分散透過性のネットワークサポートに関する研究

代表研究者　　中　里　秀　則　　早稲田大学　基幹理工学部　教授

1 はじめに

クラウド・エッジコンピューティングなどの分散処理環境では、分散配置された情報処理サービスをネットワーク経由で利用する。分散配置された情報処理サービスを利用するためには、ローカルの情報処理サービスを利用する場合とは異なり、分散を意識し、ネットワークを経由した利用手順が必要になる。このサービスの分散を意識しないようにする「分散透過性」をサポートする仕組みとして、例えばCORBA[4]が提案されている。この仕組みでは、リモートに配置されたサービスにアクセスするために、まずはローカルに存在するアダプタにアクセスし、アダプタからネットワークを経由してリモートのサービスにアクセスし、アダプタの中で分散に関わる処理を行うことによって呼び出し元のアプリケーションからは、利用するサービスがネットワーク経由であることを意識する必要がない。この仕組みでは、呼び出し元のアプリケーションからは分散が透過であっても、アダプタの中では分散を意識した処理を行うとともに、適切なサービスを利用するようにアダプタの設定を行う必要があった。これは、現在分散処理でしばしば用いられるgRPCにおいても同様である。

そこで本研究調査では、ネットワーク側で分散透過性をサポートすることにより、利用するサービスについてアダプタ内で行っていた設定も不要にすることを目標とする。ネットワーク側での分散透過性サポートにより、分散配置されたサービスの利用が容易になるとともに、呼び出されるサービスを動的に適切に設定することが可能となり、計算資源の有効な活用が可能になる。さらに、分散透過性をネットワークによりサポートする仕組みは、コンピュータのコンポーネントをばらばらにし、それらをネットワークで接続することによってコンピュータを作り上げる“disaggregated computing”での計算資源の効率的な活用も可能にする。

分散透過性のサポートでは、プログラムの中で、コンテンツあるいは情報処理サービスへのアクセスがローカルにあるコンテンツ/サービスなのか、あるいはネットワークを経由しなければならないリモートにあるコンテンツ/サービスなのかを区別することなく統一的に指定できなければならない。すなわち、アクセスするコンテンツ/サービスがローカルかリモートかに依らず統一的に指定することができ、またアクセスの手順も同一であることが求められる。手順が同一であることから、コンテンツ/サービスへのアクセスがすべて一度通信ネットワーク機能に引き渡され、指定されたコンテンツ/サービスがローカルなのかリモートなのかをネットワークが判断し、適切な物理資源に向けてパケットを転送するという手順を取ることになる。

この機能を実現するためには、以下の二つの機能が必要となる。

1. ネットワークがコンテンツ/サービスの位置を特定し、サービスの実行に必要なパラメータなどアクセス/サービス実行に必要な情報を含めて要求を転送する機能
2. プログラム処理系が、コンテンツ/サービスへのアクセス情報を直接ネットワークに送る機能

本研究では、ネットワークによるコンテンツ/サービスの動的な割り当てを可能とするために、コンピュータ内と同様に、ネットワーク内においても、コンテンツ/サービスの特定をそれらに結びつけられた「名前」によって特定するコンテンツ指向ネットワーク（Information Centric Networking: ICN）を活用し、上記機能を実現する。なお本研究では、ICNの実装の中でも多くの研究で利用されているNamed Data Networking（NDN）[1]を利用した。

上記1.については、以下の検討を行った。

- ネットワーク内に存在するコンテンツ/サービスに対して、ユーザ毎に隔離された環境でのコンテンツやサービスへのアクセス/処理要求の実現（2節）[2]、
- コンテンツやサービスの登録（3節）[5]、
- 処理要求とともにクライアントとサーバ間でデータを受け渡す方法（4節）[6]。

上記2.については、プログラム言語処理系におけるICNネットワークとのインターフェース方法、およびICNネットワーク側の言語処理系に対するインターフェース部分の機能について検討を行った（5節）[3]。

公益財団法人電気通信普及財団
研究調査助成報告書　第41号　2026年度

2 ユーザ毎に隔離された実行環境の実現方法

ネットワーク側で分散透過性をサポートし、呼び出されるコンテンツ/情報処理サービスの動的設定により計算資源の有効活用を行なえるように、ネットワーク内においても、コンテンツ/サービスを「名前」によって特定し、ネットワーク内に分散配置されたコンテンツやサービスへのアクセスを可能とするために、NDN 上に分散処理環境を構築する。その仕組みとして、Named Data Networking – Function Chain Workflow Plus (NDN-FCW+) が提案されている[2]。NDN-FCW+を用いることによって、NDN によるパケット配送中の名前解決より、指定された関数やコンテンツを使った、ネットワーク内での情報処理が可能となる。

しかし、多くのユーザが同一の情報処理ネットワークを利用した場合、各ユーザプログラムで指定した「名前」の間で競合、すまわち同一の「名前」が使われるということが発生し、正しくプログラムの処理が行なえなくなる。例えばユーザ A が変数 a を使い、ユーザ B も同様に変数 a を定義した場合、この名前によってそのままアクセスできてしまうと、ユーザ A が記録した変数 a の内容をユーザ B が上書きできるようになってしまう。

そこでこの問題を回避するための、ユーザ毎に隔離された分散透過なプログラム実行環境を構築する手法として、名前の接頭句による名前空間制限を提案した。例えば、上記例の場合、ユーザ A の変数 a は/userA/a という名前とし、ユーザ B の変数 a は/userB/a という名前とするのである。この名前空間制限を実現するための仕組みとして、図 2 のような “proxy” によってパケットがもつ名前に接頭句を透過的に追加/削除を行うことによって同一の ICN を複数ユーザで共用する仕組みである「ネットワークコンテナ」(図 1) を提案した[2]。

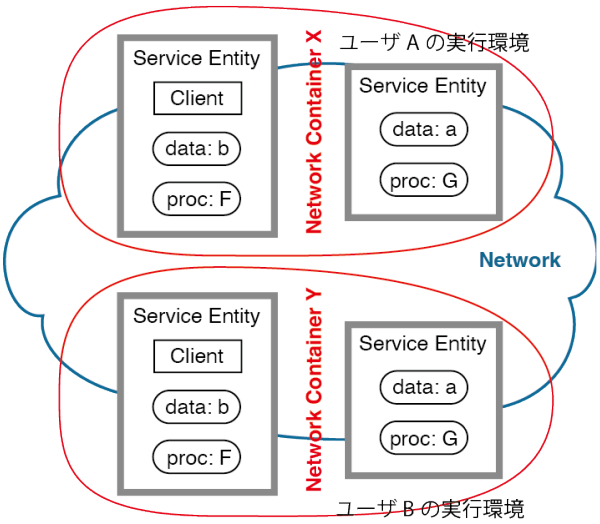

図 1 ネットワークコンテナ

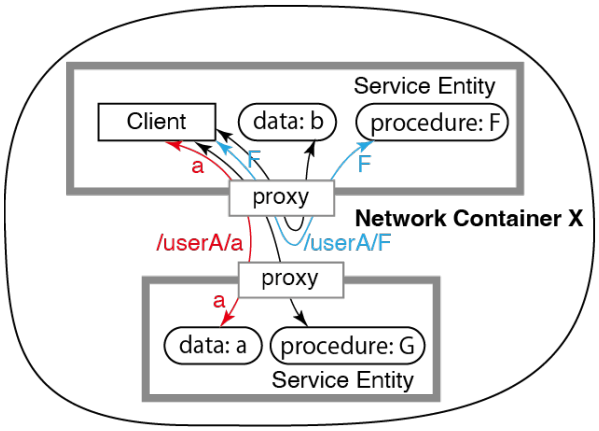

図 2　proxy による名前空間の分離

この提案では、ユーザ毎に隔離された分散透過なプログラム環境として「ネットワークコンテナ」を定義した。ネットワークコンテナは、通信するひとまとまりの「サービスエンティティ」(クライアントおよびサーバ) によって構成される。各サービスエンティティは proxy を経由してメッセージの交換を行うが、proxy によって接頭句の着け外しを行うことによって、サービスエンティティ内では、ユーザが定義した名前によってメッセージ交換が行われ、サービスエンティティ間では、ユーザ定義の名前に接頭句を着けた名前によってメッセージ交換が行われる。この仕組みにより、サービスエンティが移動した場合では、移動への対処はネットワークによって行われ、プログラム処理としてはまったく影響を受けない環境が実現できる。また、メッセージ交換する上でのオーバーヘッドも、他の移動に対して透過な仕組みと比べて、最小に抑えることができる。

3 サービス登録方法

ネットワーク上に分散された形で情報処理を実行するためには、コンテンツやサービスをネットワーク上に配置する必要がある。

現状、分散システムは、コンピュータがネットワークに接続された形で構成されており、各コンピュータ上でサービスエンティティは動作する。一般的には、各コンピュータ上にサービスを動作させるためは、そのコンピュータ上でサービスを起動する仕組みが必要になる。そこで、コンテンツやサービスを配置/起動する仕組みとして「Seed」というサービスを各コンピュータ上に配置することとした[5]。

Seed はリモートからのコンテンツ/サービス配置要求を受け取る窓口として機能する。コンテンツ/サービスの配置を求めるネットワーク内のコンピュータ上にあるサービスから対象のコンピュータの Seed に対して、配置要求を送信する。配置要求を受け取った Seed は配置要求を解析し、自身の属するコンピュータに要求されたコンテンツ/サービスを配置する。

ここでコンテンツ/サービスの配置要求には、コンテンツやサービスの記述を含めたメッセージを送ることが必要になる。NDN での通信のモデルは、要求パケット（Interest パケット）に対してコンテンツを含む応答パケット（Data パケット）が返送される pull 型であり、Interest パケットに大きなコンテンツを入れて送ることを想定していない。そこでデータサイズの小さいコンテンツの配置を要求する場合については、Interest パケットを拡張し、コンテンツの配置要求の Interest パケットにコンテンツのデータが載せられるように変更を加えて要求を行う。一方、データサイズが大きなコンテンツやサービスを Interest パケットに入れて送ることは好ましくない。そのため、サイズの大きなコンテンツやサービス記述については、次節で述べる手法を用いて Seed に送ることとした。

4 大きなデータの受け渡し方法

NDN の通信のモデルでは、Interest パケットの転送経路上のルータは、Interest パケットが通過後、Data パケットが返送されるまで、その返送先を示す情報が Pending Interest Table（PIT）に保持される。すなわち NDN ルータではメッセージ交換の状態が保持される。これは、名前を基にメッセージの配送を行う ICN では、要求元の情報（名前）をネットワーク上に広告すること無く、Data パケットを受け取るために必要な仕組みである。ただし PIT に保持される情報は、Data パケットが返送されてこない場合に対応するため、一定時間経過後に削除される。

一般に、計算処理は、単なるコンテンツの取得に比較して時間がかかるため、時間のかかる計算処理を NDN で行おうとする場合、PIT に保持された情報が削除され、Data パケットの返送ができなくなる可能性がある。そこで、時間のかかる計算処理の場合においても、要求元が計算処理の結果を受け取ることを可能とする仕組みについて検討し、提案を行った。

提案のシステムは、計算処理の要求元であるクライアント、計算処理を実行するサーバ、計算結果を中継するプロキシという 3 つの構成要素からなる。クライアントの名前は一般には経路広告されていないため、サーバからは到達できないが、クライアントとプロキシは同一データリンクに隣接している、あるいは VPN やローカルネットワークに属していて、プロキシがクライアント宛てに送信したパケットは、クライアントの名前によって、クライアントまで配送可能であるように構成する。一方、プロキシの名前は経路広告がなされており、サーバからは到達可能である。

クライアントによる計算処理の要求と

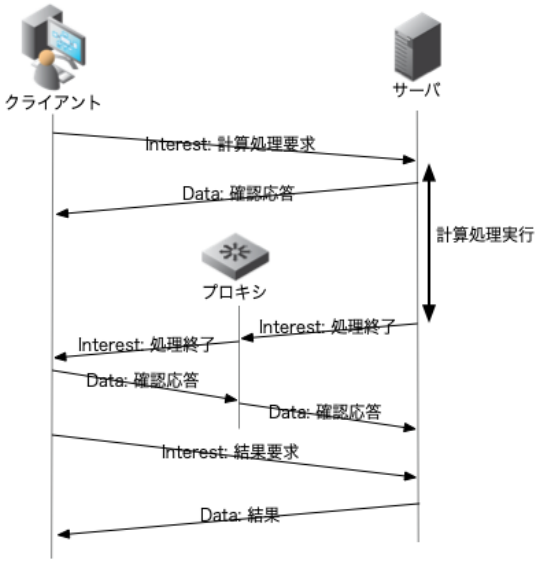

図 3　NDN による分散コンピューティング時の結果受取手順

結果の受取までの流れを図 3 に示す。クライアントが計算処理要求の Interest パケットを送出すると、ネットワークは Interest パケットに含まれる名前を見て、該当するサーバにパケットを配送する。サーバは受け取った Interest パケットに対応する Data パケットを、Interest パケットを受信したことを確認する応答として返送する。この Data パケットは、途中のルータの PIT に保存された情報に基づき、クライアントに送り届けられる。その際、PIT の情報は破棄される。また、クライアントが送った Interest パケットには、計算処理の終了を知らせるための名前（結果返送名）が含まれている。この結果返送名はプロキシの名前の後ろに、クライアントを示す名前および要求した計算処理を示した識別子を付加したものである。例えば、プロキシ名が “/proxy1” 、クライアント名が “/client1”、計算処理の識別子が “/func/call” とすると、”/prox1/client1/func/call” のようなものが結果返送名となる。

サーバは計算処理が終わると計算処理終了を知らせる Interest パケットを上記結果返送名宛てに送信する。この Interest パケットには、計算処理結果を取得するための名前（結果名）が含まれている。Interest パケットはプロキシまで配送される。プロキシでは自身の名前に該当する部分を Interest パケットに記載された名前から取り除き、クライアントに向けて Interest パケットを送信する。Interest パケットはクライアントに向けて転送され、クライアントはそのパケットを受信後、受領確認の応答として、Data パケットを返送する。受領確認の Data パケットを受け取ったプロキシは、名前に自身の名前を付加してサーバに向けて転送する。一方、計算処理終了を知らせる Interest パケットを受け取ると、クライアントは Interest パケットから結果名を取り出し、結果名を名前にもつ Interest パケットを送信する。この Interest パケットはサーバに向けて転送され、受け取ったサーバは計算処理結果をもつ Data パケットをクライアント向けに返送する。

この仕組みにより、PIT タイムアウトにより計算処理結果を受け取れなくなるという問題を解決するとともに、クライアントは自身の名前を経路広告する必要がなく、経路表が膨大になるのを防ぐことができる。

また、ここで述べた手法は、サイズの大きなコンテンツやサービス記述を Seed に送る場合にも利用することができる。この場合、コンテンツ/サービスの配置を求めるサービス（図 4）から対象のコンピュータの Seed に対して、コンテンツ/サービス配置要求の Interest パケットを送信する。この Interest パケットには、コンテンツ/サービス記述を取得するための名前(プロキシの名前にサービス配置クライアントの名前とコンテンツ/サービス記述の名前を付加したもの)が記入されている。Seed はこの Interest パケットの受領確認の応答として Data パケットを返送する。一方、Seed はコンテンツ/サービス記述を取得するための名前を記入した Interest パケットをプロキシに向けて送信する。プロキシはその Interest パケットを受け取ると、名前から自身の名前を削除した名前をもつ Interest パケットを作成し、サービス配置クライアントに向けて Interest パケットを送信する。サービス配置クライアントはその Interest パケットを受信すると、名前に指定されたコンテンツ/サービス記述をもった Data パケットを作成し、返送する。プロキシは Data パケットを受け取ると、その名前に自身の名前を付け、Seed に向けて返送する。このような手順によって、Seed は、サイズの大きなコンテンツ/サービス記述を、サービス配置クライアントから受け取ることができる。

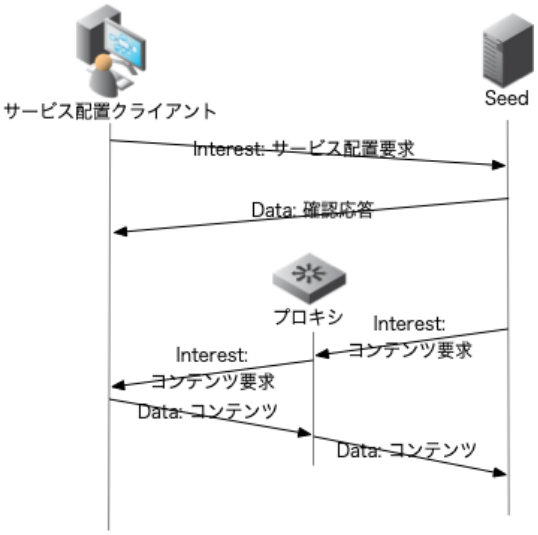

図 4　サービス記述登録の手順

5　言語処理系

ここまで、ICN によってネットワークに分散したコンテンツやサービスを利用して、計算処理を実現する仕組みを検討してきたが、実際に計算処理において分散を意識せずにプログラムを記述するためには、言語処理系をここまで検討してきた NDN のネットワークの仕様に合わせて実現する必要がある。

そこで、Python ベースのプログラム言語処理系を提案し実装を行った。ただし、現段階では分散透過性の

検証のための、最小限の言語仕様となっている。

ここで実装したプログラム言語処理系の文法定義は以下のようになっている。

⟨*program*⟩ ::= ⟨*statement*⟩+
⟨*statement*⟩ ::= let ⟨*identifier*⟩= ⟨*expr*⟩
| print ⟨*expr*⟩
| ⟨*expr*⟩
⟨*expr*⟩ ::= ⟨*expr*⟩ + ⟨*term*⟩
| ⟨*expr*⟩- ⟨*term*⟩
| ⟨*term*⟩
⟨*term*⟩ ::= ⟨*term*⟩ * ⟨*factor*⟩
| ⟨*term*⟩ / ⟨*factor*⟩
| ⟨*factor*⟩
⟨*factor*⟩ ::=- ⟨*factor*⟩
| ⟨*atom*⟩
⟨*atom*⟩ ::= ⟨*string*⟩
| ⟨*number*⟩
| ⟨*identifier*⟩ ((⟨*expr*⟩ (, ⟨*expr*⟩)*)?)
| ⟨*identifier*⟩
| interest ⟨*string*⟩
| interest ⟨*identifier*⟩
| (⟨*expr*⟩)
⟨*identifier*⟩ ::= /[a-zA-Z][a-zA-Z0-9]*/

この言語では、例えば、func(arg1, arg2) というような関数の実行は “let x = interest func(arg1, arg2)” というステートメントにより、func() というサービスや x, arg1, arg2 というコンテンツがネットワーク上のどこにあるかに関わらず func(arg1, arg2) が実行され、その結果が x に保存される。

コンテンツやサービスの分散透過性を実現しつつ、ローカルに存在するコンテンツやサービスへのアクセスはネットワークを経由することを省き、高速化するために、言語処理系内部でローカルに存在するコンテンツやサービスを特定し、内部処理する仕組みとした。内部処理の仕組みを図 5 に示す。

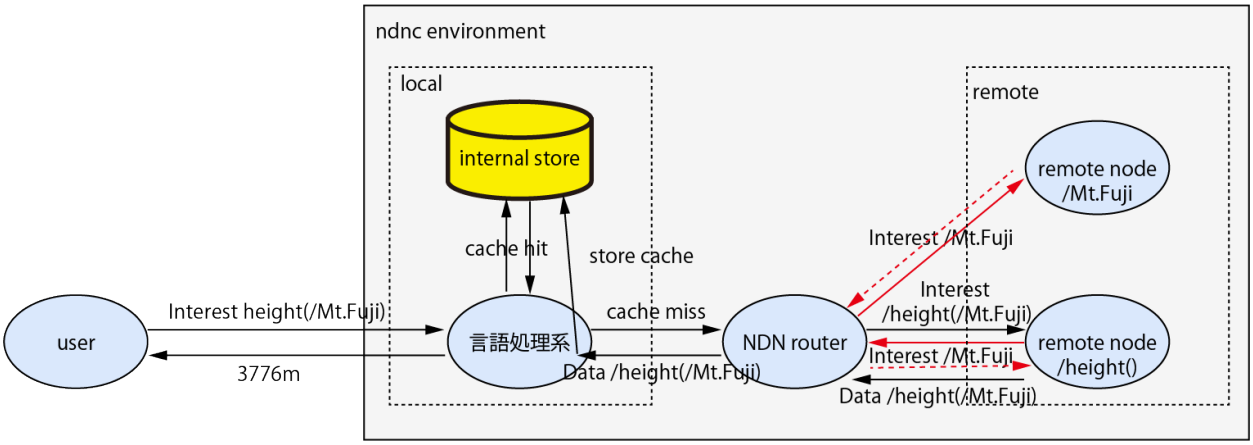

図 5　言語処理系の処理フロー

プログラムのユーザが “let x = interest height(/Mt.Fuji)” を実行すると、言語処理系は “/heigh” という名前のサービスがローカルに存在するかどうか検索し、存在すればそれを利用する。存在しなかった場合は、“/height¥¥(/Mt.Fuji)” を名前にもつ Interest パケットをネットワークへ送信する。ネットワークはこの Interest パケットを “/height” という名前のサービスに向けて転送する。

Interest パケットを受け取ったサービスは、引数である “/Mt.Fuji” というコンテンツを取得するため

に、“/Mt.Fuji”を名前にもつ Interest パケットを送信する。サービス“/height” は、コンテンツ“/Mt.Fuji” を Data パケットによって取得すると、height(Mt.Fuji)を実行し、その結果を Data パケットとしてユーザに返送する。

6 まとめと今後の課題

本研究調査では、ネットワーク側で分散透過性をサポートすることにより、これまで利用するサービス毎のアダプタ内で行っていた設定も不要な完全な分散透過性を実現する仕組みの検討を行った。この仕組みにより、分散処理の利用、分散配置されたサービスの利用が容易になるとともに、呼び出されるサービスを動的に適切に設定することが可能となり、計算資源の有効な活用が可能になる。

これまでのところ個々の仕組みについて研究を行ったが、現状それらをつなげで動作させることができていない。今後これらの仕組みを統合して、全体として動作する仕組みとしていくことが必要となる。

【参考文献】

[1] Named data networking project website. [Online]. Available: https://named-data.net. Last accessed on Nov. 2, 2019.

[2] Hidenori Nakazato and Haruto Kobayashi. Network container: Isolation mechanism for distribution transparent computing. In 2025 23rd Mediterranean Communication and Computer Networking Conference (MedComNet), pages 1–6, June 2025.

[3] Tsuruyoshi Ryu and Hidenori Nakazato. Proposal and implementation of a language processor supporting transparent distributed execution. In 2026 18th International Conference on Future Computer and Communication (ICFCC2026), June 2026.

[4] The Object Management Group. Common object request broker architecture web page. [Online]. Available https://www.omg.org/spec/CORBA/3.4, 2021.

[5] 岩田凌太郎, 中里秀則, and 小林春斗. ICN ベースの分散コンピューティングにおける動的な関数配置機構の提案. 信学技報 vol. 125, no. 415, CS2025-91, pp. 71-76, 電子情報通信学会, 2026.

[6] 池田東生, 小林春斗, and 中里秀則. NDN における分散コンピューティングのためのプロキシを介した非同期プッシュ通知手法の提案. 信学技報 vol. 125, no. 415,CS2025-90, pp. 66-70, IEICE, 2026.

〈発 表 資 料〉

題　　名	掲載誌・学会名等	発表年月
Proposal and implementation of a language processor supporting transparent distributed execution	2026 18th International Conference on Future Computer and Communication (ICFCC2026)	2026 年 6 月
ICN ベースの分散コンピューティングにおける動的な関数配置機構の提案	電子情報通信学会コミュニケーションシステム研究会	2026 年 3 月
NDN における分散コンピューティングのためのプロキシを介した非同期プッシュ通知手法の提案	電子情報通信学会コミュニケーションシステム研究会	2026 年 3 月
Network container: Isolation mechanism for distribution transparent computing	2025 23rd Mediterranean Communication and Computer Networking Conference (MedComNet)	2025 年 6 月