分散型ソフトウェアの新しい構成方式の研究

研究代表者　杉山安洋　日本大学工学部情報工学科

概 要

分散型のソフトウェアを開発する際に，ネットワーク処理をアプリケーションロジックを合わせて開発することは，アプリケーションロジックとネットワーク処理の混同を招き，開発作業を必要以上に複雑にしてしまう．本研究の目的は，分散システムの開発にあたって，アプリケーションロジックとネットワーク処理を分離し，しかもアプリケーションロジックからネットワーク処理が透過的に使用できるようにする方式を確立し，その方式を支援するツールを開発することにある．今年度の研究では，クライアント・サーバ型のシステムの分散化に加えて，ピア・ツー・ピア型やクライアントとサーバが絡み合った複合型のシステムを本手法を用いて分散化できるようになった．

1　研究目的

インターネットの発達にともない，分散型のソフトウェアの重要性が広く認識され，その需要も爆発的に増大している．しかし，分散型のソフトウェアを開発することは，技術的に難しいことも多い．分散型のソフトウェアを開発するためには，そのアプリケーション本来の機能であるアプリケーションロジックに加えて，その機能を複数の計算機に分散させて処理を行うための分散ソフトウェア特有のネットワークに関連した処理を開発しなければならない．このようなネットワーク処理をアプリケーションロジックと合わせて開発することは，アプリケーションロジックとネットワーク処理の混同を招き，開発作業を必要以上に複雑にしてしまう．また，アプリケーションロジックの開発者が，いつもネットワーク処理に精通しているとは限らない．また，既に開発済みのアプリケーションを分散化する必要が発生する場合もある．既存ソフトウェアの変更は新規開発よりも工数がかかる場合も多い．

本研究の目的は，分散システムの開発にあたって，アプリケーションロジックとネットワーク処理を分離して開発できる方式を確立し，その方式を支援するツールを開発することにある．アプリケーションロジックとネットワーク処理とを独立して開発できれば，アプリケーションロジックとネットワーク処理は，それぞれの分野を得意とする別の開発者が担当し効率よく開発できるようになり，また，既存のアプリケーションには外付けでネットワーク処理を追加できるようになる．

研究は，現在分散システムの開発に広く使用されている Java 言語を対象として行っている．また，方法論の研究のみではなく，その方法論を実現するためのツールを合わせて開発し，実用化を目指した研究を行っている．また，開発するツールは，分散ソフトウェアを開発する際に使用するだけでなく，実行時におけるバージョン・構成管理など，開発者やシステム管理者の負担を軽減する機能を含んだ総合的なツールにすることを狙っている．

2　研究成果

本研究では，まずアプリケーションロジックとネットワーク処理を分離して開発することが可能なような分散システムのアーキテクチャを提案し，それに基づいた分散システム開発の手法を提案する．さらに，その分散化手法を実現するためのツールを開発し提供する．

2.1　提案する分散システムのアーキテクチャ

アプリケーションロジックとネットワーク処理を分離して開発する際には，アプリケーションロジックからネットワーク処理が透過的に使用できることが望ましい．アプリケーションロジックからネットワーク処

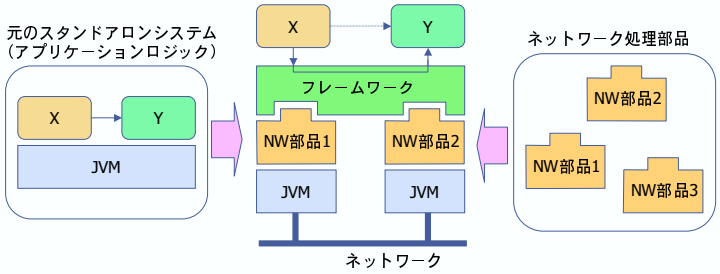


図 1: 提案する分散型システムのアーキテクチャ

理を意識したコーディングをしなければいけないようであれば，両者を分離したとしても，その効果が半減してしまう．しかし，本研究では，ネットワーク処理のすべてを透過的に行ったり，ネットワーク処理を100%自動生成することを狙っているわけではない．

本研究で提案する分散システムのアーキテクチャを図 1 に示す．ネットワーク処理には，アプリケーションによらない共通の処理と，アプリケーションごとに開発が必要な個別の処理がある．そこで，共通な処理はフレームワークとしてまとめ，アプリケーションごとに開発しなくても済むようにし，開発者の負担を軽減させる．また，アプリケーション固有の処理は，フレームワークのホットスポットとして，開発者が追加したり，フレームワークの機能を変更できるようにしている．また，追加するネットワーク処理については，ネットワークスペシャリストが自分で開発できることは勿論であるが，準備された部品の中から選択しても使用することができるようなシステムアーキテクチャとしている．

現在，我々のシステムにおける開発者が機能を追加できるホットスポットは，サーバ側における排他制御などを行うためのスケジューリング機能と，クライアント側でサーバ側で発生した分散処理に関する例外を処理するための，例外処理機能である．スケジューリング機能に関しては，いくつかの典型的なスケジューリングを行うための部品を用意してあり，また，例外処理については，サーバ側の例外をクライアントが処理可能な例外へマッピングすることが可能な部品などを用意している．これらを用いると，開発者は自らネットワーク処理を開発しなくても，その部品をシステムに組み込むだけで分散処理が実現できる．

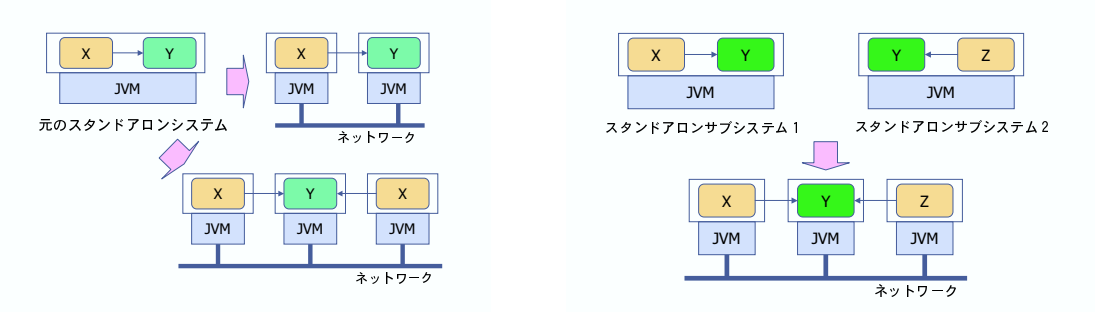
なお，透過型のリモートメソッド呼び出しを謳っているシステムは数多く存在している．例えば，J-Orchestra, JavaParty, NRMI, CentiJ, Addistant などが知られている．しかし，いずれのシステムでも，アプリケーションロジックとネットワーク処理を分離して開発することを狙っているわけではない．また，開発時のみならず，システムの実行時から運用時までを含めたトータルサポートを実現しているシステムも無い．

2.2 分散化の基本手法

スタンドアロンシステムを分散化すると言っても，様々な方式が考えられる．我々の分散システムへの変換の方式は，基本的に 3 つに分類できる．第 1 の方式は，図 2(1) に示すように，スタンドアロンプログラムとして正しく動作するシステムのオブジェクトを分散オブジェクト化，即ち，ネットワーク経由で別の計算機からアクセスできるようにすることにより，もとのシステムと等価な分散システムを開発する方式である．言わば，もともと一体のシステムをネットワークを用いて複数の計算機上に分割して協調動作させる方式である．以下では分離型と呼ぶ．

第 2 の方式は，逆に統合型と呼ぶ．この方式では，図 2(2) に示すように, まず開発対象の分散システムをいくつかのスタンドアロンのサブシステムとして開発する．そして，それらのサブシステム中のオブジェクトを分散オブジェクトとして複数のサブシステムに共有させ，サブシステムが協調動作できるように統合する方式である．

いずれの方式の場合であっても，もとのスタンドアロンのソフトウェアを分散化した際には，そのまま動



(1) 分離型　　(2) 統合型

図 2: 基本的な分散化パターン

作することを期待しているわけではなく，分散化に際し，ネットワーク関連の処理を外付けで追加できる．その際，もとのスタンドアロンソフトウェアには，一切変更は不要である．

例えば，図 2(1) のように，元々はひとつのクライアントからアクセスされていたサーバであっても，分散化によって複数のクライアントから同時にアクセスされることが可能となってしまう．そうなると，サーバがマルチスレッドを考慮して設計されていれば良いが，そうでない場合は資源の競合等が発生し，正しく動作しなくなってしまう．これを防ぐために，排他制御機能を後付けで追加できるようにしている．

第 3 の方式は，分離型と統合型を組み合わせた方式である．複雑なシステムにおいては，ひとつのオブジェクトをネットワーク経由でアクセスできるようにするだけでは不十分で，システム中の複数のオブジェクトを分散化させる必要が出てくる．その際には，各オブジェクトのシステム中での役割や位置づけに応じて，その分散化の方式を決定する．

2.3　開発支援ツール

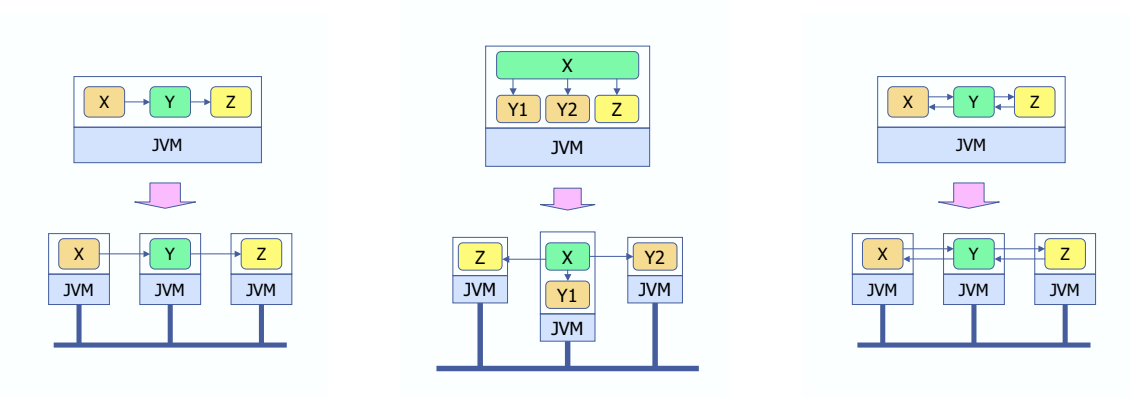
これまでに述べてきた方式によって，既存のアプリケーションシステムを分散化したり，アプリケーションロジックをネットワーク処理から分離しながらシステムを開発することを支援するツールを継続的に開発している．

我々のツールを用いて開発済みのスタンドアロンシステムやサブシステムを分散化するためには，そのシステム中でネットワーク経由でアクセスしたいクラスを指定する．すると，そのクラスのオブジェクトをネットワーク経由でアクセスするためのフレームワークが自動生成され，システム中の他のクラスは一切変更することなく，指定したクラスのインスタンスにネットワーク経由でアクセスすることができるようになる．なお，ツールによって生成される分散処理用のフレームワークであるが，すべてのクラスに共通な汎用のフレームワークではなく，クラス毎にフレームワークを自動生成する方式を採用している．これは，指定したクラスのインスタンスへのネットワーク経由でアクセスを効率良く行うためと，開発者がネットワーク処理を開発する際に，その作業をなるべく減らすために，ツールが自動生成できる処理を増やすためである．

これまでに，我々のツールを用いて分散化することができることが確認されているシステムの構造は以下の通りである．

2.3.1　クライアント・サーバシステム

ひとつのサーバに対して複数のクライアントが存在するという，基本的なクライアント・サーバシステムは，分離型と統合型のどちらでも自動的に分散化できる．また，基本的なクライアント・サーバを複数組み合わせたシステムも分散化できる．例えば，図 3(1) に示すような，あるサーバオブジェクトが他のサーバオブジェクトのクライアントになっている場合なども含む．なお，開発者がサーバとして分散化の対象となるクラスをひとつ指定すると，そのクラスのメソッドの引数や戻り値となるクラスは，自動的に分散化



(1) 複合型　　(2) マルチサーバ型　　(3) ピア・ツー・ピア型

図 3: 分散化可能なシステム構造の例

の対象とすることもできる．しかし，それら以外のクラスについては，開発者が明示的に指定しない限り，分散化の対象にはならない．

2.3.2　マルチサーバシステム

図 3(2) に示すような，ひとつのクライアントが複数のサーバをアクセスしているようなシステムも分散化できるようになったことも，今年度の成果のひとつである．マルチサーバシステムとは言っても，すべてのサーバが異なるクラスのインスタンスの場合は，クライアント・サーバを複数組み合わせたシステムでしか無く，我々のツールを複数回適用すれば分散化ができていた．しかし，今年度の研究成果は，分散オブジェクトの配置を実行時に決定できるようになったことである．これまでは，オブジェクトを配置するホスト計算機は，クラスごとに指定する形式をとっていた．そのため，同一クラスのインスタンスを複数のホスト計算機上に配置することができず，負荷分散などが行えなかった．

今年度の改善により，同一クラスに複数のインスタンスが生成される場合，あらかじめホストとなる計算機を登録しておくと，登録されたホスト計算機の中から実行時に選択した計算機に，そのクラスのインスタンスを生成し配置することができるようになった．これにより，近年用いられることが多くなってきたグリッドコンピューティングと類似した形態の分散システムを開発できるようになった．

2.3.3　ピア・ツー・ピアシステム

単純なクライアント・サーバ型システムでは，クライアントとサーバが相互に参照を持つことは無い．これまで，我々のツールでは相互に参照を持つシステムを分散化することはできなかったが，これを正しく処理できるように改善した点も今年度の成果である．今回実現したものは，オブジェクトが直接的あるいは間接的に相互の参照を持つ場合である．図 3(3) の形は，オブジェクトが相互に参照を持っているピア・ツー・ピア型である．なお，今回，完成した方式では，２つのオブジェクトが相互参照を持っている場合のみに限らず，複数のオブジェクトが巡回的に参照を持っている場合も含んでいる．

2.4　評価と今後の課題

今年度の研究の結果，かなり多くのシステムが我々の手法で分散化できるようになったと考えられる．しかし，これまで検討してきた以外の構造を持つシステムも多く，システムの構造の分析と，その構造に対する分散化の可能性について，さらに検討を重ねる予定である．

この点も考慮し，来年度の予定としては，次の点を考えている．現在の我々のシステムの問題点のひとつに，どのクラスを分散化するかの判断が，基本的には開発者に任されている点がある．システム中には多くのクラスが存在し，それらのうち，どれを分散化するかによって，分散システムとしては全く異なるものになると言っても過言でない．本稿で述べたクライアント・サーバ型やピア・ツー・ピア型といったシ

ステムの構造も，分散化の対象とするクラスを変更すれば，変わってしまう場合も多い．また，システムが整然と設計されていて，どのクラスを分散化すれば良いかが明確に判断できるシステムばかりではなく，クラスが複雑に関連しあい，それらを別のホストに配置することが難しいようなシステムも多く存在する．

従って，ツール側がシステムの構造を解析し，開発者に対して分散化の方式を提案できるような仕組みを実現したいと考えている．そのために，まず，システムの静的あるいは動的な構造表示する機能を開発する予定である．システムの構造を表示することの目的のひとつは，開発者に構造を提示することにより，そのシステムをどの形で分散化させるかを開発者が容易に判断できるようにすることにある．また，分散システムには，いくつかの典型的な形があり，それを判断することも重要である．分散化の対象となるアプリケーションプログラムの形態が，典型的な分散システムの形態と一致する時には，開発者に分散化の形式を提案することもできるようになり，分散化するオブジェクトを判断させる必要がなくなる．さらには，一般の複雑なシステムは，これらの基本的な構造のシステムが組み合わさった形をしている場合が多い．その場合は，システムの構造図をもとに，組み合わされている基本構造を開発者に提示することができれば，開発者の労力を大幅に軽減することができる．

また，分散化後のシステムのクラスの配置の自動化も来年度の大きな課題である．我々のツールで分散化したシステムを実行するためには，開発者が開発したクラスの他に，ツールが自動的に生成したフレームワークのクラスファイルを正しく各ホスト計算機上に配置する必要がある．現状では，クラスファイルの配置は，開発者が手作業で行なっているため，誤りも発生しやすい．そこで，ツール側でクラスファイルの配置を行なう機能の実現を予定している．また，各クラスファイルに複数のバージョンがある場合などは，バージョン間の整合性などの問題も発生し，より一層配置の問題が重要になる．そのため，クラス間のみならず，クラスとフレームワークやネットワーク部品間のバージョン管理を含めた，クラスファイルの配置機能を検討する予定である．

研究成果

研究発表

- 伊藤祥，杉山安洋: “ MIMO：モバイルオブジェクトのためのリモートメソッド呼び出しフレームワーク”，ソフトウェア工学の基礎 XI, 2004.
- 杉山安洋，加藤剛彦: “TRMI におけるマルチスレッド制御機能”, ソフトウェア工学の基礎X, pp.149-160, 2003.
- 杉山安洋: “TRMI によるオブジェクトの分散化”, コンピュータソフトウェア, Vol. 19, No. 5, pp.40-59, 2002.
- 千葉雄一郎，杉山安洋: “TRMI を用いた分散システム構築の自動化”, ソフトウェア工学の基礎 VIII, pp.245-256, 2001.
- 北川俊広，杉山安洋: “ニューラルネットワークを用いたコマンド予測シェルの試作”, 電子情報通信学会　技術研究報告, 2004 年 8 月.
- 梶原直人，杉山安洋: “複数名技術者による同時編集作業における競合の抑制と並列化の両立”, 電子情報通信学会　技術研究報告, 2004 年 8 月.

公開ソフトウェア

- 杉山安洋: TRMI

  URL: http://www.trmi.net

  2005/3 末に公開予定.