Vol.2015-DPS-162 No.2
Vol.2015-CSEC-68 No.2
2015/3/5

分散システムの環境変化に動的適応できるミドルウェアの設計と実装

孫　静涛[1,†1,a)]　佐藤一郎[1,b)]

概要：近年, 大量なデータが集積されるようになり, それを安価に処理できるスケールアウト型の分散システムに関する研究が大きく注目されている. しかし, 既存の分散システムでは, 頻繁に変化するシステムのニーズやアプリケーションの要求などを考慮せずに, 固定なアーキテクチャを用い, システムを構築する事例は殆どである. そこで, 本稿では, 既存の分散システムにこれらの環境変化を動的適応できるミドルウェアを提案する. 本提案手法の鍵となるアイデアは, 既存の分散システムが適応ポリシーを基づき, ローカルコンピュータのソフトウェア・コンポーネントをリモートコンピュータに再配置することで, 分散システムのアーキテクチャを動的に変え, 環境変化に適応する. また, 適応ポリシーに関しては, システムの利用者は我々が提案するポリシー言語で記述可能にするものである. 本稿では, 本提案の動的自適応ミドルウェアの設計と実行について, 説明した上に, 3つの応用事例を用いて本ミドルウェアの評価結果を論じる.

キーワード：動的適応性，ポリシー言語，コンポーネント再配置，分散システム

1. はじめに

近年, データの巨大化とネットワークの発展により, スケールアウト型の分散システムに関する研究開発は大きく注目されている. しかし, 既存の分散システムにおいて, システムの様々な実行環境やソフトウェアの要件 [10] などは常に変化する. 例えば, コンピュータ台数の増減や, エージェントの移動により, ハードウェアとソフトウェアの環境変化などである. それに対して, システムは不変のままで, 運用してきた事例は殆どである. そこで, 我々は, 既存の分散システム自身が様々な変化を動的適応できるポリシー・ベースでの分散ミドルウェア「ミモザ」を提案する.

分散システムに適応性を導入する研究 [1], [2], [4], [6], [7], [8], [9], [11], [12] などが数多く行われてきた. しかし, これらの研究では, 分散システムの状況変化をパラメータの変更 [18] やソフトウェアの書き換えやポリシーの策定 [5] などで提案してきた. これらの研究方法を利用すれば, システムの信頼性が高めることができるが, 可再利用かつ状態を持つソフトウェア・コンポーネントの再配置を行うことすることができない.

そこで, 本研究では, 従来研究と違い, 様々な変化が起きる際に, コンピュータ上に実行しているソフトウェア・コンポーネントは事前に用意したポリシーに従い, 指定されたコンピュータに再配置する. 各ソフトウェア・コンポーネントはソースコードだけではなく, 状態も持っているため, 配置先に到着した後に, 中断された処理を直ちに, 再開することができる. それと, ソフトウェア・コンポーネントの自由な組み合わせにより, 分散システムのアーキテクチャが動的変えられる. つまり, 本研究は分散システムにおける様々な状況が変化すると伴に, 分散システムはその状況下での優位なアーキテクチャの元で, 処理を行いながらシステムの信頼性を高めていくと実現することができる. また, 本研究では, ソフトウェア・コンポーネントとポリシーを分離して記述できるので, システムの管理者とアプリケーションの開発者はそれぞれの関心なことを専念でき, 記述されたポリシーとプログラミングしたソフトウェア・コンポーネントも分散システム状況の変化に応じて, 再利用することが可能になる. さらに, 既存の分散システムにコンピュータの動的に追加・削除できるので, 最小台数のコンピュータで必要なリソースを提案することで, システムの全体的な運用コストを下げることができる点になる.

「ミモザ」システム, アーキテクチャレベルでの動的適応性の実現に向けて, 以下の三点に集約される.

[1] 国立情報学研究所
NII, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, Japan
[†1] 現在，総合研究大学院大学 複合科学研究科 情報学専攻
Presently with The Graduate University for Advanced Studies
[a)] sun@nii.ac.jp
[b)] ichiro@nii.ac.jp

- 信頼性, 耐障害性, 再利用性の高さ
- コンポーネントの再配置先の記述の明確化
- ポリシー言語によるソフトウェアへの親和性

プログラマが事前に分散システムやアプリケーションの状況変化を記述し, その記述されたポリシーの組み合わせことでシステムの動的適応性を向上できるように構成される必要がある. しかし, ソフトウェア・コンポーネント同士に対して, 複数のポリシーの組み合わせることで, コンポーネント間に衝突や分岐を生じることがあるため, ポリシー言語で条件を記述する際に, どんなコンポーネントを優先的に実行すべきという判断はプログラマの経験次第である.

本論文では, 以下のように構成されている. 第 2 章では, 自適応システムの関連研究について概述する. 第 3 章は我々は本研究の基本的な考え方を説明する. 第 4 章では, ポリシーを記述するための適応言語の文法と事例を紹介する. 第 5 章では,「ミモザ」ミドルウェアの設計実装方法を具体的に示す. 第 6 章では「ミモザ」システムへの応用事例についてを紹介する. 第 7 章では, 三つソフトウェア・コンポーネントを用いて,「ミモザ」の評価実験結果を説明する. 最後に, 本稿をまとめ, さらに今後の展望を述べる.

2. 既存研究

既存分散システムに動的適応性を導入するために, いくつの研究がなされている. 典型的な例題としては,ORB(Object Request Broker) や遺伝プログラミングや適応記述言語の方式の適応などが挙げられる. 例えば,[4], [7], [9] の研究では, クライアント側とサーバ側のプログラムはお互いに協調しながら変化を適応している. クライアント側のオブジェクトを自律的に優位なサーバを選択し, そのサーバのオブジェクトのメソッドを動的に呼び出すような仕組みで, システムの変化を適応していた. しかし, この方式では, プログラムの連携方式しか変えていないし, 適応可能範囲が限定されているため, コンピュータの動的に追加・ 削除することができない. また,MIT の John R. Koza 氏 [5] を初めとする遺伝プログラミングの研究では, ソフトウェアを書き換えは事前予測不可能であるし, 変化を予期するのに対して, プログラムは膨大な計算時間とコンピュータのリソースが必要のため, 頻繁に起きる変化を処理し切れない状況が多い. そして,Nicodemos Damianou チーム開発していた Ponder 言語と Amano チームの適応記述言語研究 [1], [2] では, 分散システムの変化を適応するソフトウェア・コンポーネントそのものの定義, 状態遷移の変化とタイミングを記述可能ですが, 具体的なソフトウェア・コンポーネント再配置先は記述できない.

本研究では [14], [15], [17], ポリシー言語を新たに定義し, ソフトウェア・コンポーネントの基本戦略はもちろん, 再配置先を記述することができる. また, 本研究では, モバイルエージェントをベースに実装し, 分散システムの変化を動的適応しているが, モバイルエージェント技術 [16] と違い, それはモバイルエージェント技術は動的に起きる変化をサポートしていない. しかし, 本研究では, ポリシー記述に基づき, ソフトウェア・コンポーネントの実行位置を他のコンピュータに再配置することで, 動的適応性を実現した. また, 本研究では分散システムの状況変化を柔軟に応じて, 分散システムのアーキテクチャを自由に変えられ, アプリケーションの実行環境を相応しいサポートの提供は可能である.

3. 提案手法

既存の分散システムでは, システムの実行環境やソフトウェアの要件定義などは常に変化する. しかし, 分散システムはこのような変化を柔軟に対応していない. また, ソフトウェア開発にあたって, 固定化されたアーキテクチャに基づき, ソフトウェア内部でこれらの変化の対応を取らざるえない. そのために, これらの変化に応じて, システムアーキテクチャレベルでの動的に適応できる能力は必要としている. そこで, 本研究では, 従来の分散システムアーキテクチャを特化するではなく, アーキテクチャでのメリットを持ちつつ, ソフトウェア・コンポーネントは事前に定義していたポリシーに従い, 再配置することで, 既存の分散システムに動的適応性の導入を目標にしている.

3.1 システムの構築要件

既存分散システムに動的適応性を導入するためには, 本研究は次のような構築要件に絞る.

- 動的適応性
  分散システム自身とシステム外部の変化を適応するため, 動的適応性が必要としている. また, ソフトウェア・コンポーネントは異なるシステム上に実行できるので, 各コンピュータの情報を持つコンポーネントの移動方式を用いることで, コンピュータ同士はお互いに協力できる. そのため, 分散システムは自己適応可能になる.
- フォールトトレランス性
  多くの分散システムはハードウェアとソフトウェアの障害を検出するために, フォールトトレランスメカニズムを採用している. 我々のミドルウェアも最大限でハードウェアとソフトウェアの障害を自己検出することが望ましい. 障害が発生しても, 処理は中断せずに, システムを回復できる能力が要求される.
- 関心事の分離・可再利用
  ソフトウェア・コンポーネントとポリシーはそれぞれの特性を持っているために, 独立化させるべきである. ソフトウェアの内部でポリシーを記述する場合には, ソフトウェアは再利用できなくなるし, ソフトウェアの開発にも困難さを加える. そのために, お互いに分離すれば, 開発者への親和性を高めるので, 頻繁に生じる

同様な変化を迅速に再利用することができる.

- 汎用性の高いアプリケーションの開発
  特殊なアプリケーションを構築するために, 分散システムを採用するではないはず. そのため、我々のシステムは汎用性がある一般的なアプリケーションの開発にしても, 分散システムの環境変化が起きた際に, アプリケーションではなく、分散システム自身がこの変化を適応できる能力が要求される.
- リソースの有効利用
  複数のコンピュータから構成されたコンピュータは限られたリソースを持っている. 例えば,CPU の処理能力やメモリ容量やネットワークの振幅および位相などがある. そのために, 我々のミドルウェアは出来る限りで, コンピュータのリソースを最大限に活用させることが要求されている.

3.2 基本考え方

本研究で提案する動的適応可能なミドルウェア「ミモザ」(図 1) の鍵となる基本的なアイデアは二つに挙げられる. 一つ目は分散アプリケーションは一つ以上のソフトウェア・コンポーネントを持っているので, ソフトウェア・コンポーネントの再配置をベースにし, 様々な変化を適応する. 二つ目はこれらのソフトウェア・コンポーネントはどんな状況下で, どのように再配置するのはユーザが一つのポリシーとして定義できる仕組みを導入する. 同じソフトウェア・コンポーネントに対して, 複数のポリシーを制御する場合には, コンポーネント間の衝突や分岐 [3] を起こりうるので, 我々が新しいポリシー言語を用いて, ソフトウェア・コンポーネントの再配置を行うことにした. 本研究で提案するポリシー言語は条件と再配置という二つの部分を用いて構築する. 前者はシステムの状態やネットワークの状況などの条件を複数に記述できるようにした. 後者は図 1 に示したようにシステムの状況に基き, ソフトウェア・コンポーネントはポリシーに指定されたコンピュータに移動し, 移動したオブジェクトは再配置側のコンポーネントを通信できるような動的メソッド呼び出し仕組みをコアとして,「ミモザ」システムの設計と実装を行うことにした.

本提案手法では, 以下のようなメリットを持つ.

- 分散システムの利用状況が変化する際に, システムアーキテクチャは動的に変えられ, その状況下に相応しい処理をサポートできる.
- コンピュータの動的加入・脱離は, 分散システムが自己判断し, 状況の変化を適応する.
- ネットワークのレイテンシーが高くなる前に, ソフトウェア・コンポーネントの再配置を行い, 複数回での通信を一回までに減らせるので, 不安定のネットワークを使用していた時にも, システムの信頼性が高くなる.
- データサイズの巨大化を進んでいる現在では, データを転送してから処理するより, コンポーネントの再配置を用いけば, データの処理コストはより小さくなる.
- ソフトウェアの内部で適応条件を加えなくても, アプリケーションの開発はよりシンプルになるので, 開発者への親和性とソースコードの可読性が高くなる.

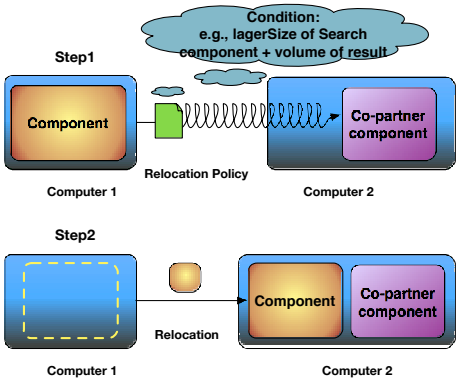

図 1 ポリシーによるコンポーネントの動的適応.

4. ポリシー言語

本章では, ソフトウェア・コンポーネントを再配置するためのポリシー言語の設計について, 説明する.

4.1 再配置ポリシーの設計

ポリシー言語を説明するため, 本研究が使う表記法を定義する.L はロケーション名の集合体として定義する.X はロケーション変数名の集合体として定義する.current は頻繁に使用されるコンピュータのコンポーネントである.C はコンポーネントの識別子の集合体として定義する.「ミモザ」は下記の文法を用いて, システムの利用者はポリシーを記述できる.

- 文法
  D は位置処理式の集合体として定義する.C は 4.1 で定義された条件になる.E in 0 は E に省略する. τ は, 呼び出す動作として定義する.

$$
\begin{aligned}
D,D_1,D_2 &::= \ell\,[E|P] \text{ (Located component)}\\
&\mid D_1 \| D_2 \text{ (Distributed components)}\\
E,E_1,E_2 &::= C \textit{ then } M \textit{ in } E \text{ (Conditional action)}\\
&\mid E_1 + E_2 \text{ (Alternative selection)}\\
&\mid \textit{0} \quad \text{(Termination)}\\
M &::= \textit{moveTo}(x) \quad \text{(Migration)}\\
&\mid \textit{copyTo}(x) \quad \text{(Duplication\& migration)}\\
&\mid \textit{remove} \quad \text{(Elimination)}\\
&\mid \tau \quad \text{(Internal execution)}\\
P,P_1,P_2 &::= P_1,P_2 \quad \text{(Composition)}\\
&\mid \textit{A} \quad \text{(Component)}\\
&\mid \epsilon \quad \text{(No component)}
\end{aligned}
$$

これらのポリシーに従い, コンピュータ上に実行して

いるソフトウェア・コンポーネントは指定されたコンピュータに再配置することができる. 我々は適応ポリシーの構成要素条件と再配置場所の記述を分割した. 前者は真であれば, 後者は活用され, 前者は偽であれば, 後者は実行されない仕組みとなっている.

- 条件

ここで, ポリシー言語の条件の集合体は C として定義し, 以下の式で表現することができる. ϕ は論理的な述語であり, 真または偽を返す.

$$C, C_1, C_2 ::= \varnothing \mid \neg C \mid C_1 \wedge C_2 \mid C_1 \vee C_2 \mid true \mid false$$

- パラメータ

各述語は下記のようにいくつのパラメータを持つことができ, 条件を満たせば,true を返し, 満たさなければ,false を返す.

$$exist(A, \ell)(exist : P \times L \rightarrow true \quad or \quad false)$$
$$delay(time)(delay : T \rightarrow true \quad or \quad false)$$
$$received(N, \ell, A)(requested : P \times N \times L \rightarrow true \quad or \quad false)$$

- ポリシー

上記の文法を用いて, 我々は図 2 のように以下の五つの定義する.

(1) 吸引ポリシー

コンピュータ 1 の上で実行しているソフトウェア・コンポーネント A はシステムの状況変化に応じて, コンピュータ 2 に移動する. ポリシー言語を用いて, 下記の式で表現する.

$$\ell[exist(A, another) then \quad moveTo(another) inE \mid A]$$

(2) 反発ポリシー

コンピュータ 1 の上で実行しているソフトウェア・コンポーネント A と A‘はシステムの状況変化に応じて,A‘は反発され, コンピュータ 2 に戻る. ポリシー言語を用いて, 下記の式で表現する.

$$\ell[exist(A, cureent) then \quad moveTo(another) inE \mid A]$$

(3) 拡散ポリシー

コンピュータ 1 の上で実行しているソフトウェア・コンポーネント A はシステムの状況変化に応じて, 自分自身を複製しからコンピュータ 2 に転送する. ポリシー言語を用いて, 下記の式で表現する.

$$\ell[\neg exist(A, another) then \quad copyTo(another) inE \mid A]$$

(4) 終結ポリシー

コンピュータ 1 の上で実行しているソフトウェア・コンポーネント A と A‘はシステムの状況変化に応じて,

$$\ell[exist(A, cureent) then \quad remove \quad in \quad 0 \mid A]$$

集合する. ポリシー言語を用いて, 下記の式で表現する.

(5) 時系ポリシー

コンピュータ 1 の上で実行しているソフトウェア・コンポーネント A はタイマーをかけられ, 設定時間になった場合には A はコンピュータ 2 に移動する. ポリシー言語を用いて, 下記の式で表現する.

$$\ell[delay(t) then \quad remove \quad in \quad 0 \mid A]$$

既存の分散システムは常に様々な変化が起きている. 従って, 分散システムは複数の変化を適応できると期待されている. そこで, 本研究はこれらの変化を我々が定義するポリシー言語を用いて定義することができ, 高信頼性の分散システムの実現は可能になる.

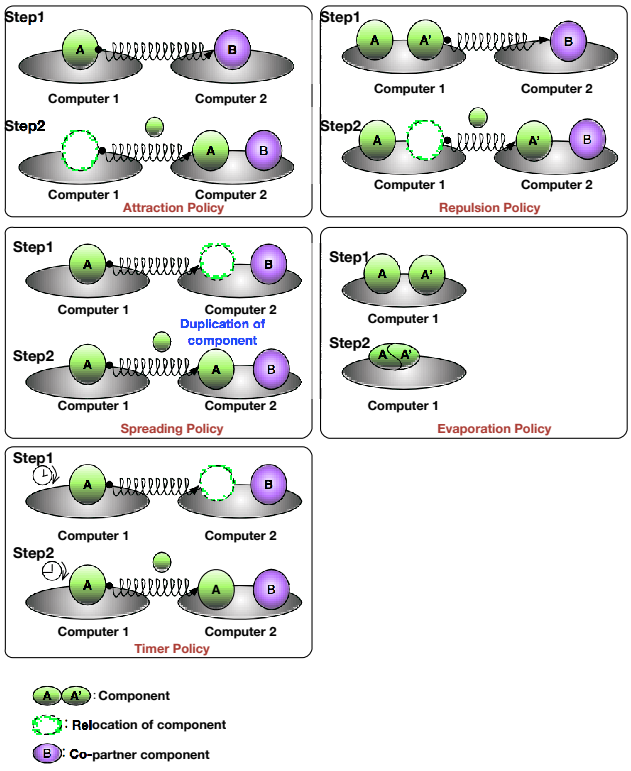

図 2 再配置ポリシー.

5. ミモザシステムの設計と実装

前章で紹介したシステムの要件定義を満たすために, 我々は「ミモザ」システムを設計した. 本章では, 「ミモザ」の基本構造及び各機能の詳細について, 説明する.

5.1 ミモザシステムの構造

「ミモザ」システムは, オペレーティング・システムとアプリケーションの間に位置付けられ, コアミモザ, アダプテーションマネージャーとコンポーネント・ランタイムシ

ステムと言った三つの部分から構成される. また, それぞれのコンピュータ上に実行している「ミモザ」同士はネットワークを介して相互に通信を行う (図 3). コアミモザはソフトウェア・コンポーネントとポリシー記述言語をサポートするために, インタフェースの定義, ポリシーデータベースの利用, ライブラリなどを提供する. アダプテーションマネージャーはポリシー言語で記述された適応政策の解釈と分析において, 役目を果たす. コンポーネント・ランタイムシステムはソフトウェア・コンポーネントの実行, 複製, 再配置, メソッド呼び出しなどの機能を組み合わせたサービスを行う. 我々が提案した「ミモザ」システムは Java 仮想マシン（JVM）上で実行されているため, コンピュータの実行環境に頼らずに, 異なるオペレーティングシステムでも実行可能というメリットがある. それと, 本システム上に動いているソフトウェア・コンポーネントはすべて Java オブジェクトから構成されている.

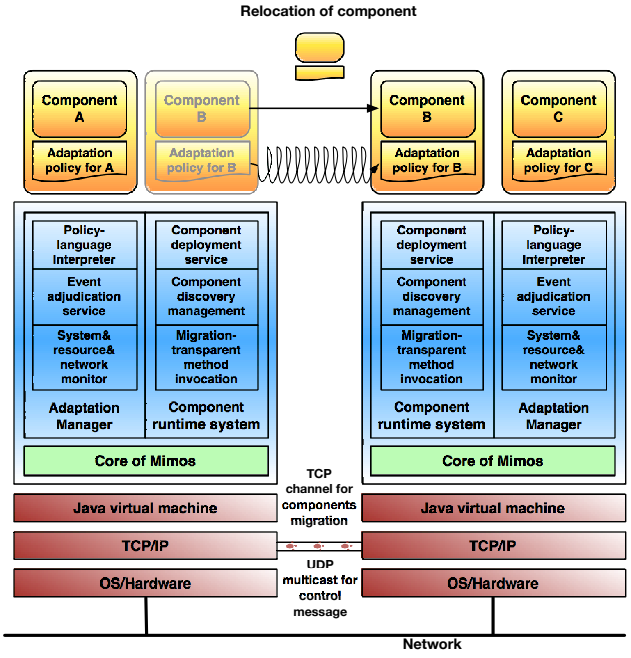

図 3 ミモザミドルウェアのアーキテクチャ.

5.2 アダプテーションマネージャー

各アダプテーションマネージャーは定期的に UDP マルチキャスト形式で自身の IP アドレスメッセージとして自身以外のコンピュータに通知する. 受診されたコンピュータたちは TCP チャネルを通じて自身の IP アドレスを送信側のコンピュータに返す形でコンピュータの増減を自動認識する.

また, アダプテーションマネージャーは専用のインタプリタを用いるため, ポリシーデータベース上に保存しているポリシーの条件を解釈・分析することができる. 外部システムから分散システムの環境変化を検出した場合に, アダプテーションマネージャーはその条件変化に従い, コンポーネント・ランタイムシステムに指示を出し, ソフトウェア・コンポーネントの再配置を命じる. その環境変化は例えば, ユーザーの要件定義とコンピュータリソースの可使用率などがある.

各ポリシーは 4.1 節で定義されたポリシー言語の文法に基づいて記述することができる. 我々が提案するポリシー言語は外部条件の記述部分とソフトウェア・コンポーネントの配置先から構成されていて, ポリシーデータベースに保存しておく. 前者は一階述語論理の述語 C,C_1, C_2…, のように様々なシステムとネットワークの属性を反映する. 例えば, プロセッサの処理能力, ネットワーク接続, およびアプリケーション固有の条件などである. 後者はソフトウェア・コンポーネント再配置場所 E,E_1, E_2..., を記述する. 監測モニターから分散システムの環境変化をお知らせると, ソフトウェア・コンポーネントは一つのポリシーの戦略に従い, 目的地のコンピュータに再配置を行う. 一旦, コンポーネントを目的側のコンピュータに到着すると, このコンポーネントを持つポリシーは目的側のポリシーデータベースに登録を行う. 目的側のアダプテーションマネージャーはこのポリシーを解釈・分析する. コンポーネントの衝突や分岐が発生する場合に, このコンポーネントを一時的にブロックする. その反面で, 再配置されたコンポーネントと目的側のコンポーネントを動的メソッド呼び出しメカニズムに介して, 中断された処理を再開させる.

5.3 コンポーネント・ランタイムシステム

各コンポーネントランタイムシステムは各自のコンピュータ上で実行されていて, そして, 各コンポーネントはライフサイクル状態を持つアクティビティスレッドを実装される. 例えば, コンポーネントの作成, 複製, 移動, 終結がある. 一旦, コンポーネントのライフサイクル状態が変われば, コンポーネント・ランタイムシステムは特定のイベントを出す. このようなイベントを捉えるためには, 各コンポーネントはもう一つイベント監視リスナーインタフェースを実装されている. 各コンポーネントは複数のリスナーインタフェースを持つことが可能である.

各コンポーネントラン・タイムシステムは TCP チャネルを介して, モバイルエージェント技術のようにコンポーネントの移動を実現可能になる. 特に,「ミモザ」は Java のオブジェクト直列化パッケージの使用により, コンポーネントはビットストリームに変換され, 目的地に到着した後に, ビットストリームから再びコンポーネントを還元される. そのため,「ミモザ」はコンポーネントのソースコードだけでなく, コンポーネントそのものの状態までを移動先に持っていくことができ, 中断された処理を再び再開することができるようになる. コンポーネントは目的地に配置された後で, 動的メソッド呼び出しメカニズムの使用によ

り, ローカルまたはリモートコンピュータで実行されている他のコンポーネントから呼び出すことができる.

5.4 ミモザの実装

「ミモザ」システムはモバイルエージェント技術をベースにして実装を行っているが, 既存のミドルウェアと大きく違い,「ミモザ」はポリシーベースに基づき, ソフトウェア・コンポーネントの再配置を狙って分散システムの動的適応を導入したものである. 現在,「ミモザ」は Java の直列化・反直列化パーケージを採用し, コンポーネントの移動と複製を実装を行っている. しかし, 直列化・反直列化パーケージはスレッド管理をサポートしていないため, 移動や複製を行う際に, スレッド処理を一時的に停止させる必要がある. また, 各ソフトウェア・コンポーネントは汎用性を持つため, 開発者に負担を欠かずに, 関心事を専念することができるし, すべてのコンポーネントは標準の JAR ファイル形式にまとめているので, 使用にも利便性が高い. それから, ソフトウェア・コンポーネントは一台のコンピュータから他のコンピュータに再配置する際に, 悪意的な改ざんを防がなければいけない. そのため, ソフトウェア・コンポーネントを移動・複製する前に, 暗号化処理を加え, 目的地のコンピュータに到着した後に, 暗号を解く仕組みを導入する必要がある. 現段階は,「ミモザ」はこのような仕組みを実装されていないけれど, 次回の論文で論じたいと考えている.

6. 応用事例

本章では,3 つの応用事例を用いて,「ミモザ」システムの応用について, 紹介する.

6.1 遠隔情報検索

遠隔情報を検索する例では, ユーザがローカル側のコンピュータから検索したいキーワードをリモートサーバ側に送り, リモートサーバ側のコンピュータはファイルシステム上に保存されているファイルから検索されたいキーワードを一行ずつ読み込み, ローカルコンピュータにその結果を返していく. しかし, データのサイズが大きければ大きいほど, 検索にかかってしまう. また, 二台のコンピュータは通信中に, 一旦ネットワークが切れてしまう場合には, ローカルコンピュータから再送信し, 検索を要求しなければいけない事情がある. この場合には, 図 4 のように「ミモザ」システムはこれらの変化を動的に適応できる. ユーザは吸引ポリシーを定義すれば, ローカル側の検索コンポーネントをリモート側に移動し, リモート側のコンピュータで検索処理を行う. そうすることにより, コンピュータ間の通信遅延問題を避けられるし, 同一のコンピュータ内の情報検索なので, 結果をより短時間で得られるメリットがある. 結果を一旦得られたら, 検索コンポーネントは再びにローカル側に戻れば, 複数回での通信を一回で済むことができる. また, システムのアーキテクチャは従来のクライアントコンポーネントをサーバ側に送ったため, サーバ側は両コンポーネントを持つことで, システムアーキテクチャは Peer-to-Peer 方式に変更することができる.

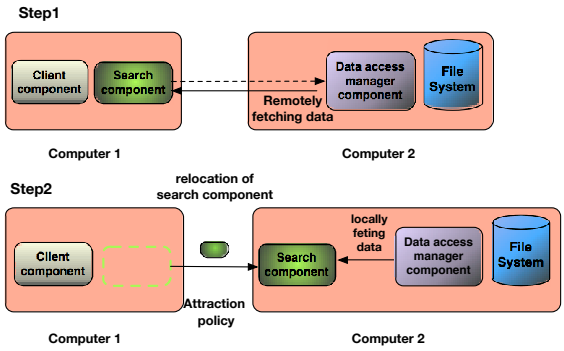

図 4 遠隔情報検索.

6.2 Publish/Subscribe システム

Publish/Subscribe システム [13] では, メッセージの送信側（pub）は複数のブローカーに経由して, 受信側（sub）に通信する仕組みとして構築されている. しかし, 複数のブローカーの実行を維持するにはコストがかかるので, 使用していないブローカーの釈放や追加するブローカーをロードするには, 向いていない. また,Pub 側からのメッセージに対して, ブローカー上に実行しているソフトウェア・コンポーネントを自由に組み合わせすれば, システムの全体的な信頼性と有効性を高めていく必要がある.

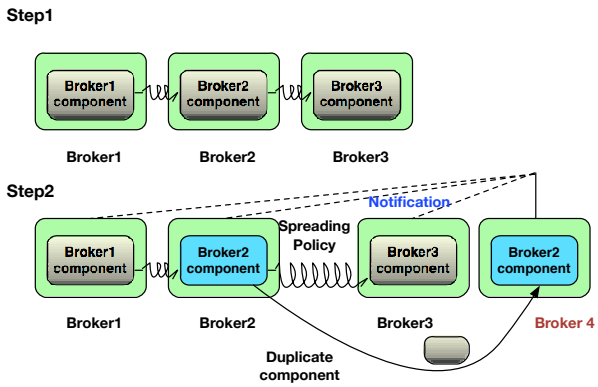

図 5 Publish/Subscribe システム.

その場合には, 図 5 のように「ミモザ」システムはこのような変化を動的に適応できる. ユーザは我々が提案するポリシー言語の使用によれば, 様々な対応ができる. 例えば, 動的にブローカー増加には, 拡散ポリシーを使えば, 一つのブローカーのソフトウェア・コンポーネントを複製し, その複製したものを新しいブローカーに再配置することで, ブローカーは自律的にブローカーグループに追加すること

ができる. また, アプリケーションの要求を変われば, いくつのブローカー上に動いているソフトウェア・コンポーネントは終結ポリシーに従って, 自由に組み合わせすることで Sub 側への適応能力が進化させることができる. また, ブローカーのアーキテクチャは従来の Server/Client 方式から Peer-to-Peer 方式に変更することができる.

6.3 センターシステム

既存のセンサーネットワークは様々な変化を起きている. 典型的な例としては, センサーノードは自分の観測領域内に環境変化や人の存在を検出した後に, その地理的に隣接するノード同士は一定期間後にも同様な変化を検出できる傾向がる. このような変化を起きた際に, 図 6 に示した応用のように, ソフトウェア・コンポーネントは隣接するノード同士に自分の複製を作り, そのクローンを隣接ノードに再配置することで, 動的な変化を適応できる. また, センサーノード情報の転送時にネットワークの不安定により, 通信が遅れる場合には, 吸引ポリシーを定義し, コンポーネントの再配置すれば, 通信回数を大幅に減らせるので, このような変化を適応できる. そして, 新しいセンサーノードを追加したり, リンク障害を生じたりする時に, ソフトウェア・コンポーネントは反発ポリシーや時系ポリシーに従えば, このような変化にも動的適応できる. また, システムのアーキテクチャは従来の Peer-to-Peer 方式から Server/Client 方式に変更することができる.

図 6 センサーシステム.

7. 性能評価

本章では,「ミモザ」システムのパフォーマンスを評価するためのシミュレーション環境と結果を説明する.

7.1 シミュレーション環境

我々は三台のコンピュータを用いて, 実験環境を構築した. 下記の表 2 は本実験環境の情報である.

そこで, 実験結果を説明しやすくために, 我々は三台のコンピュータを C_1,C_2,C_3 として定義した.C_1 コンピュータ上には三つのソフトウェア・コンポーネントは Component_1,Component_2,Component_3[*1] として実行している.

[*1] コンポーネントのサイズは約 20KB である.

表 2 実装環境

	System Information
CUP	2 Ghz Intel Core i7
OS	OS X Mavericks v10.9.2
Memory	8GB
Net Speed	28.98Mb/s
Language	java 1.7.0.45

その他の二台は「ミモザ」システムを起動したのみであり,C_1 上に動いているソフトウェア・コンポーネントはこの二台は一切持っていないである. 表 1 は C_1 コンピュータ上に実行しているソフトウェアの情報を示したものである. 三つのソフトウェア・コンポーネントは一台のランタイムシステム上に実行しているため, 同じランタイムシステム ID を持ちながら, 各の特殊なコンポーネント ID も持っている. コンポーネントを実行時に, コンポーネントは Creation という生命周期に入っており, 変化が起きた時に, 複製や再配置や終結などの状態に変更する.Delay(ms) とは, 三つのソフトウェア・コンポーネントを TCP チャネルに介して, 転送時間を示している. そこで, 新しいコンピュータを追加する際に, 吸引ポリシーを定義し、その結果を測ることにした.

7.2 シミュレーション結果

以上の条件の下で, 本節では, 新しいコンピュータの追加という分散システムの変化を持たせた時に, 吸引ポリシーの実行により, コンポーネントの移動側とコンピュータ追加側のコンピュータの CPU とメモリの使用率の変化は図 7 のように示す. 左上 1 と左下 3 はコンポーネントの移動側を表している. 右上 2 と右下 4 は追加されたコンピュータのデータを示す.

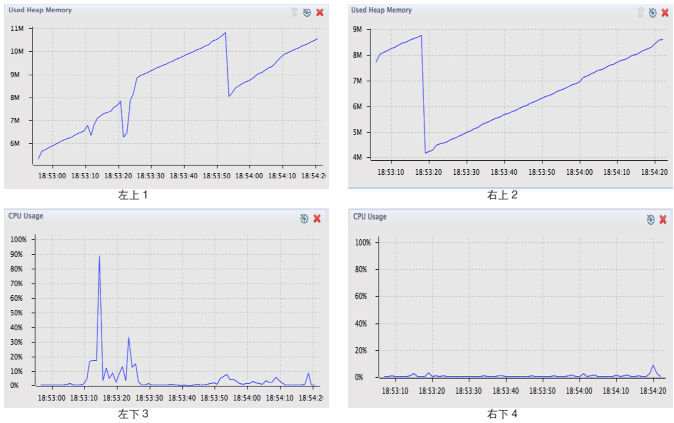

図 7 シミュレーション結果.

左下 3 の図から Time(18:53:10,18:53:15) 秒前後には吸引ポリシーを実行しているため,CPU の上がっている. その後, ソフトウェア・コンポーネントの再配置を行い,Time(18:53:23,18:53:26) 秒の間に移動側のコンピュー

表 1 ソフトウェア・コンポーネントの移動.

Runtime_ID	Policy	Component_ID	Component_Name	Life_cycle	Delay(ms)
136187081056800015012616334999 59	/Policies/AttractionPolicy	dc36fae696d04cd18ff1eab7429606f1	app.Component1	creation	161
136187081056800015012616334999 59	/Policies/SpreadingPolicy	b89ebe96181540259ce8e09a4e858485	app.Component2	creation	201
136187081056800015012616334999 59	/Policies/TimerPolicy	2ea7663f79fa479a8a974222caf353dc	app.Component3	creation	189
...	...	...	...	...	...

タのメモリ使用が下がり,CPU の使用率は 30%前後上がり, 再配置する側のコンピュータのメモリ使用は徐々に上がっていて,CPU の使用率はほんの少し上がっていた.

以上の結果より, ポリシーの実行から, ソフトウェア・コンポーネントの再配置までに, 移動する側のシステムのリソースは大幅に減らせることができ, また CPU の使用率がそこまで上がっていないことから,「ミモザ」システムは優れている信頼性を備えていることがわかる.

8. おわりに

本稿では, 提案したポリシー・ベースでの動的適応できる「ミモザ」ミドルウェアの要件定義, 設計・実装方法及び 3 つの応用事例を用いて「ミモザ」の信頼性評価について概説した. また, 今後「ミモザ」システムの利用上に不可欠なポリシーの記述を一般化するため, 6 つの基本コアとして構成された適応ポリシー言語を提案した. 本提案システムの評価結果では, 多様な変化に対して, 分散システムの信頼性を向上したこととアプリケーションリソースの使用率は小さいのを確認した.

本提案は既存分散システムをベースにしているため, 変化が起きると伴に, 「ミモザ」システムは利用者が事前に記述した五つパターンのポリシーに従い, 変化が起きたコンピュータ上に動いているソフトウェア・コンポーネントを自律的に目的地のコンピュータに再配置させ, 変化を適応する. 従来方式と違い, 本提案システム自身はそれらの変化を柔軟に対応することができる. また, 従来の固定化されたシステムのアーキテクチャそのものも, 変化に応じて, 双方向でシステムアーキテクチャを動的に変更することができる.

今後は, 我々はポリシー記述に持たされたソフトウェア・コンポーネントの衝突と分岐問題を解決するため, 提案したポリシー言語に実行優先度やメタポリシーメカニズムの導入を検討している. また, 本提案手法では, サイバーフィジカルシステムへの応用も試していく予定である.

参考文献

[1] A.〜N and T.〜W, "LEAD: A Language for Dynamically Adaptable Applications," IEICE Transactions on Fundamental of Electronics, Communications and Computer Science, Vol.E81-A N0.6, pp.992-1000.

[2] D.〜N, D.〜N, L.〜E, S.〜M, "The ponder policy specification language," Policies for Distributed Systems and Networks. Springer Berlin Heidelberg, 2001, pp.18〜38.

[3] E.〜Lupu, and M.〜Sloman, "Conflicts in policy-based distributed systems management," , *IEEE Trans. on Software Engineering*, 25.6, 1999, pp.852〜869.

[4] J.〜Zhang and B.〜H.〜Cheng, "Model-based development of dynamically adaptive software," in *ICSE*, 2006, pp. 371–380.

[5] K. 〜J R, and J.〜P R, "Genetic programming II: automatic discovery of reusable programs," Vol. 40. Cambridge, MIT press, 1994.

[6] K.〜N, J.〜S, et al, "PobSAM: policy-based managing of actors in self-adaptive systems." Electronic Notes in Theoretical Computer Science 263 (2010), pp.129〜143.

[7] L.〜ChinHui, and H.〜Q, "On adaptive decision rules and decision parameter adaptation for automatic speech recognition," Proceedings of the IEEE, 88(8), (2000), pp.1241〜1269.

[8] L.〜L, L.〜E, S.〜M, "An adaptive policy based management framework for differentiated services networks," Proceedings of the Third International Workshop on Policies for Distributed Systems and Networks. IEEE, 2002, pp.147〜158.

[9] P.〜Oreizy, N.〜Medvidovic, and R.〜N.〜Taylor "Runtime software adaptation: framework, approaches, and styles," in *ICSE*, 2008, pp.899〜910.

[10] P.〜Zave, and M.〜Jackson, "Four dark corners of requirements engineering," *TOSEM*, 1997, pp.1–30.

[11] R.〜M, T.〜G, S.〜C, "A policy-based mobile agent infrastructure," Applications and the Internet, 2003. Proceedings. 2003 Symposium on. IEEE, 2003. pp. 370〜379.

[12] R.〜N.〜Taylor, N.〜Medvidovic, and P.〜Oreizy , "Architectural styles for runtime software adaptation," in *WICSA/ECSA*, 2009, pp. 171–180.

[13] R.〜S. Kazemzadeh and H.-A. Jacobsen, "Opportunistic multipath forwarding in content-based publish/subscribe overlays," in *Middleware*, 2012, pp. 249〜270.

[14] S.〜J T, S.〜Ichiro,"A Policy-based Middleware for Self-Adaptive Distributed Systems," The Seventh International Conference on Dependability (DEPEND 2014),2014, ISBN: 978-1-61208-378-0, pp.25〜31.

[15] S.〜J T, S.〜Ichiro, "Dynamic Deployment of Software Components for Self-Adaptive Distributed Systems," The Seventh International Conference on Internet and Distributed Computing Systems (IDCS 2014), 2014, LNCS 8729, pp. 149〜203.

[16] S.〜Ichiro, "Self-organizing software components in distributed systems," Architecture of Computing Systems-ARCS 2007, Springer Berlin Heidelberg, 2007, pp.185〜198.

[17] 孫 静涛, 佐藤 一郎, "動的適応可能な分散システムアーキテクチャ," マルチメディア, 分散, 協調とモバイル (Dicomo 2014) シンポジウム, pp.666〜673.

[18] Y.〜S S,K.〜F, et al, "Reconfigurable context-sensitive middleware for pervasive computing," IEEE Pervasive Computing 1(3), 2002, pp.33〜40.