Vol.2016-OS-137 No.5
2016/5/30

分散システム向けの Topology Manager の改良

照屋 のぞみ[†1]　河 野 真 治[†2]

Alice では、スケーラブルな分散プログラムを信頼性高く記述できる環境を実現するために、Computation と MetaComputation による階層化を採用している。分散環境の構築等の複雑な処理を Alice が Meta Computation として提供することで、仕様変更を抑え、変更前の信頼性を保ったまま拡張可能にする。本研究では、分散トポロジー管理の Meta Computation である Topology Manager に NAT 越えを実現するための設計を行う。そしてその設計が Alice アプリケーション同士の接続も可能にすることを示す。

Improvement of Topology Manager for distributed system

Nozomi TERUYA[†1] and Shinji KONO [†2]

1. 分散アプリの Topology Manager

当研究室ではデータを Data Segment、タスクを Code Segment という単位で記述する分散フレームワーク Alice[1)] の開発を行っている。Alice ではスケーラブルな分散プログラムを信頼性高く記述できる環境を実現する。ここで言う信頼性とは、定められた環境下で安定して仕様に従った動作を行うことを指す。

Alice では、処理を Computation と Meta Computation に階層化し、コアな仕様と複雑な例外処理に分離する。そして分散環境の構築に必要な処理を Meta Computation として提供する。プログラマはコアな仕様の変更を抑えつつプログラムの挙動変更ができるため、信頼性の高い分散アプリケーションの記述が可能となる。

分散アプリケーションでは複数のノードを管理する必要がある。接続する分散アプリケーションを見つけ、それらを接続し合ってトポロジーを構成する。その際には NAT などネットワークの問題を考慮しなければならない。そして、ノードの障害発生時にはトポロジーの再構成などの対応を用意しなければならない。また、別々の分散アプリケーション同士を接続・連携させる拡張をしたい場合もある。このように分散アプリケーションにおいてノードのトポロジー管理は重要である。しかし、プログラマがそれを全て記述するのは困難である。

Alice の Meta Computation のひとつである Topology Manager は、アプリケーション外部からトポロジーの構成・管理をサポートする。本研究では、分散アプリケーションにおける課題である NAT 越えなどの機能を TopologyManager で実現するための設計を行う。同時に AliceVNC や AliceChat といった Alice 上でのアプリケーションを連携するための設計を行うことで、相互干渉なく容易にアプリケーションの接続・拡張ができる環境の提供を目指す。今までは 1 つの分散アプリケーションに対して 1 つの Topology Manager を用いていたが、Topology Manager を複数用意しそれぞれに LAN 内と WAN 内のトポロジーを管理させることで、NAT 越えや別の分散アプリケーションの連携が容易にできることが期待される。

†1 琉球大学理工学研究科情報工学専攻
Interdisciplinary Information Engineering, Graduate School of Engineering and Science, University of the Ryukyus.
†2 琉球大学工学部情報工学科
Information Engineering, University of the Ryukyus.

2. 分散フレームワーク Alice の概要

[Data Segment と Code Segment]

Alice では Code Segment（以下 CS）と Data Segment（以下 DS）の依存関係を記述することでプログラミングを行う。CS は実行に必要な DS が全て揃うと実行される。CS を実行するために必要な入力される DS のことを InputDS、CS が計算を行った後に出力される DS のことを Output DS と呼ぶ。データの依存関係にない CS は並列実行が可能である（図 1 ）。CS の実行において DS が他の CS から変更を受けることはない。そのため Alice ではデータが他から変更され整合性がとれなくなることはない。

Alice は Java で実装されており、DS は Java Object に相当する。プログラマが CS を記述する際は、

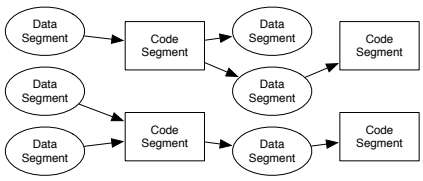

図 1　CodeSegment の依存関係

CodeSegment クラスを継承する。

[**Data Segment Manager**]

DS は数値や文字列などの基本的なデータの集まりを指し、Alice が内部にもつデータベースによって管理されている。このデータベースを Alice では DS Manager(以下 DSM) と呼ぶ。CS は複数の DSM を持っている。DS Manager には対になる String 型の key が存在し、それぞれの Manager に key を指定して DS にアクセスする。DSM には Local DSM と Remote DSM が存在する。Local DSM は各ノード固有のデータベースである。Remote DSM は他ノードの Local DSM に対応する proxy であり、接続しているノードの数だけ存在する（図 2 ）。他ノードの Local DSM に書き込みたい場合は Remote DSM に対して書き込めば良い。

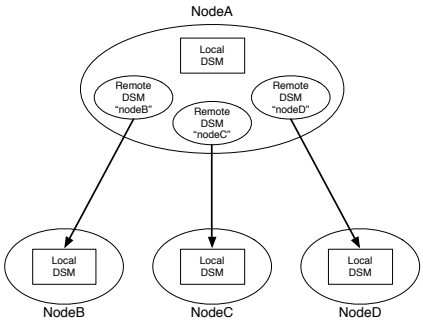

図 2　Remote DSM は他のノードの Local DSM の proxy

[**Computation と Meta Computation**]

Alice では、計算の本質的な処理を Computation、Computation とは直接関係ないが別のレベルでそれを支える処理を Meta Computation として分けて考える。Alice の Computation は、key により DS を待ち合わせ、DS が揃った CS を並列に実行する処理と捉えられる。それに対して、Alice の Meta Computation は、Remote ノードとの通信時のトポロジーの構成やデータの表現形式の選択の処理と言える。つまりこれらの処理は Alice の Computation を支えている Computation とみなすことができる。

Alice の機能を追加するということはプログラマ側が記述する Computation を支えるための Meta Computation を追加することと言い換えられる。Alice では Meta Computation として分散環境の構築等の機能を提供するため、プログラマは CS を記述する際にトポロジー構成や切断、再接続という状況を予め想定した処理にする必要はない。プログラマは目的の処理だけ記述し、切断や再接続が起こった場合の処理を Meta Computation として指定する。このようにプログラムすることで、通常処理と例外処理を分離することができるため、仕様の変更を抑えたシンプルなプログラムを記述できる。

[**Topology Manager**]

Alice では、ノード間の接続管理やトポロジーの構成管理を、Topology Manager という Meta Computation が提供している。この Topology Manager も CS/DS を用いて実装されている。

静的トポロジー

プログラマはトポロジーファイルを用意し、Topology Manager に読み込ませるだけでトポロジーを構成することができる。トポロジーファイルは DOT Language[2] という言語で記述される。DOT Language とは、プレーンテキストを用いてデータ構造としてのグラフを表現するためのデータ記述言語の一つである。ソースコード 1 は 3 台のノードでリングトポロジーを組むときのトポロジーファイルの例である。

```
digraph test{
    node0 −> node1[label=”right”]
    node0 −> node2[label=”left”]
    node1 −> node2[label=”right”]
    node1 −> node0[label=”left”]
    node2 −> node0[label=”right”]
    node2 −> node1[label=”left”]
}
```

Code 1　トポロジーファイルの例

また、DOT Language ファイルは dot コマンドを用いてグラフの画像ファイルを生成することができる。そのため、記述したトポロジーが正しいか可視化することが可能である。

Topology Manager はトポロジーファイルを読み込み、参加を表明したクライアント（以下、Topology Node）に接続するべきクライアントの IP アドレスやポート番号、接続名を送る（図 3)。

トポロジーファイルで lavel として指定した名前は Remote DSM の名前として Topology Node に渡される。そのため、Topology Node は Topology Manager の IP アドレスさえ知っていれば自分の接続すべきノードのデータを受け取り、ノード間での正しい接続を実現できる。

動的トポロジー

実際の分散アプリケーションでは参加するノードの数が予め決まっているとは限らない。そのため Topol-

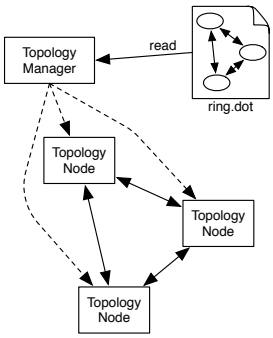

図 3　Topology Manager が記述に従いトポロジーを構成

ogy Manager は動的トポロジーにも対応している。トポロジーの種類を選択して Topology Manager を立ち上げれば、あとは新しい Topology Node が参加表明するたびに、Topology Manager から Topology Node に対して接続すべき Topology Node の情報が渡され接続処理が順次行われる。そして Topology Manager が持つトポロジー情報が更新される。現在 Topology Manager では動的なトポロジータイプとして Binary Tree と Star に対応している。

障害発生時の対応

ノード間接続が切れた場合、次の通信が行われるまで切断を発見することができない。また、接続状態ではあるが応答に時間がかかる場合もある。これらの問題を検知するために、KeepAlive という定期的に heart beat を送信しノードの生存確認を行う Meta Computation が用意されている。一定時間内にノードからの応答がない場合、そのノードの Remote DSM が切断され、再接続すべきノード情報を要求する。

また、各ノードに切断・再接続時に対する処理を特別に用意したい場合がある。そのために、切断・再接続を検知した際に任意の CS を実行できる Meta Computation もある。プログラマは切断の際に実行したい CS を書き、Meta Computation に指定しておくだけで良い。

これらの Meta Computation は Topology Manager にも含まれているため、Topology Manager を用いることで構成したノード間の接続が途切れてもトポロジーを再構成することができる。

3. TreeVNC の NAT 越え

TreeVNC とは、当研究室で開発を行っている授業向け画面共有システムである[3]。オープンソースの VNC である TightVNC[4] をもとに作られている。授業で VNC を使う場合、1 つのコンピュータに多人数が同時につながるため、性能が大幅に落ちてしまう。この問題をノード同士を接続させ、木構造を構成することで負荷分散を行い解決したものが TreeVNC である (図 4)。

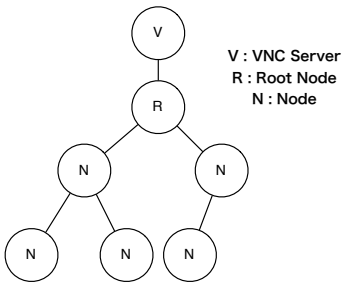

図 4　TreeVNC の構造

TreeVNC は授業向けのシステムであるため、プライベートネットワーク内のみでの使用を前提に作られている。しかし学外から授業に参加したい場合、教室にカメラを設置するだけではスクリーンに写した教員の PC 画面までは見ることは困難であるため、学外のノードからでも画面配信の Tree に入りたい要求が生まれた。つまり、NAT を越えた通信に対応する必要がある。そのために、TreeVNC では画面配信側ネットワークがグローバル IP アドレスを持っていることを前提とし、別ネットワーク上のノードが画面配信側ルートノードの IP アドレスを指定して直下の子になる Direct Connection を実装した (図 5)。

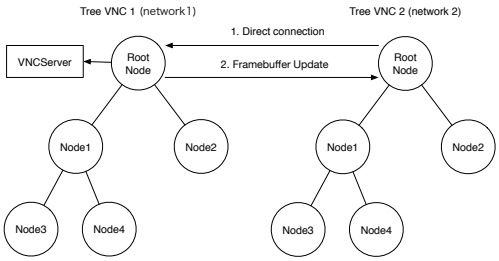

図 5　TreeVNC の Direct Connection

しかし、この方法だと複数の別ネットワークからの接続があった場合、ルートノードに大量に子が接続されてしまうためルートノードに接続台数分の負荷がかかってしまう。

また、別ネットワーク側のノードが途中で画面を配信したい場合（Server Change Request）がある。TreeVNC ではソケットの反転が考案されたが、ソースコードが膨大で拡張した場合どこに影響するかわからないほど複雑であったため、実装までに至らなかった。

さらに、どちらのノードもプライベートネットワークであった場合、TreeVNC では NAT 越えのための中間サーバをプログラマが作らなければならない。

このように、NAT 越えは分散アプリケーション構築における課題の 1 つでもあるが、その実装は容易ではない。Alice の Topology Manager にも NAT 越えをサポートする機能が必要であると考えた。

4. AliceVNC と AliceChat の接続

Alice が実用的なアプリケーションを実装するのに充分な性能があるかをテストする例題として、Alice 上に TreeVNC と Star トポロジーの Chat が実装された。それぞれを AliceVNC、AliceChat と呼ぶ。これらは全く別のアプリケーションであるが、お互いに接続させたい要求がでてきた。例えば、AliceChat 上に AliceVNC の画面のスナップショットを載せたい場合や、AliceVNC 上に AliceChat の内容をコメントとして画面に流したい場合である。

このように別トポロジーのアプリケーション間で相互干渉なく接続するための機能が必要であると考えた。

5. Topology Manager の拡張設計

[別トポロジー間での接続]

AliceVNC と AliceChat のように同一ネットワーク内での別アプリケーションの接続を実現する仕組みが図 6 である。

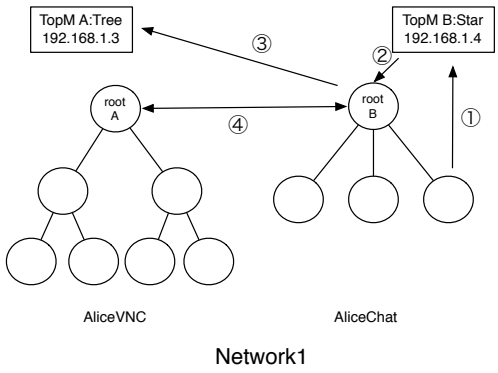

図 6　プライベートネットワーク内での接続

(1)　接続を要求する側のいずれかの Node が接続先 Topology Manager(A) の IP アドレスを自身を管理する Topology Manager(B) の DSM に保存
(2)　Topology Manager(B) は RootNode(B) に Topology Manager(A) への接続をするよう要求
(3)　RootNode(B) が Topology Manager(A) と接続し、自身の接続先ノードの情報を取得
(4)　取得した情報をもとに RootNode(A) に接続

これで AliceChat 側に AliceVNC のスナップショット情報を送ることができる。

また、(1) の手順を踏むことで AliceChat のトポロジーの再構成時に AliceVNC へ再接続も自動で行うことができる。TopologyManager は Node 間の接続が切れるとトポロジーを再構成するため、RootNode(B) が落ちると、それを検知した Topology Manager(B) が他のノードを RootNode として配置し接続をやり直す。どのノードが落ちても Topology Manager(B) が接続先 Topology Manager(A) の情報を保持したままなので、再び (2) 以降の手順で AliceVNC の接続が行われる。

今までの Alice では、ノードに対して Topology Manager は 1 つと決められていた。Topology Manager と各ノードのやり取りをするのは、ノードごとに実行される Topology Node という Meta Computation である。Topology Manager は接続された node の情報 (nodeName と IP アドレスの HashMap) を”nodeTable” という Key に対応する DS として保存している。そして Topology Node は Topology Manager から割り当てられた nodeName を”hostname” という Key に保存する。つまり、接続する Topology Manager が増えれば TopoloyNode に割り当てられる nodeName も増えるため、今までのように”hostname”という 1 つの Key だけでは対応できない。TopologyNode が複数の TopologyManager に対応できるようにしなければならない。

そこで、Meta Computation として、通常の Local DSM とは別に Topology Manager ごとの Local DSM を立ち上げる方法が考えられる (図 7)。

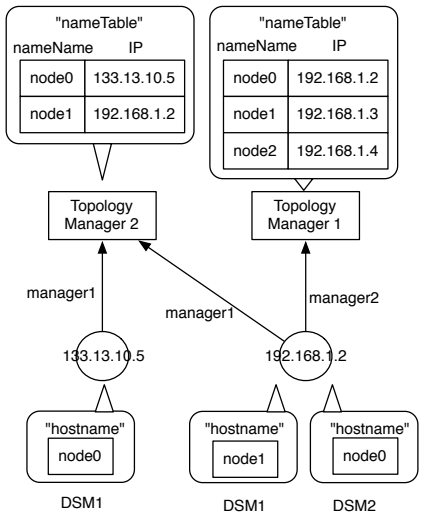

図 7　Topology Node は複数の nodeName を持つ

それぞれの Topology Manager に対応する DSM を作り、そこにそれぞれの nodeName を格納することで、DSM を切り替えるだけで TopologyNode の仕様は変えずに複数の Topology Manager に対応できる。

しかし、現在の Alice のコードでは DSM を管理する class が static class であったため、複数の Local DSM を持つことができない。static を取り除くためには Alice の大部分のコードを修正する必要がある。そのため、現状では Key である"hostname"のあとに Topology Manager ごとの番号を付け加えることで、Key によって Topology Manager ごとの対応を分けている。Alice の再設計を行う際には static class のない実装を行い、DSM 切り替えによる方式を実現したい。

[別ネットワーク間での接続]

TreeVNC でお互いにプライベートネットワークのノードの接続をするには、NAT 越えのための中継プログラムをプログラマが書かなければならなかった。しかし、Alice ではトポロジー管理をアプリケーションから分離しているため、グローバル IP アドレスを持った Topology Manager(以下、Global Topology Manager) を立てるだけで良い。プライベートネットワークの Topology Manager（以下、Private Topology Manager）はプライベートネットワーク内で木を構成し、Global Topology Manager は各ネットワークの RootNode で木を構成する。つまり、3 次元的な木構造が構成される。そのため、複数の Topology Manager を立ち上げるだけで、Topology Manager 自体の「参加表明のあったノードを木構造」に接続するという仕様に変更はない。

NAT 越えは Topology Manager の「参加表明したノードでトポロジーを構成する」Computation を支える Computation、つまり Meta Meta Computation と言える。NAT 越えのため以下の機能を Topology Manager/Node の Meta Meta Computation として取り入れる。

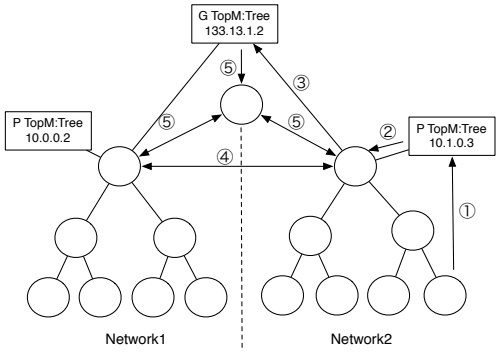

図 8　NAT を越えた接続

(0) 接続を受け入れる側 (Network1) のルートノードがグローバル IP アドレスを持った Global Topology Manager を立ちあげておく
(1) 接続を要求する側 (Network2) のいずれかの Node が Global Topology Manager の IP アドレスを自身を管理する Topology Manager の DSM に保存
(2) Topology Manager は RootNode に Global Topology Manager への接続をするよう要求
(3) RootNode が Grobal Topology Manager と接続し、自身の IP アドレスを送る。Global Topology Manager が受け取った IP アドレスがプライベートアドレスであれば、ノードに対して NAT の外側 IP アドレス/ポート番号を要求されるので、RootNode はそれに返答。
(4) UDP hole punching が行われ、Network1 の RootNode と Network2 の RootNode が接続される
(5) もし接続が確立されなければ、Global Topology Manager がデータ中継用の CS を用意しデータを中継する

Meta Meta Computation が NAT 越えをサポートするため、Topology Manager も Topology Node も接続要求のあったノードがグローバルかプライベートかを気にせず扱うことができる。

6. 他言語等との比較

[Erlang]

並列指向プログラミング言語 Erlang[5] は、プロセスと呼ばれる id 付きの独立したタスクに対して、データをメッセージでやりとりする。タスクをプロセスという細かい単位に分割して並列に動かす点や、メモリロックの仕組みを必要としない点は Alice と同様である。

しかし Erlang では分散環境の構築等はプログラマ自身が記述しなければならない。Alice では分散環境の構築は Topology Manager が一括して管理するため、プログラマはトポロジーを指定するだけで良い。

Erlang には NAT 越えをサポートするためのライブラリが存在し[6]、NAT 機器と通信し外側 IP アドレスを取得メソッドや、NAT に指定したポート番号でポートマッピングするメソッドが用意されている。

Alice では、それらの機能も Topology Node の一部に含めることで、NAT 越えを意識しなくとも Topology Node を用いるだけでサポートできるようにする。

[Akka]

Akka[7] は Scala・Java 向けの並列分散処理フレームワークである。アクターモデルを採用しており、アクターと呼ばれるアドレスを持ったタスクに、データをメッセージでやりとりする点が Erlang と似ている。

Akka の特徴として、メッセージを送りたいプロセスのアドレスを知っていればアクターがどのマシン上にあるかを意識せずにプログラミングできるという点がある。逆に Alice はどの Remote DSM に対してやり取りをするかを考慮するが、CS が Output した DS を次にどの CS に渡すかを意識する必要がない。この点はアクターモデルと CS/DS モデルのパラダイムの違いと言える。

また、Akka のもう一つの特徴として、アクターで親子関係を構成できる点がある。分散通信部分を子アクターに分離し、親アクターは子アクターの Exception が発生した時に再起動や終了といった処理を指定できる。さらに Router という子アクターへのメッセージの流れを制御するアクターや、Dispatcher というアクターへのスレッドの割当を管理する機能を Akka が提供している。このように処理を階層化し複雑な処理をフレームワーク側が提供する仕組みは Alice の Meta Computation と共通している。

Akka では NAT に対応するために、外側 IP アドレスとポート番号を指定することができる。しかし NAT 機器へのポートマッピングはプログラマが記述しなければならない。

7. ま と め

並列分散フレームワーク Alice では、スケーラブルかつ信頼性の高いプログラムを記述する環境を実現するため、CS/DS の計算モデルと Meta Computation による実装の階層化を採用している。NAT を越えたノード間通信及び Alice 上のアプリケーション間接続を実現するために、分散トポロジーの構成・管理をする Meta Computation である Topology Manager/Node の拡張設計を行った。DSM の切り替えにより Topology Node を複数の Topology Manager に対応させ、さらに Meta Meta Computation として NAT 越えの機能を追加することで、Topology Manager/Node のコードを大きく変えずに別トポロジー・別ネットワーク間のノードの接続が行われ、自由度の高い通信が可能になると期待される。しかし、それを実装するには Alice の DSM を管理する static class の static を取り除かなければならず、それは容易ではなかった。今後の課題としては、Alice そのものの再設計を行った後にこれらの機能を実装し、NAT を越えたアプリケーションの接続が実際にできるかをテストする必要がある。

参 考 文 献

1) Nozomi Teruya and Shinji KONO: 分散フレームワーク Alice の PC 画面配信システムへの応用, 第 56 回プログラミング・シンポジウム (2015).
2) : Dot Language, http://www.graphviz.org/.
3) Tatsuki IHA and Shinji KONO: 有線 LAN 上の PC 画面配信システム TreeVNC の改良, 第 56 回プログラミング・シンポジウム (2015).
4) : TightVNC Software, http://www.tightvnc.com.
5) : Erlang, http://www.erlang.org/.
6) : erlang-nat, https://github.com/benoitc/erlang-nat.
7) : Akka, http://akka.io/docs/.