Vol.2010-DPS-145 No.21
Vol.2010-GN-77 No.21
Vol.2010-EIP-50 No.21
2010/11/26

ネットワーク分散システムのための抽象ネットワークアプローチ

中 村 和 敬[†1]　日 比 野 靖[†1]

筆者らはネットワーク分散システムの設計方針として、既存資源を最大限活用することにより新しい通信プロトコルやドライバソフトウェアの開発の必要性をなくし、レイヤを重ねた際のオーバヘッドを低減させる、“抽象ネットワークアプローチ” を考案し、そのプロトタイプ実装を行った。本稿では抽象ネットワークアプローチの概要とその実装の一つである “Abstract Network” について解説する。

Conceptual Network Approach for Network Distributed Systems

Kazutaka NAKAMURA [†1] and Yasushi HIBINO[†1]

We had devised a design policy of network distributed system construction, and named it “conceptual network approach”, that does not require any new communication protocols and driver software, and reduces overheads in multi-layer environment. In this paper, the approach itself and “Abstract Network”, a prototype implementation of this approach, is described.

1. はじめに

1.1 背　景

計算機システムやネットワークは、一般的にはインターフェースが異なる様々なハードウェアから構成されている。これらの機器を統一的に利用するための手法として様々な方式が提案されてきた。これらの既存方式のアプローチは、一般的に新たなシステムの上位層を定義し、様々なインターフェースをその新しい上位層のインターフェースに適合させるためのドライバソフトウェアを開発することにより、新たな上位層の上での統一的な利用を可能にしようとするものであった[2)]。しかしながらこの方法は、様々なハートウェアに対し、一つ一つドライバソフトを開発する必要がある。また、これらのドライバソフトは、個別のコンピュータ上で動作する必要があり、ネットワークを介しての利用には、さらに上位の階層であるネットワークプロトコル階層を介する必要がある。既存のネットワーク分散システムフレームワークの多くは、前者の問題を既存のオペレーティングシステムを利用することによって解決している。これはハードウェア統合の上手い方法である。しかし、同時にこれは階層を重ねるということであり、ネットワーク分散システムフレームワークのもとでネットワークを介さずに資源を利用しようとする場合には、後者の上位プロトコル階層を通す必要があるため、オーバーヘッドが発生してしまう。この問題はネットワーク分散環境でない用途の場合には既存 OS から直接利用するこによって解決できるが、これはネットワーク分散システムを含めたシステム全体に統一的なインターフェースを与えるという当初の目的に反している。

このような理由で、既存のアプローチによるネットワーク分散システムの開発では階層が積み重なることは避けられないと考えられる。と言うのもネットワーク分散環境において新しいサービスを構築する際には、ネットワーク上に存在する様々なサービスを集め結合して新たなサービスを提供することになるが、この時統合するための上位の層を利用することとなるからである。ネットワーク分散システムが社会に浸透することを考えると、これらのネットワーク分散システムフレームワークは、階層 (レイヤ) の多重化のオーバーヘッドに対応したものでなくてはならない。そしてもしこのことが実現したネットワーク分散システムフレームワークであるならば、ローカルな資源利用には、このフレームワークを適用することにより、階層を重ねるオーバーヘッドを最小限に抑えることが可能となるはずである。このようなネットワーク分散システムフレームによれば、スタンアロン環境から、ネットワーク分散環境まで、システム全体を通じて、統一的なインタフェースを与えることができるはずである。

1.2 抽象ネットワークアプローチ

筆者らはこういった予測の元に、開発するドライバソフトウェアを最低限に抑え、かつレイヤ多重化を行ってもオーバーヘッドを抑えることが可能な抽象ネットワークによる手法を考案した既存の通信資源は既に十分なプロトコルとソフトウェアを備えており、通信に必要

†1 北陸先端科学技術大学院大学
Japan Advanced Institute of Science and Technology

Vol.2010-DPS-145 No.21
Vol.2010-GN-77 No.21
Vol.2010-EIP-50 No.21
2010/11/26

な機能を十分に備えている。したがって既存の通信資源を的確に活用するだけで新たにプロトコルやソフトウェアを開発せずとも異なる通信資源を利用する機能資源同士の接続を実現できるはず、すなわちネットワーク分散システムを実現できるはずである。

本稿で提案する手法ではまずネットワーク上の利用可能な資源を、通信機能を提供する通信資源と、様々な計算機能を提供する計算資源に分類し、前者を Link 、後者を Node とする、ネットワークグラフによって管理する。これを “Base Netowrk”(以下 BaseNet と省略) と呼ぶ。ネットワークグラフの形式をとっているのは、どの資源がどの資源を利用出来るかを扱う為である。個々の計算資源は各々勝手なプロトコルによって通信を行う。ドライバプログラムはこの BaseNet 上では中継を行う計算資源として扱われ、あたかも通常のネットワークのスイッチであるかのように抽象化する事ができる。

この BaseNet をデータリンク層に用いて、抽象的なネットワーク層として設けられるのが “Abstract Network”(以下 AbNet と省略) である。これはシステムをネットワークとして抽象化するという事であり、筆者らはこれをネットワーク抽象と呼ぶ。AbNet が通常のネットワークと大きく異なる点は、通信の端点は抽象的なネットワーク層のプロトコルではなく実際に採用している、 BaseNet のプロトコルによって通信を行う点である。この様な事が可能なのは、既存の計算機システムやネットワークの階層化方式が毎回全ての階層を上り下りする、いわばインタープリタ的な方式であるのに対し、抽象ネットワークでは全てを下位層に写像してしまうコンパイラ的な方式をとるからである。この事は同時に、 AbNet が事前に全ての経路を決定してしまう、コネクション指向ルーティングを行うという事を意味し、この時に最適な経路を選択することによって、不必要なオーバーヘッドの発生を最低限に抑える事ができる。

またレイヤ多重化によるオーバーヘッドの発生を抑える為にシステムの構成は、計算結果を直ちにその結果を必要とする計算資源に送ることの出来るデータフロー方式によらなければならない。なぜなら既存システムで一般的なクライアントサーバ方式では返り値は常に呼び出し元に戻らなければならず通信回数が増えてしまうので、複雑な分散アプリケーションを多重化する際には大きなオーバーヘッドとなるからである。またこの様な形式はネットワーク抽象と相性がよく、同時に計算資源の状態を減らす事は分散アプリケーションの構築を容易にする。

以上のような抽象的なネットワークとしてシステムをとらえ、統一的なインターフェースを与えようとするのが抽象ネットワークアプローチである。本手法はネットワーク抽象によって利用可能な資源の結合関係に従ってプロトコルの変換を行うので、最低限のドライバソフトウェア、すなわちプロトコル変換ソフトウェアのみ開発すれば動作させられる。また全ての AbNet の結合は BaseNet の結合に写像されるので、仮に AbNet を BaseNet として利用してさらに AbNet を構築しても、最終的に全て基底の BaseNet の結合に変換されるので不必要なオーバーヘッドは全く発生しない。

筆者らは抽象ネットワークの一つの実装として、Abstract Network を定義しこれを Common Lisp の実装の一つ、UNIX OS 上で Common Lisp を用いてプロトタイプを実装し、実験環境を用意して、この上でデータフロー言語 SDFL によって定義されたデータフロー関数が AbNet 上に正しく展開され、目的の機能を実現できる事を確認した。本稿では Abstract Network の概要について述べ、続いてその詳細をみていく。

2. Abstract Network とは

Abstract Network を利用する場合、まず データフローグラフ (以下 DFG: Data Flow Graph) の形でネットワークアプリケーションプログラムを記述する[3]。DFG は関数 Node と関数の引数を結ぶ Ark で書き表される。

次にこれらの Node と Arg は Node と Node を結ぶ Link から成る AbNet に対応付けられる。AbNet Node は一つの Function を保有している。DFG 上の関数 Node は AbNet Node に対応付けられ、DFG 上の Ark は AbNet Link 上を通る、AbNet Connection に対応付けられる。このように、ネットワークアプリケーションプログラムは AbNet Connection の集合に変換される。

さらに AbNet Connection の集合は、BaseNet 上の資源に対応付けられる。BaseNet Node はいくつかの Function と Link との接点である Port を含み、Node 中の Function は入出力 Argument を持っている。Argument は Port に結びつける事が出来、これを “Port Assignment” と呼ぶ。AbNet Connection は BaseNet 上の Port Assignment の集合に対応付けられる。

最後に、 Port Assignment 情報は、あらかじめ用意された Signaling Network を通して BaseNet 上の Setup Function に送られ、Setup が対応する実際の処理を行う。この様にして DFG で記述されたネットワークアプリケーションが、BaseNet 上で実現される。この DFG から BaseNet の変換過程は図 1 に示される。

抽象ネットワークによるアプローチを採用すると、ユーザは詳細な通信方式を知る事無く、抽象ネットワーク上で自在にリソースを結合することが出来る。そうでありながら、抽象ネットワーク上での結合は不必要なオーバーヘッドが発生しない。なぜなら抽象ネット

Vol.2010-DPS-145 No.21
Vol.2010-GN-77 No.21
Vol.2010-EIP-50 No.21
2010/11/26

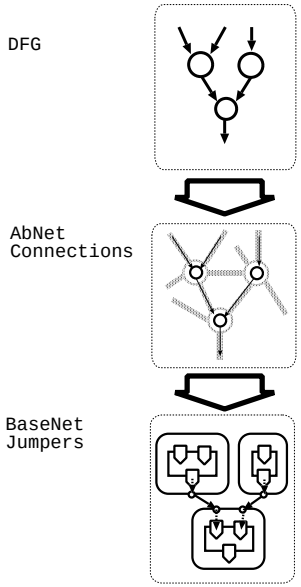

図 1 DFG から BaseNet への変換過程
Fig. 1 Transforming Procedure from DFG to BaseNet

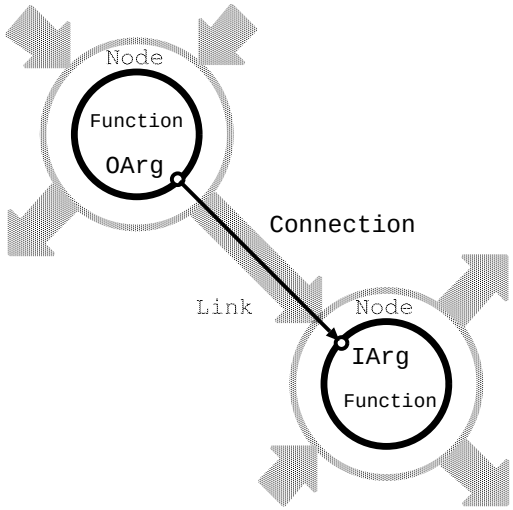

図 2 AbNet の構成要素
Fig. 2 Composition of AbNet Core

ワークはコンパイラ方式によるコネクション指向型のネットワーク[1] であるからである。一度コネクションが確立されると、実際に処理に必要な資源のみで動作し、抽象ネットワーク階層に由来するプログラムは全く動作する必要が無い。

2.1 Abstract Network

AbNet は Node と Link から成る単純有向グラフ[⋆1]である。すべての Node は AbNet の名前空間での名前を持っており、文脈によって (node n0) もしくは n0 の様に識別される。Link は Node 間の単方向データ転送機能を提供する。AbNet の Link は多重化されているので、1 Link あたり複数のデータストリームが通りうる。Link は Node の組として (link n0 n1) の様に記述される。この時 n0 は始点 Node であり、n1 は終点 Node である。

各 Node は一つ Function を持っており、入力を受け取り何がしかの処理を行なって結果を出力する。Function 名はそれが所属している Node 名と同じである。Function はデータ I/O の為に Argument(以下 Arg と省略) を持つ。Arg は単方向であり、Function に入力データを供給するものを Input Argument (以下 IArg と省略) と呼び、Function からデータを出力するものを Output Argument(以下 OArg と省略) と呼ぶ。すべての Arg は名前を持っており、その名前空間は Node 毎に独立している。文脈によって (arg n0 a0) もしくは (n0 a0) の様に識別される。ここで n0 は Node 名であり、a0 はその Node 内での Arg 名である。Node は (node n0(i0 i1)(o0 o1)) の様に記述する。ここで n0 は Node の名前であり、i0 と i1 は n0 内での IArg の名前、o0 と o1 は n0 内での OArg の名前である。Connection は 1 つの OArg と 1 つの IArg の間で、Link を経由することによっての単方向の通信機能を提供する。Connection は (connection(n0 o0)(n1 i0)) の様に記述され、この時 (n0 o0) は Node n0 の OArg 、(n1 i0) は Node n1 の IArg である。

AbNet の各要素の関係を図 2 に示す。Node と Link は実際の機能資源と通信資源にマップされる。それらの実資源は抽象ネットワークアプローチの文脈では BaseNet と呼ばれる。

2.2 Base Network

BaseNet は、機能資源である Node と通信資源である Link からなる有向グラフ[⋆2]である。ネットワークグラフの形で資源管理を行うことによって、どの機能資源がどの通信資源を利用出来るのかを取り扱う。Node は機能資源を抽象化したものであり、例えばインターネットノードである。すべての Node は、BaseNet の名前空間での名前を持っており、文脈により (node n0) または n0 のように識別される。Link は通信資源であって、ふたつの Node 間での単方向データ転送機能を提供する。例えば TCP/IP コネクションの出力方向のストリームである。Link は多重化されておらず、一本あたり 1 つのデータストリームだけを通す。

Port は、 Node の Link のに対する接点である。Node に対し入力方向の Link の接点を Input Port(以下 IPort と省略)、出力方向の接点を Output Port(以下 OPort と省略) と呼ぶ。Node は IPort を通じてその IPort で接する Link からデータを受け取り、OPort を通じてその k OPort で接する Link へデータを受渡すことが出来る。すべての Port は名前を持っており、その名前空間は Node 毎に独立している。文脈により (port n0 p0) または (n0 p0) のように識別される。n0 は Node 名、p0 はその Node 内での Port 名

⋆1 自己ループや平行多重辺のない有向グラフ

⋆2 自己ループや平行多重辺のありえる有向グラフ

Vol.2010-DPS-145 No.21
Vol.2010-GN-77 No.21
Vol.2010-EIP-50 No.21
2010/11/26

```
(node n0 (in  (a(protocol p0)) (b(protocol p0)) (c(protocol p1)) ...)
         (out (x(protocol p0)) (y(protocol p0)) (z(protocol p1)) ...)
  (fun (f0 (in  (a(allow(protocol p0))) (b (allow(port a b))))
           (out (x(allow(and(protocol p1)(port z)))))
 (att attr0 attr1) ) (f1 ...) ))
```

図 3 BaseNet node 記述
Fig. 3 BaseNet node description

である。Port の記述は Node の記述の中で行われ、様々な属性情報を付加する事が出来。例えば (p0(protocol a)) のように記述する。この例は、Port p0 はプロトコル a を利用するものであるという意味である。Link は、 Link への入口ある、ある Node の Output Port(OPort) からデータを Link の出口である、別の Node の Iutput Port(IPort) へ出力する。Link は始点 Port と終点 Port の組として、(link (n0 o0)(n1 i0)) のように記述される。n0 は Node 名、i0 と o0 はその Node 内での 始点 Port 名と終点 Port 名である。

各 Node は内部にいくつかの Function を持っている。Function は、機能資源である Node の機能を実現するものであり、例えば TCP/IP サーバプロセスである。Function はいくつかの入力を受け取り、何がしかの処理を行なっていくつかの結果を出力する。Function は多重化されておらず、一度に一つの処理しか行う事が出来ない。すべての Function は名前を持っており、その名前空間は Node 毎に独立している。文脈により (func n0 f0) または (n0 f0) のように記述される。n0 は Node 名、f0 はその Node 内での Function 名である。Function の記述は Node の記述の中に含まれており、様々な属性情報を付加する事が出来る。Function はデータの入出力の為に Argument(以下 Arg と省略) を持つ。Arg は単方向であって、Function に入力データを供給するものを Input Argument(以下 IArg と省略) と呼び、Function からデータを出力するものを Output Argument(以下 OArg と省略) と呼ぶ。すべての Arg は名前を持っており、その名前空間は Function 毎に独立している。文脈により (arg n0 f0 a0) または (n0 f0 a0) のように記述される。n0 は Node 名、f0 はその Node 内での Function 名、a0 はその Function での Argument 名である。Node は図 3 の様に記述され、この記述の中で Port と Function に付いても記述される。

2.3 AbNet から BaseNet への変換

以下では AbNet の各々の要素を BaseNet にどのように対応させていくかに先立って、まず BaseNet でどのように隣接する Function 同士を接続するか解説する。

2.3.1 Port Assignment

BaseNet では、 Arg がどの Port から入力、ないし出力を受け取るかの割り当てを行う事ができる。この割り当てを Port Assignment と呼び、(assign n0 f0 a0 p0) の様に記述される。ここで、n0 は Node 名、f0 は その Node 内での Function 名、a0 は その FUnction 内での Argument 名、p0 は その Node 内での Port 名である。IArg には IPort のみを、OArg には OPort のみを割り当てる事が出来る。Port Assignment によって、BaseNet は隣接する Node の Function 間での通信を提供する。

実際の機能資源がどの通信資源を取り扱う事が出来るかは実際に接続されているかどうかの他、その機能資源がどのプロトコルを利用できるかによる所が大きい。BaseNet では Arg の属性情報として、Port の制約を、各 Argument に対応付け可能な Port 名や、Port のプロトコルによって記述することが出来る。図 3 の (allow ...) 節が制約である。この制約は、AbNet Connection を BaseNet に対応付ける際に重要な役割を果たす。

2.3.2 Mapping of AbNet resources to real resources on BaseNet

続いて AbNet の資源を BaseNet の資源にどのように対応させるかについて述べる。ひとつの AbNet Node はひとつの AbNet Function を持ち、ひとつの BaseNet Function に対応する。この対応関係は単射であって、AbNet Node mapping table に保持され、各々の対応関係は (map an0 (n0 f0)) の様に記述される。ここで an0 は AbNet Node 名であり、(n0 f0) は BaseNet Function 名である。AbNet Function の Argument は、対応する BaseNet Function の Argument とそれぞれ記述された順番で対応する。AbNet で、ふたつの Node の間に Connection を張る事の出来るのは、AbNet の Link が存在する場合のみである。AbNet Connection は BaseNet の Port Assignment による Function の結合に対応付けられる。

2.3.3 System Functions

AbNet Connection を BaseNet Port Assignment の集合に変換する為には、いくつかの特定の機能を持った BaseNet Function が必要である。第一が “Relay” である。これは入力データをそのまま出力する 1 in 1 out Function である。この Relay の派生として “PCrelay” も必要である。これは 1 in 1 out Function であって、IArg と OArg が別々のプロトコルを取り扱う事の出来る、つまりプロトコル変換を行うことの出来る Relay(Protocol Conversion Relay) である。これら Relay と PCRelay は、十分な数が BaseNet に存在するとする。

第二が “Router” である。これは AbNet Connection 記述を BaseNet Port Assignment

Vol.2010-DPS-145 No.21
Vol.2010-GN-77 No.21
Vol.2010-EIP-50 No.21
2010/11/26

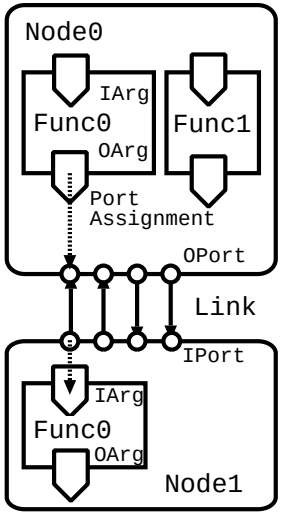

図 4 BaseNet の Port assignment
Fig. 4 Port assignment on BaseNet

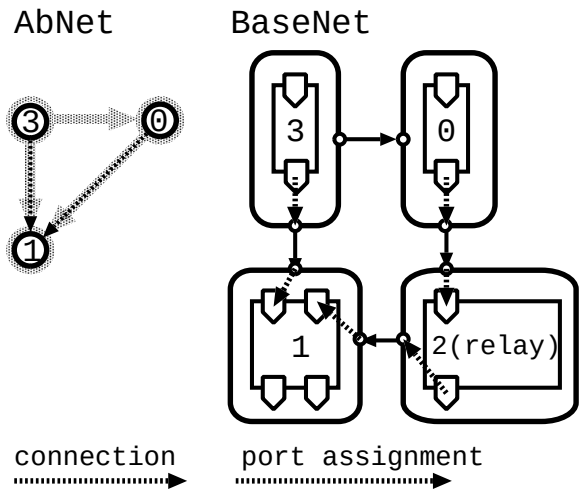

図 5 AbNet connection の BaseNet assignment への変換
Fig. 5 Mapping of AbNet connection to BaseNet assignment

の集合に変換する 1 in 1 out 関数である。Router は AbNet node mapping table を持っており、Router は 指定された AbNet Connection を満足する様な BaseNet グラフ上での経路を探索し、BaseNet 上での Port Assignment 記述の集合として経路を出力する。この経路探索は Argument と Port の接続制約を考慮して行われ、また必要ならば Relay/PCRelay を中継する形での経路を出力する。

2.4 変 換 例

それでは Router によって AbNet Connection が如何に BaseNet に map されていくかを、具体例を挙げて見ていく。今ユーザが Router に AbNet Connection (connection(an0 ax0)(an1 aa0)) を要求したとする。これに対し、 Router は要求された AbNet Connection の OArg と IArg を AbNet Node mapping table を参照して、BaseNet 上での名前に変換する。例えば今、 mapping table が (map an0(n0 f0))、(map an1(n1 f1)) というエントリを保持していたとすると、要求された AbNet Connection の OArg (an0 ax0) と IArg (an1 aa0) は各々(n0 f0 x0) と (n1 f1 a0) に変換される。

2.4.1 直 接 接 続

ここで BaseNet Node n0 、n1 の間に、利用可能な BaseNet Link (link(n0 o0)(n1 i0)) が存在し、BaseNet Argument (n0 f0 x0) 、 (n1 f1 a0) の各々の制約が、各々、BaseNet Port (n0 o0) 、 (n1 i0) を割り当てる事を許すならば、要求された AbNet Connection (connection(an0 ax0)(an1 aa0)) は、BaseNet Port Assignment (assign n0 f0 x0 o0) と (assign n1 f1 a0 i0) に変換される。

2.4.2 中 継 接 続

上記のような BaseNet Link が存在しない場合でも、BaseNet Node n0 と n1 の間に利用可能な Relay を経由する経路が存在するならば、それらを経由する形での BaseNet Port Assignment 集合に変換される。Relay は十分な数が有ると仮定する。例えば、 BaseNet Relay Function (n2 r0) と、BaseNet Link (link(n0 o0)(n2 i0)) と (link(n2 o0)(n1 i0)) が存在して、各々が利用可能でありかつ制約から割り当てが可能であるならば、要求された AbNet Connection (connection(an0 ax0)(an1 aa0)) は、BaseNet Port Assignment (assign n0 f0 x0 o0) 、 (assign n2 r0 a0 i0) 、(assign n2 r0 x0 o0) 、(assign n1 f1 a0 i0) に変換される。

2.4.3 プロトコル変換を伴う中継接続

BaseNet の端点である Argument (n0 f0 x0)、(n1 f1 a0) が、各々異なるプロトコルの BaseNet Port しか扱えない場合も、すなわちプロトコル制約が一致しない場合も、プロトコル変換を扱う PCrelay Function を用いた接続として取り扱う事が出来る。

例えば BaseNet に、(n2 pcr0) なる PCrelay Function が存在し、(link(n0 o0)(n2 i0))、(link(n2 o0)(n1 i0)) なる Link が存在して、各々が利用可能でかつ制約から割り当てが可能であるならば、すなわち、 BaseNet Port の制約が (n0(o0(protocol p0)))、 (n2(i0(protocol p0))) 、(n2(o0(protocol p1))) 、 (n1(i0(protocol p1))) 、であって、 BaseNet Argument の制約が (n0 f0(x0(allow(protocol p0)))) 、(n1 f1(a0(allow(protocol p1)))) 、(n2 pcr0(a0(allow(protocol p0)))) 、(n2 pcr0(x0(allow(protocol p1)))) であるならば、要求された AbNet Connection (connection(an0 ax0)(an1 aa0)) は、(assign n0 f0 x0 o0)、(assign n2 pcr0 a0 i0)、(assign n2 pcr0 x0 o0)、(assign n1 f1 a0 i0) という Port Assignment に map される。したがって、AbNet では経路探索によって適切なプロトコル変換を行うことが出来る。

以上から分かるように AbNet ではそれが実際の資源をどのように結合するものであって

Vol.2010-DPS-145 No.21
Vol.2010-GN-77 No.21
Vol.2010-EIP-50 No.21
2010/11/26

も、たとえプロトコル変換が必要なものであったとしても、すべて同じ AbNet Connection 記述によって書き表す事が出来る。そして最短経路が探索されるならば、つまり経由する BaseNet (PC)Relay Function が最小限に抑えられるならば、実際に動作するプログラムもまた最小限に抑えられる。直接接続と中継接続の例を図 5 に示す。各々の数字は AbNet Node と BaseNet Function の対応関係を示している。

2.5 Setup と Signaling Network

以下では Port Assignment が実際にどのようにして物理的な接続と対応付けられるか説明する。実際の Port Assignment 処理を行う為には、さらに二つの BaseNet Function が必要である。

Setup Function: N in M out の BaseNet Function である。Setup はすべての Node に存在し、自身が所属する Node を配下に置き、自身への制御情報に基づいた実際の接続操作を行うとともに、制御情報の流通を行う。Setup はいずれかの IArg から入力を受け取ると、その中から自 Node の Port Assignment 要求をより分け、それらに基づいて各々の Argument/Port の組に合った実際の接続操作を行う。その後操作の完了結果情報を適切な OArg に出力するとともに、自 Node 宛てでなかった入力を、結果出力と同じかまたは別の OArg に出力する。

Console Function: 1 in 1 out BaseNet Function である。ユーザからの接続要求を受付け、接続完了通知を出力する。Console はユーザからの指令を直ちに OArg に出力し、IArg に戻ってきた接続操作完了通知をユーザに対して出力する。

2.5.1 Signaling Network

AbNet が正しく機能する為には、BaseNet 上に存在する Link と Function によって、以下の目的の為の接続があらかじめ実現されていなければならない。

ユーザからの要求 Console で受け付けたユーザの接続要求指令の Router への伝達
制御情報の配送 Router から出力された Port Assignment の適切な Setup への配送
応答情報の集約 各々の Setup からの応答の Router への集約
ユーザへの応答 Router の接続操作完了結果の Console への応答

この接続が Signaling Network である。

2.5.1.1 Signaling Network の構成とその方法

まず、Console と Router の間については、既に存在する BaseNet Link と (PC)Relay を利用し、Console から Router へ、 Router から Console への両方の接続が行うことができなければならない。Link が存在すれば、ダイクストラ法等の最短経路探索の結果に基づいて接続を行う事によって可能である。

つぎに、Router からすべてのノード (Setup) への経路すなわち制御情報の配送経路と、すべてのノード (Setup) から Router への経路すなわち接続完了応答の集約経路を、有向グラフである BaseNet 上で形成できなければならない。配送経路は Router を根とし各々の Setup をその他の頂点とする BaseNet の外向全域木[⋆1]として構成することが出来、集約経路は同様の BaseNet の内向全域木[⋆2]として構成できる。これら外向及び内向全域木は、プリム法を有向グラフに拡張し辺の重みを 1 とおくことによって探索が可能であり、この 2 つの最小全域木を重ねあわせることにより、Signaling Network を構成する。上記 2 つの最小全域木は、すべてのノード (Setup) を含むので、Router から最小外向全域木の Link を通じ、すべてのノード (Setup) へ Port Assignment の情報の配送行うことができ、また、すべてのノード (Setup) から最小内向全域木の Link を通じ、接続動作完了通知を Router に集約できる。

2.6 SDFL: Simple Data Flow Language

AbNet 上でネットワーク分散アプリケーションを記述する為に簡単なデータフロー言語 (SDFL: Simple Data Flow Language) を定めた。

2.6.1 Function definition

SDFL はデータフロープログラムの内、有向非循環グラフで書き表せるものを記述する為のものである。SDFL は関数単位で記述される。SDFL の関数はプリミティブ関数と SDFL で記述された関数の両方を用いて、その関数のデータフローノードをインスタンスとして作成し利用する事が出来る。プリミティブ関数は AbNet に既に存在する Function であり、四則演算 add, sub, mul, div の 4 つを用意した。

関数は以下の様に定義する。

```
(sdfl-defun f0 (in a0 a1) (out r0)
  (i (f1 i0) (f2 i1))
  (c (a0 (i0 a0)) (a1 (i0 a1))
     ((i0 r0)(i1 a0)) ((i0 r1)(i1 a1)) ((i1 r1) r0))  )
```

この記述では `f0` という名前の新しい関数を定義している。`f0` は `a0` と `a1` と言う名前の入

⋆1 与えられた連結なグラフの、全ての頂点を含む連結なサブグラフであって、根の入力次数が 0、それ以外の頂点の入力次数が 1 であるもの
⋆2 与えられた連結なグラフの、全ての頂点を含む連結なサブグラフであって、根の出力次数が 0、それ以外の頂点の出力次数が 1 であるもの

Vol.2010-DPS-145 No.21
Vol.2010-GN-77 No.21
Vol.2010-EIP-50 No.21
2010/11/26

力を持ち、r0 という名前の出力を持っている。i 節で内部では f1 という名前の SDFL 関数のインスタンスを i0 と言う名前で作成し、f2 という名前の SDFL 関数のインスタンスを i1 と言う名前で作成して利用する事を宣言している。c 節ではそれらのインスタンスと自身の入力と出力をどの様に結合するか宣言している。例えば (a0(i0 a0)) は自身の a0 を i0 の入力 a0 に対応付ける事を意味し、((i0 r0)(i1 a0)) は i0 の出力 r0 を i1 の入力 a0 に結合する事を意味する。SDFL 関数はすべてプリミティブ関数の結合グラフに展開される。

2.6.2 Allocator

SDFL を AbNet に割り当てるために SDFL Allocator が存在する。SDFL Allocator はあらかじめプリミティブ関数に対応する AbNet Function の対応テーブルを持っている。Allocator には SDFL 関数と、その関数に対する入力データを与える AbNet Node の OArg と、その関数からの出力データを受ける AbNet Node の IArg を与える。SDFL Allocator は SDFL 関数の接続関係を考慮しながら与えられた SDFL 関数が必要とするプリミティブ関数と AbNet Function の間での対応をとって、SDFL 関数が必要とする結合関係を AbNet Connection 記述に展開する。SDFL Allocator は以下の様に呼び出す。

```
(sdfl-alloc(f0)((n0 x0)(n1 x0))((n2 a0)))
```

この様に呼び出すと、SDFL Allocator は f0 内部で利用されるプリミティブ関数のインスタンスを AbNet Function に map し、内部の接続を AbNet Connection 記述に変換して、Abnet Arg(n0 x0)、(n1 x0) を入力とし、(n2 a0) に出力するような、AbNet Connection 記述を生成する。そしてここまでで生成された AbNet Connection 記述を出力する。

3. SDFL 関数の変換例

以下では簡単な SDFL 関数を定義し、それがどのような過程を経て変換されていくか見ていく。AbNet 記述と BaseNet 記述は紙面の都合上割愛する。

まず SDFL 関数 dda と da を定義している (図 6(1)(2))。da のインスタンスは ida と言う名前で dda の中で用いられている (図 6(1))。続いて SDFL アロケータに対して、daa 関数を (n0t o0)、(n0t o1)、(n0t o2) を入力とし、(n0t i0) を出力として割り当てるように要求する (図 6(3))。

SDFL アロケータは図 7 の様な Connection 記述を出力する。AbNet の Link と SDFL 関数の結合定義を照らし合わせて、実際に Connection が張れるものを出力する。

次に connection 記述を Router に渡すと、図 8 の様な Port Assignment 記述を出力する。

```
(sdfl-defun dda (in a b)(out x)
  (i (da da) (div d))
  (c (a(d a)) (b(d b)) ((d x)(da a)) ((d y)(da b)) ((da x)x)) )   (1)
(sdfl-defun da (in a b) (out x)
  (i (div d) (add a))
  (c (a(d a)) (b(d b)) ((d x)(a a)) ((d y)(a b)) ((a x)x)) )      (2)

(sdfl-alloc (dda) ((n0t o0) (n0t o1)) ((n0t i0)))                 (3)
```

図 6 SDFL 関数定義と allocation
Fig. 6 SDFL function definition and allocation

```
(connection( t o0)(d0  a)) (connection( t o1)(d0  b))
(connection(d0  x)(d1  a)) (connection(d0  y)(d1  b))
(connection(d1  x)(a1  a)) (connection(d1  y)(a1  b))
(connection(a1  x)( t i0))
```

図 7 生成された AbNet Connection
Fig. 7 Generated AbNet Connection

```
(assign n0  t o0 udo00) (assign n0  d  a udi00)
(assign n0  t o1 udo01) (assign n0  d  b udi01)
(assign n0  d  x ino10) (assign n1  d  a ini00)
(assign n0  d  y ino11) (assign n1  d  b ini01)
(assign n1  d  x udo10) (assign n1  a  a udi10)
(assign n1  d  y udo11) (assign n1  a  b udi11)
(assign n1  a  x udo20) (assign n2 r0  a udi10)
(assign n2 r0  x ino00) (assign n0  t i0 ini20)
```

図 8 生成された BaseNet Port Assignment
Fig. 8 Generated BaseNet Port Assignment

Vol.2010-DPS-145 No.21
Vol.2010-GN-77 No.21
Vol.2010-EIP-50 No.21
2010/11/26

最後にこれらの Port Assignment 記述は `Setup` に送られ、適切な実際の操作が行われた。これにより、Abnet Nodet を用いて実際に `dda` 関数を利用することが出来る。

4. 結　　論

本稿ではネットワーク分散システムを実現する為の手法として抽象ネットワークによる手法を提案した。これは利用可能な資源をネットワークグラフによって管理し、これに抽象的なネットワーク層を導入する方式である。この上でデータフロー方式に則ってアプリケーションを開発することにより、プロトコル変換ソフトウェアの開発の必要性を最低限に抑え、レイヤが多重化してもオーバーヘッドは最低限に抑えられる。

次に抽象ネットワークの一つの実装として、Abstract Network(AbNet)、Base Network(BaseNet) を定義し AbNet Connection から BaseNet Port Assignment への変換の方法と BaseNet Port Assignment を実際に行う Setup からなる Signaling Network を説明した。さらに AbNet のインターフェースとして単純なデータフロー言語 SDFL を定義し、続いて SDFL で記述されたデータフローグラフを AbNet に展開する方法を提示した。

この抽象ネットワークによる手法が正しく動作し有効である事を示すために、抽象ネットワークの一つの実装として Abstract Network を定義し、プロトタイプシステムを実装して実験 AbNet/BaseNet 環境で動作させた。その結果プロトコルの違いをまったく意識せずにネットワーク分散環境の上で機能を結合でき、それでいて最低限の資源でこれを実現できるとわかった。

5. 今後の課題

本プロトタイプはネットワーク分散環境へ抽象ネットワークアプローチを適用したものである。本アプローチはシステム構築に際してコンパイラ的な手法を適用するというものであり、これが適用可能な対象はネットワーク分散システムの枠に止まらないと思われる。そしてもしそうであるならば計算機システムに対する包括的なインターフェースを構築できる可能性があると言えるだろう。しかしその可能性を追求するにはまだ様々な課題がある。まず経路探索に関する課題がある。今回の実験では単純にホップ数の少ない経路を選択するアルゴリズムを用いていた。実際には経路の性能評価値、例えば転送遅延時間によって評価されるべきである。より効率的な割り当て/経路探索アルゴリズムを実装すべきである。また本稿で論じた実装では、与えられた SDFL 関数に対してまず Function の割り当てのみを行い、その後 connection の map を行っていた。この様に Function と Connection の割り当ての探索を分けて行うと局所最適解を探索してしまい、全体としては最適でない可能性が高くなる。効率的な Function 割り当ての為にはこの二つの探索を同時に行う必要がある。そうでありながら一方で巨大な BaseNet を取り扱う場合、探索範囲が非常に広くなってしまう可能性がある。上手く探索範囲を限定し、徐々に拡大出来るような仕組みが必要である。

また上記の様なより良い探索の為には BaseNet でより多くの情報を取り扱える必要がある。記述の改良は探索の為だけではなく通信路の暗号化やバッファリング等の様々な機能を正確に扱う為にも必要である。

別の課題として、Setup と Signaling Network に関する課題がある。現状の Setup は BaseNet の Port Assignment を理解できる必要があるが、これでは初めから BaseNet に組み込む為に作られた資源のみしか組み込む事が出来ない。この部分を BaseNet 上できちんと抽象化してやる必要がある。また現状の Signaling Network はアドホックなものであるので、構成法も含めて見直す必要がある。

参 考 文 献

1) Tanenbaum, A.S.: *Computer Networks 4th Edition*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA (2002).
2) Tanenbaum, A. S. and van Steen, M.: *Distributed Systems: Principles and Paradigms (2nd Edition)*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA (2006).
3) Whiting, P.G. and Pascoe, R. S.V.: A History of Data-Flow Languages, *IEEE Ann. Hist. Comput.*, Vol.16, No.4, pp.38–59 (1994).