，関口，佐藤，長嶋，“Ninflet:Javaによる World-Wide High Performance Computing環境”，Internet Conference'97.