12-IS02

クラウド基盤ミドルウェアのスケーラビリティ向上に関する研究

杉木 章義（筑波大学）

概要

本研究では，学際大規模情報基盤共同利用環境を想定したクラウド基盤ミドルウェアのスケーラビリティ向上を目指す．本研究は既存のミドルウェアの一部の改良に留まらず，独自のミドルウェアを開発し，ミドルウェアのアーキテクチャ設計からの機能改善を目指す．また，本研究のスケーラビリティでは，単なる性能向上を目標にするだけでなく，学際的な多様な要望への対応から，管理の容易性を維持したまま，総合的な拡張性を実現することを目標とする．

1. 研究の目的と意義

クラウドコンピューティングは，近年，学術分野および産業分野に急速に普及している．学術分野においては，従来の HPC（High Performance Computing）の分野で研究されてきたグリッドコンピューティングをより発展させ，大規模数値計算によるシミュレーションから，ゲノム情報やストリームデータなどの大規模データ処理，仮想化技術に基づくサーバ利用など，学際的な環境を利用した多様な利用形態へとその応用が広がっている．よって，多拠点に設置された大規模情報基盤の結びつきをより高め，学際的な共同研究を推進していくためには，ミドルウェア自身の研究や開発も進めていく必要がある．

本研究グループでは，クラウド基盤ミドルウェア「Kumoi」を独自に研究開発している．本システムは Operating System（OS）の設計に基づき，ミドルウェアの中核を出来る限りコンパクトにし，その他の多くの機能を拡張部分としてスクリプトとして実現することで，柔軟性とカスタマイズ性を高めている．この設計手法は OS のマイクロカーネルの目標と共通しており，学際的な環境における多様な要望にも対応しやすいと考えている．また，本ミドルウェアはクラウド基盤ミドルウェアを開発するためのメタミドルウェアとしても見ることができ，ミドルウェア内部のさまざまなアルゴリズムを試行するための実験用プラットフォームとしても利用することができる．

本研究課題では，学際大規模情報基盤共同利用環境に対応するためのクラウド基盤ミドルウェアのスケーラビリティを高める研究開発を実施した．

具体的には，本研究課題では，研究のステップを二段階に分けて実施することを目標とした．最初の段階として，ミドルウェア自身の改良を進めた．Virtual Machine（VM）や物理計算機などの資源情報のキャッシュ機構を実装し，実用上の応答性を高めるとともに，同時に利用対象者を広げるためのセキュリティ機能の導入を行った．また，耐障害機能の導入や，仮想ネットワークやアプリケーションの支援機能を導入することで，少数の管理者が扱える計算機の規模を拡大させた．

また，より長期的な段階として，数理的検証を行った分散アルゴリズムをスクリプトとしてミドルウェア上で動作させることを目標にした．本ミドルウェアは分散アルゴリズムをミドルウェア上にスクリプトとして記述する設計となっているため，短いサイクルで試行錯誤を繰り返すことが可能である．この方針は，OS・システムソフトウェアの研究者と，数理的検証の研究者の学際的な協力により，クラウドにおけるさまざまな問題を解決することを意図している．設計段階から，モデル検査法を中心とする数理的検証技術を適用し，数理的検証技術が学際大規模情報基盤共同利用環境での利用にも耐えうる信頼性の高いミドルウェア構築に役立つことを示すことを目標にした．

本研究は，独自の問題意識に基づき，クラウド基盤ミドルウェアのアーキテクチャの再設計を目指している点に特徴がある．現在，クラウド基盤

ミドルウェアとして，さまざまなものが提供されおり，実際の学際大規模情報基盤環境でも，これらのミドルウェアが広く利用されている．商用では VMware vSphere，Citrix XenServer などがあり，オープンソースでは，Nimbus，Eucalyptus，OpenNebula，Xen Cloud Platform，CloudStack，OpenStack などが知られている．これらのミドルウェアはよく設計されているが，今後のメニィ・コアを始めとする計算環境の変化や，利用形態の変化などを考えると，クラウド基盤ミドルウェアに，より一層の柔軟性やカスタマイズ性が求められていると考えられる． 本研究課題は，その領域に焦点を当て，研究開発を実施した．

2. 当拠点公募型共同研究として実施した意義

(1) 共同研究を実施した大学名

東京大学

(2) 共同研究分野

超大規模情報システム関連研究分野

(3) 当公募型共同研究ならではという事項など

本研究は先行研究の成果を活用しているが，学際大規模情報基盤共同利用環境での運用を想定し，改良を進めた点が異なる．従来は Non-technical Computing のクラウドを対象にしており，上記の環境での利用には問題があった．HPC を含めた Technical Computing にも対応できるクラウド基盤ミドルウェアを目標として，過去のグリッドコンピューティングの成果も意識しながら，改良を進めた点が異なる．

3. 研究成果の詳細と当初計画の達成状況

(1) 研究成果の詳細について

クラウド基盤ミドルウェアのスケーラビリティ向上に関して，下記の 5 つの研究開発を実施した．

(A) キャッシュ機能の導入

クラウド基盤ミドルウェアに統一的なキャッシュ機構を導入し，性能向上を実現した．本機構は，従来の分散オブジェクトのキャッシュ技術の自然な拡張で実現されており，また VM や物理計算機など，すべての資源でキャッシュが有効化されている．

(B) セキュリティ機能との連携

学際大規模情報基盤共同利用環境では，多数の研究者が共同で利用するため，認証・認可・課金・監査・暗号化などのセキュリティ機能を実現することが必須である．これらの機能を整備するとともに，上記のキャッシュ機構とも連携するよう工夫を行った．

(C) 透過的な障害対策機能の導入

上記の大規模環境では，規模の大きさから障害は絶えずどこかで発生していると考えるのが自然である．従って，これらの障害にうまく対処できなければ，実用上，利用可能なシステムの規模を低下させてしまう．本研究では，未使用となっている冗長資源を活用することで透過的な障害対応を実現した．

(D) 仮想ネットワークへの対応

学際的な利用環境では，機密性やプライバシーへの配慮から，利用者同士のネットワークも仮想的に分離することが望ましい．本研究課題では，OpenFlow と Open vSwitch を基礎とした仮想ネットワークへの対応も行った．

(E) VM 内で稼働するアプリケーションへの対応

クラウド基盤ミドルウェアが負荷に応じて VM のインスタンス数を自動的に増減させても，VM 内で稼働するアプリケーションが対応しなければ，適切なサービスを提供することは難しい．本研究課題では，VM 内で稼働するアプリケーションも統一的に管理するための機能を追加した．

以降，3.1 節でクラウド基盤ソフトウェア Kumoi の概要について説明し，次に 3.2 節以降で個別の機能について詳しく述べる．

3.1 クラウド基盤ソフトウェア Kumoi

3.1.1 概要

Kumoi の概要を**図 1** に示す．Kumoi では，クラウドに要求される俊敏性への対応から，高度なカスタマイズ性と拡張性を目標としており，中核となる機能を最小化し，その他の多くの機能を周辺機

能としてスクリプトで実現するというアプローチを取っている．

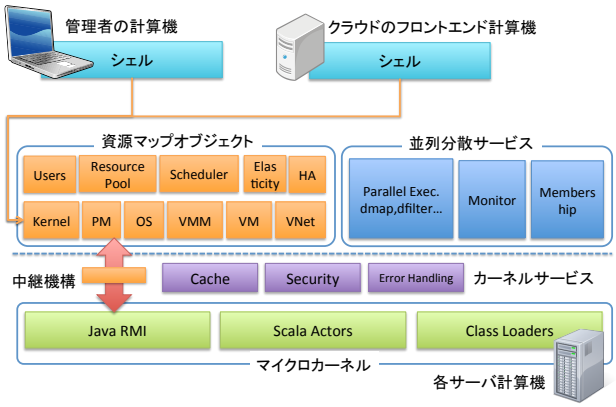

図 1：Kumoi の概要

その目標のため，ミドルウェアの機能をまず，カーネルとシェルに分割し，カーネルはデータセンター内の各計算機で動作させ，シェルは管理者の計算機やクラウドのフロントエンド計算機で動作するように設計してある．

カーネルでは，Java Remote Method Invocation (RMI) と Scala の Actor を組み合わせた通信機構のみを**マイクロカーネル**として最下層に配置している．その上で，物理計算機や VM などの各資源を分散オブジェクトとして抽象化し，統一的なインターフェースを提供している．Kumoi では，これらのオブジェクトを**資源マップオブジェクト**と呼んでいる．一方で，スケーラビリティを向上させるために， `dmap()`や `dfilter()`などの並列スケルトンや，メンバシップ機能などの**並列分散サービス**を提供している．

クラウドの管理者や開発者は，これらの 2 つの機能を組み合わせ，実現したい作業をスクリプトとして**シェル環境**で記述していく．本記述は Scala をベースとしている．

図 2 は具体的なスクリプト記述例である．`pms` は現在，システムに参加している物理計算機に対応する資源マップオブジェクトのリストである．一方で，`vms` はそれぞれの計算機で稼働している VM に対応する資源マップオブジェクトのリストである．本スクリプトでは，並列版の `map` 関数である `dmap()`を使用し，最後に `flatten` で現在稼働している VM の一覧のリストとしている．

```
kumoi> pms.dmap(_.vms).flatten
```

図 2：スクリプトの記述例

3.1.2 クラウド環境の構築

Kumoi を使用してクラウド環境を構築した場合の例を**図 3** に示す．前節で述べたように，Kumoi のカーネルはサーバ資源プールの各計算機で動作する．一方で，シェル環境はクラウドのフロントエンドとなる計算機で動作させ，また必要に応じて管理者の計算機で起動することもできる．スクリプティングによって，さまざまな利用者インターフェースを柔軟に構築し，クラウドの利用者に提供することができる．なお，シェル環境はあくまで利用者インターフェースの構築に利用することを想定しており，クラウドの利用者が直接，シェル環境を操作することは想定していない．

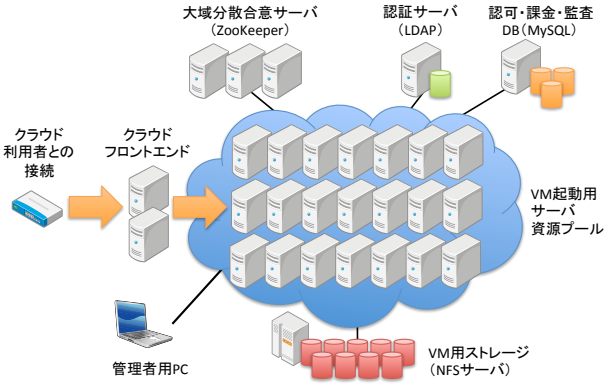

図 3：Kumoi を利用したクラウド環境の構築

Kumoi では，この他にもさまざまな計算機を利用して，クラウド環境を構築する．まず，資源プールなどの大域的な機能の高可用性を実現するために，ZooKeeper を利用する．次に，LDAP サーバを通じて利用者認証を行い，その認証情報をもとに，さまざまな資源へのアクセス認可を行うデータベースを MySQL で構築している．また，クラウド環境の提供には欠かせない課金情報の記録や監査ログの記録なども MySQL で行っている．最後に，VM のライブマイグレーションを実現するために，

VM 用のストレージも利用しており，現在は NFS サーバで実現している．

3.2 キャッシュ機構の導入

3.2.1 概要

本研究課題で実現したキャッシュ機構の概要を**図 4** に示す．3.1.1 節で述べたように，Kumoi では VM や物理計算機などのさまざまな資源を資源マップオブジェクトとして抽象化していることから，実装に用いている Java RMI の通信をフックすれば，キャッシュ機構を実現することができる．このアプローチには，従来からの**分散オブジェクトのキャッシュ技術**を活用できるという利点と，さまざまな資源に対して**統一的に**キャッシュを導入できるという利点がある．

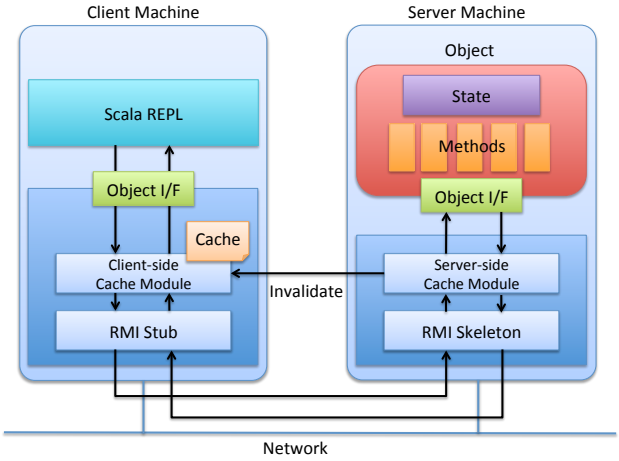

図 4：キャッシュ機構の概要

説明における便宜上，RMI 通信の呼び出し側を**クライアント**，要求を処理する側を**サーバ**と呼ぶ．Kumoi における各計算機は，基本的に対等に扱われることから，どの計算機もクライアントやサーバになることができる．また，クライアントはクラウド利用者の計算機とも異なる点に注意が必要である．

図 5 の実装は基本的に通常の Java RMI の機構と同一であるが，2 つの点で異なる．一つ目は，クライアント側，サーバ側ともに**キャッシュ機構のためのモジュール**が挿入されている点である．これらのモジュールはそれぞれの RMI スタブおよびスケルトン実装の直上に位置し，通信をフックして，対応する．二つ目は，サーバ側におけるオブジェクト実装の状態は実資源の状態に対応している点である．

3.2.2 キャッシュポリシーの指定

本研究のキャッシュ機構も，一般的なキャッシュ機構と同様に，キャッシュの保持期間やその内容の破棄のタイミングをキャッシュポリシーとして指定可能にしている．Kumoi では，ミドルウェアの実装が言語環境と強く統合されているため，**図 5** のようにそのポリシーを言語上のアノテーションとして指定する．

```
@remote trait HotPhysicalMachine {
  @persistcache def name(): String
  @persistcache def addr(): InetAddress
  @persistcache def os(): HotOS
  @persistcache def info(): List[PMInfo]
  @cache def stats(): List[PMStat]
  @cache(3000) def cpuRatio(): Double
  @cache(3000) def cpuAvailable(): Double
  @cache(3000) def memory(): Long
  @cache(3000) def freeMemory(): Long
  @persistcache def maxMemory(): Long
  // some shortcuts
  @persistcache def vmm(): VMM
  @cache def vms(): List[HotVM]
  // update (feedback) operations
  @invalidate @nocache def shutdown()
  @invalidate @nocache def restart()
  ...
}
```

図 5：キャッシュポリシーの指定

これらのアノテーションは，キャッシュ機構の中継機構によって，実行時に読み取られ，キャッシュの制御に活用される．現在，キャッシュの制御として実現できているのは，キャッシュの**保持期間指定**と**無効化**（invalidation）である．キャ

ッシュの保持期間指定は，通常時にキャッシュの鮮度を一定の範囲内に保つために使用され，無効化プロトコルは，一貫性に矛盾が発生するような状況の解決や障害への対策などに使用される．

3.2.3 実現方法の考察

Kumoi の設計では，耐障害性を向上させるために，**状態の最小化**（stateless）というアプローチを取っている．

Kumoi のマイクロカーネルや資源マップオブジェクトでは，状態の一貫性管理や障害対策を考慮して，状態を極力持たないように設計している．そのため，スクリプトから資源オブジェクトに操作を行うと，その内容を直ちに実際の資源に反映する．また，資源オブジェクトの状態をスクリプトから取得しようとすると，実際の資源から状態を直接取得し，返却する．

この方式はシステムの堅牢性を高めており，万が一,Kumoi のカーネルに障害が発生しても,Kumoi カーネルのみを再起動すれば,動作中の VM などを含めた現在の計算機の状態から，資源マップオブジェクトを正しく復元することができる．

また，この設計には別の意図もあり，資源オブジェクトと RMI 実装との通信をフックすることで,性能を向上させる機構，セキュリティ機構や耐障害性機構など，さまざまな機能を追加していくことを想定している．この**役割の分離**は，システムの設計を単純化するとともに，特定の資源に依存しない共通機能の導入を可能とする．本研究では，この機構を利用してキャッシュ機構を導入している．

3.3 セキュリティ機構との連携

Kumoi で，現在，セキュリティ機能として提供しているのは，下記の 5 つである．

- **認証機能**（authentication）: アカウント名やパスワードなどの情報をもとに，クラウド利用者が本人であるかどうか確認する．
- **認可機能**（authorization）: 上記の認証情報をもとに，仮想マシン（VM）やストレージ，ネットワークなどへの資源へのアクセスを制御する．
- **課金機能**（accounting）: VM やネットワーク転送などの資源の利用状況を追跡し，安全な場所に記録する．クラウド利用者の異常な資源利用の傾向を追跡可能にするとともに，資源の実使用量に基づいた料金の請求のためにも用いる．
- **監査機能**（audit）: 資源に対する操作履歴をすべて記録しておくことで，クラウド利用者の不正な操作を事後的に追跡可能にする．
- **通信経路の暗号化機能**（encryption）: Kumoi の管理用の通信をすべて暗号化することで，通信内容の読み取りや改ざんを防止する．

以上の機能は，Kumoi の当初設計からの自然な拡張として実現している．Kumoi では，物理計算機，VM，仮想ネットワークなどのさまざまな資源を資源マップオブジェクトとして抽象化するとともに，統一的な操作方法を与えている．これらのオブジェクトは，実際には，**遠隔オブジェクト**（remote objects）で実装している．よって，この遠隔オブジェクトへの通信をすべて中継すれば，セキュリティに必要な**完全調停**（complete mediation）を実現できる．

この通信の中継は，すべての上記のセキュリティ機能を実現するために利用しており，また，統一的な操作方法は，特定の資源に依存しないセキュリティ機能の実現に役立てている．

3.3.1 認証機能

セキュリティ機能を有効化した Kumoi では，シェル環境をカーネルに接続した後，すべての資源アクセスの前に認証を要求する.認証機能は,JAAS（Java Authentication and Authorization Service）を使用して実装しており，現在は LDAP（Lightweight Directory Access Protocol）を使用して認証を行う．認証が成功すると，シェル環境上の auth と呼ばれる特別な変数に認証情報を格納する.auth は AAA 型のオブジェクトであり，認可機能をはじめとするさまざまな機能で使用する．

3.3.2 認可機能

認可機能では，資源マップオブジェクトのすべてのメソッド呼び出しを，カーネル通信中継機構によりフックし，呼び出しを許可するかどうか判定する．呼び出しを許可しない場合には，例外を送出する．

本機構は，3.2 節のキャッシュ機構と連携するように設計されている．Kumoi では，ある物理計算機で動作している VM の一覧を取得する場合，次のような内部呼び出しを必要とする．

```
kernel.local().vmm().vms()
```

これは実際の資源の物理的な階層構造を反映しており，カーネルからの物理計算機の取得，ハイパバイザーの取得，VM 一覧の取得へと 3 回のメソッド呼び出しとそれぞれでの中央認証サーバでのアクセス権のチェックを必要とする．

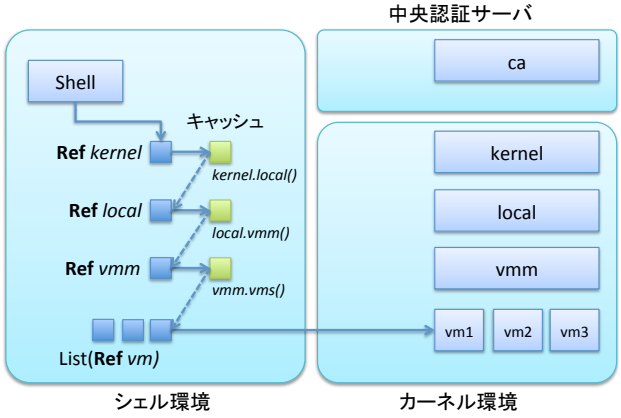

図 6：キャッシュ機構との連携

しかしながら，**図 6** のようにキャッシュ機構により，中央認証サーバでの認可処理，カーネルでの資源アクセス処理がともに省略される．

3.3.3 課金機能

課金機能では，課金処理専用のアクターをそれぞれのサーバ計算機に配置し，定期的に資源使用量を中央認証サーバに報告させることで実現している．Kumoi では，アクターを OS のプロセスのように使用している．また，VM の起動時や終了時など注意が必要な場面では，資源マップオブジェクトが直接課金処理を依頼できるようになっている．

3.3.4 監査機能

監査機能では，カーネル通信中継機能を使用し，資源マップオブジェクトのメソッド呼び出しをすべて記録する．時刻やホスト名を含む詳細な呼び出し情報を記録するとともに，呼び出しの成功の可否までを含めて記録することで，不正な操作の事後追跡を可能にする．

3.3.4 通信経路の暗号化機能

セキュリティ機能の拡張では，データセンター内部で使用する Kumoi の管理用の通信を暗号化する． Kumoi で暗号化が必要なのは，下記の 3 つの通信である．これらの暗号化は， 個別に SSL（Secure Socket Layer）化することで実現した．

- **RMI 通信**：資源マップオブジェクトへの同期的なアクセスに使用する．
- **アクター通信**： 非同期的に行うメッセージ通知に使用する．また，Kumoi 内部の分散アルゴリズムの実装に使用している．
- **遠隔クラスローダ通信**： シェル環境で定義したクロージャをカーネル環境で実行する場合など，遠隔のクラス定義を必要とする場合に用いる．

3.4 冗長要素を利用した透過的な障害対策

本研究の Kumoi では，多数の VM をデータセンター上に配置する場合などの操作を，**図 7** のような非常に簡潔な記述で行う事ができる．

ここでのポイントは，`take(n)`関数であり，物理計算機の資源プール `pms` が有する多数の要素の中から先頭の 5 台しか使用していない．そこで本研究では，`take(n)`以降の処理で障害が発生した場合に，使用しなかった 5 台以外の冗長要素を活用して代替処理を行うことで，簡潔なスクリプト記述のまま，透過的な障害対策を実現する．`take(n)`関数以外にも，同じように未使用の冗長

要素を有するリスト操作関数は多数あり，これらに対しても同様に実装を行った．

```
kumoi> pms.filter(_.cpuRatio < 0.3)
  .filter(_.freeMemory > 512*1024*1024)
  .take(5)
  .zipWithIndex
  .foreach(deployVM)
```

図 7：5 台の VM を起動する場合の記述例

3.5 仮想ネットワークへの対応

本研究では，機密性やプライバシーへの配慮から**仮想ネットワーク**への対応も行った．これらの仮想ネットワークへの対応も，Kumoi の当初の設計の自然な拡張で実現されている．

本研究の仮想ネットワークは，OpenFlow と Open vSwitch を使用して実現している．実装としては，**OpenFlow コントローラ**，**仮想スイッチ**をそれぞれ資源マップオブジェクト化することで実現した．

OpenFlow コントローラの実装では，Kumoi の実装で使用している Scala との親和性から，Floodlight を使用した．本コントローラの実装では，Floodlight のモジュールの一つとして実現しており，Java RMI を使用した Kumoi のカーネルの通信方式と同一にすることで，OpenFlow のルールをシェル環境から Scala のクロージャとして動的にコントローラに登録可能にした（**図 8**）．そのため，同一のシェル環境から VM，仮想ネットワークなどを一体的に制御することが可能になっている．

一方，Open vSwitch 対応の実装では，仮想スイッチ，仮想ブリッジ，仮想ポートのそれぞれを資源マップオブジェクト化した．これらは本来の Open vSwitch 機能として使用することも可能である．また，前述の OpenFlow コントローラと接続して OpenFlow 対応スイッチとして使用することもできる．

```
kumoi> ofc.add({ case PacketIn(sw, id,
  inPort, totalLen, reason, data, ctx) =>
    sw.send(PacketOut(bufferId, inPort,
     data, List(Output(Port.Flood, 0))),
       context)
    true
})
```

図 8：OpenFlow コントローラの操作例

3.6 VM 内で稼働するアプリケーションへの対応

最後に，VM 内で稼働するアプリケーションも統一的な環境から操作できるようにするための拡張を行った．**図 9** はそれぞれの VM 内で稼働している Apache ウェブサーバのコンフィグレーションを書き換え，再読み込みを行う場合の記述例である．この機能は，個別のアプリケーションを資源マップオブジェクトで表現し，VM 内にもカーネルを配置するという従来設計の自然な拡張で実現されている．また，クラウド環境において VM 内で利用されるアプリケーションは多岐にわたることから，個別のアプリケーションに対応する資源マップオブジェクト生成を支援する機構も同時に作成した．

```
kumoi> pms.map(_.vms.map { vm =>
  val apache = vm.local.apps(0)
                 .asInstanceOf[Apache]
  apache.maxClients = 1024
  apache.keepAliveTimeout = 5
  apache.reload()
}
```

図 9：VM 内の Apache を操作する場合の記述例

(2) 当初計画の達成状況について

本研究課題では，クラウド基盤ミドルウェアのスケーラビリティ向上を実現することを目標に，着々と研究を進めた．まず，当初の目標としていたキャッシュ機構を実現し，性能の向上を実現し

た．また，このキャッシュ機構は学際的な情報基盤環境には欠かせないセキュリティ機能と連携して動作する．さらに，スケーラビリティを実現するためには，障害対策が欠かせないことから，未使用の冗長要素を利用した透過的な障害対策の研究を行った．最後に，学際的なさまざまな共同研究の要望に応えるために，仮想ネットワークや VM 内アプリケーションへの対応を行った．

一方で，当初予定していた実際の学際大規模情報基盤共同利用環境でのミドルウェア評価については実施できなかった．また，数理的検証を行った分散アルゴリズムのシェル環境を活用した評価についても，着手したものの完了に至らなかった．これらは，ミドルウェアの改良に想定以上の時間を要し，時間が十分確保できなかったことが理由である．また，当初研究協力者として予定していた学生が十分確保できず，開発用として予定していた所属機関のクラスタ機器が故障したことも研究の妨げとなった一因である．達成できなかった事項については，今後の検討課題とする予定である．

4. 今後の展望

本研究課題の成果は，平成 25 年度採択課題「クラウド基盤ミドルウェアの運用性向上に関する研究」に継続して発展させる予定である．主要な機能の研究開発については平成 24 年度でほぼ完了したことから，今後は実際の学際大規模情報基盤共同利用環境を利用した動作検証や評価を中心に実施する予定である．

また，本研究課題は Kumoi という特定のクラウド基盤ミドルウェアを中心に研究を実施してきたが，本研究で得られたいくつかの成果については他のミドルウェアにも適用可能であると考えている．たとえば，近年の Windows OS では Windows Management Instrumentation (WMI)を通じて VM や物理計算機の情報を取得し，操作するための機構を提供しており，これを利用すれば，Kumoi の資源マップオブジェクトの代替として利用することができる．また，実際の学際大規模情報基盤共同利用環境で広く利用されている CloudStack や OpenStack に成果を適用することも検討しており，これらのミドルウェアに適用すれば，実際の学際大規模情報基盤共同利用環境での実用化により近いと考えられる．

最後に，本研究課題で得られた成果を取り込んだ最新版の Kumoi は，GitHub 上でオープンソースとして広く一般に公開している[5]．この成果が，JHPCN の対象領域の一つである超大規模情報システム研究関連分野の研究の発展の一助となれば幸いである．

5. 研究成果リスト

(1) 学術論文

[1] 杉木 章義，加藤 和彦，”仮想資源管理基盤におけるキャッシュ機構の導入”，情報処理学会論文誌コンピューティングシステム (ACS 41)，6(1)，pp 31-44，2013 年 1 月．

(2) 国際会議プロシーディングス

なし

(3) 国際会議発表

[2] Akiyoshi Sugiki and Kazuhiko Kato, “Elements and Composition of Software-defined Data Centers”, 13th ACM/IFIP/USENIX Int’l Conf. on Middleware (Middleware 2012), 2 pages, Demos and Posters Track, December 2012.

(4) 国内会議発表

[3] 奥畑 聡仁，杉木 章義，加藤 和彦，“クラウド基盤ソフトウェアにおける Lineage を利用した障害対策手法の検討”，並列／分散／協調処理に関するサマーワークショップ (SWoPP 2012)，2012 年 7 月．

[4] 大山 裕泰，杉木 章義，加藤 和彦，“サービス指向の IaaS クラウドシステム環境の構築”，インターネットコンファレンス 2012，ポスター展示，2012 年 11 月．

(5) その他（特許，プレス発表，著書等）

[5] Kumoi ソースコード公開 (Apache License 2.0) https://github.com/axi-sugiki/kumoi ，2013 年．