情報処理学会研究報告
IPSJ SIG Technical Report

連邦型 Linda による分散アルゴリズムをデバッグするためのメタプロトコル

赤嶺悠太[†1]　小野雅俊[†2]　河野真治[†2]

連邦型 Linda とは、複数の Lida Server を接続して分散アルゴリズムを実装するフレームワークである。今回は、分散アルゴリズムをデバッグするために、通常のタプル通信の他にメタな通信を行なうプロトコルエンジンの設計と実装を行なった。メタ通信は、タプル通信と相似な形式を持っているが、Linda サーバ内部に直接アクセスすることが出来る。これにより、デバッグプロトコル自体をスケーラブルな分散プロトコルとして実装することが出来る。

Meta Protocol for Debugging of Distributed Algorithm by Federated Linda

Kento YOGI,[†1] Kenta MIYAGI[†1]
and Shinji KONO [†2]

1. はじめに

分散アプリケーションが多数開発されている近年、ノードやネットワークの動的な変化、故障、性能の多様性を考慮したフレームワークが必要である。分散環境ではスケーラビリティを確保すること重要である。ここでのスケーラビリティとはノードの規模の増大にも関わらず、アプリケーションの性能を維持できることを指す。分散アルゴリズムの作成には、論理的なプログラムの正しさだけでなく、スケーラビリティのデバッグを可能にする必要がある。つまり、デバッガ自体をスケーラブルに作る必要がある。そのため、分散フレームワークとして本研究室が提案している Federated Linda に、スケーラビリティなデバッグを行うためのメタな通信を行うプロトコルエンジンを設計し実装した。

分散プログラムのデバッグを行う際には通信は必須であるが、その通信によって本来の通信に影響を及ぼす恐れがある。ここでは、スケーラビリティには若干問題があるが、本来の通信に影響を与えないように、一つのトークンをリング上に流す方法を提案[1),6)] している。本論文では PC クラスタ上で実際の通信を行ない、100 台程度の分散プログラムでの評価を行なった。

2. Scale する分散プログラム

Internet 上で動作している分散プログラムは、例えば、DNS、SMTP、NNTP、あるいは、さまざまな P2P や DHT などがある。これらのサービスで重要なのは、Scalability つまり、サービスの規模が大きくなっても、応答速度などの要求仕様を満たすことである。このようなプログラムは、一つのコンピュータに閉じた逐次型のプログラムや、マルチスレッドのプログラムとは様相が異なる。

Federated Linda[4),5)] は、計算モデルが明解な Linda[3)] を複数接続することで、分散プログラムの作り方を明確なモデル上で学ぶことができるようにするように設計されている。

我々は、この上で投票システムや Compact Routing[2),7)] 等を実装して来たが、分散プログラムは、それ自体が複雑なだけでなく、デバッグ自体の困難さが問題であることがわかってきた。

最初の目標は、琉大情報工学科にある 200 台規模の PC クラスタ上で、Federated Linda を用いた分散プログラムを作成しデバッグすることである。

分散プログラムは正しい答えを出すだけでなく、Scalability があるかどうかを調べることが必要である。PC クラスタ上のデバッグでは、デバッグ自体に通信が必要であり、その通信が Scalability に影響しないようにする必要がある。

3. プロトコルエンジンとタプル空間

Federated Linda とは、複数のタプル空間を相互に結ぶプロトコルエンジンによって接続する分散プログラミングモデルである。Linda と同じ API を持つが、Linda が一つのタプル空間を共有するのに対し、Federated Linda は複数のタプル空間を個別に持つ。クライアントのアクセス数に対応して、タプルスペースの数を増やし処理を分散させる事により、ス

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

ケーラビリティを保つ。

4. プログラミング例

Federated Linda は以下の様にして通信を行う。

- public PSXLinda open(String _host,int _port)
  Linda Server に対し、接続を行う。引数はホスト名とポート番号をとる。返り値はタプルスペースの番号となる。
- public PSXReply in(int id)
  タプルスペース番号より引数で指定した id のタプルの受け取りを要求する。受け取りは、返値の PSXReply をチェックすることにより非同期に行なう。in では待ち合わせは行なわれない。Call back 形式でタプルを受け取ることも出来る。
- public PSXReply out(int id, ByteBuffer data)
  引数で指定した id で ByteBuffer 内のデータを送信する。
- public int sync(long mtimeout)
  接続している Linda Server とタプルの送受信のポーリングを行う。

Protocol Engine とはタプルスペースを介してデータをやり取りする Engine である。二つのサーバー間でやり取りを行う場合、以下のようなプログラムとなる。

```
FederatedLinda fdl;
PSXLinda getpsx;
PSXLinda sendpsx;
PSXReply in;
ByteBuffer data = ByteBuffer.allocate(10);

//通信を初期化する
fdl = FederatedLinda.init();
//データを取ってくるホストと受け渡すホストとの接続開始
getpsx  = fdl.open("cs100",10000);
sendpsx = fdl.open("cs101",10001);
//取ってくるホストから in を指定してデータを取得
in = getpsx.in(10)
data = in.getData();
//受け渡すホストに対しデータと id を指定して同期
sendpsx.out(10,data);
fdl.sync();
```

callback の API も用意されていて、`fdl.sync()` した時に、`in, rd` の結果が戻っていれば、その callback が実行されるようになっている。

5. デバッグするには?

Federated Linda 上でデバッグする一つの方法は、デバッガからタプルスペースへ問い合わせの通信を行なうことである (図 5)。

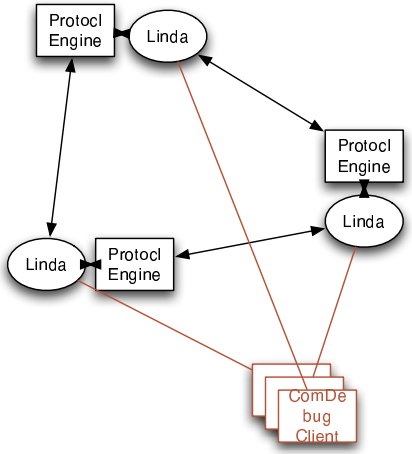

図 1 集中型デバッガ

この方法では、Linda Server の ad-hoc な改変が必要であり、デバッガは各 Linda Server へ 1 対多の集中的な通信を行なう必要がある。この方法では、デバッガは Linda Server への直接の通信路を持つ必要があるが、分散環境では、ファイアウォールなどの関係で、それが可能であるとは限らない。

デバッグ自体は、タプル空間に直接アクセス出来るプロトコルエンジンと考えることが

でき、Federated Linda のメタエンジンととらえることができる。メタエンジンの API を Linda にそろえることにより、Linda Server への ad-hoc な改変を、決まった API 上のデバッグプロトコルの設計にすることができる (図 5)。

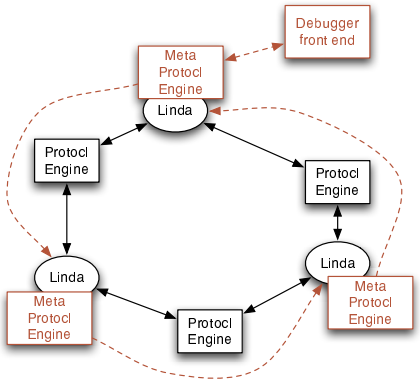

図 2 Debugger

デバッグ自体を Scalable にして、分散計算への影響を少なくするためには、デバッグ用の通信自体が Scalable である必要がある。

それには、デバッグプロトコル自体が、Federated Linda によって Scalable だと示されたプロトコルであることが望ましい。つまり、最初に情報収集などに適したプロトコルを Federated Linda で作成し、それをそのままデバッガのプロトコルに採用できることが望ましい。

6. メタエンジン

デバッグ用の通信は Linda Server 内部に直接アクセス出来なければならない。これを Linda Server 内の Meta Protocol Engine によって実現する (図 6)。

Meta Engine は、通常の FederatedLinda と同様の in/out API を持つ。Meta Engine では、`sync()` が、属する Linda Server 自体のポーリングを行なう。`sync()` の間は、通信は行なわれず タプル空間は変化しない。Meta Engine は安全にタプル空間にアクセス出来る。Meta Engine のプログラムは、Linda Server のメインループで指定することが出来、Server 毎に独自の動作をさせることが可能である。

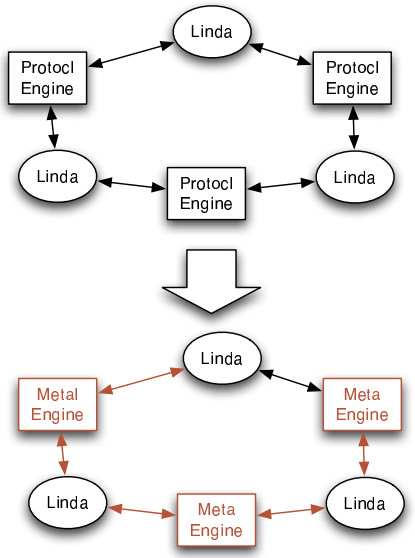

図 3 メタへの移行

7. メタエンジンの実装

ここでの Linda Server は、Single Thread に実装されており、Java nio の select で待ち、通信パケットを受け取って処理するというメインループを持っている。メタエンジンは、このメインループを置き換える形で実装した。つまり、メタエンジン自体が `sync()` しながらループするという構造を持っている (図 6)。

メタエンジンは、Linda server の立ち上げ時、または、メタエンジンそのものから、特殊なものに置き換える API を用意する。

API は、通常の Linda の API だが、メタエンジンでは、タプルスペースに対して直接アクセスする。

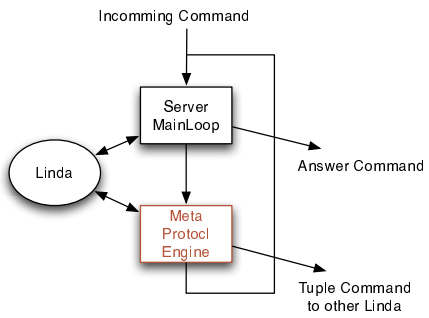

図 4 メタエンジン

Single Thread Server 上のメインループの一部として実現されているので、タプルスペースを lock することなく自由にアクセスすることができる。ここでの Linda API は `in,rd` で待ち合わせしない仕様になっていて、待ち合わせは、`sync()` とポーリング、または、call back によって実現されている。従って、メインループの一部としても、`in,rd` の待ち合わせによってデッドロックするようなことはない。

8. 分散プログラムのデバッグ手法

Federated Linda では、プロトコルエンジンは、タプルスペース (Linda Server) から、タプルを受け取って、それに計算を施して、他のタプルスペースへ引き渡す。従って、バグは、あるタプルを受け取って、どのようなタプルを出力するかというのを見ることになる。

個々のプロトコルエンジンの計算が正しくても、大域的なエラーが起きる場合も存在するので、個々の処理だけでなく、全体的な状態の情報も必要となる場合がある。

通信状態を含めた大域的な状態は分散スナップショットと呼ばれる。分散スナップショットを取ること自体に通信が必要である。また、分散スナップショットは、未来からの通信が含まれている場合は、再実行可能でないことがある。再実行可能なスナップショットを取るためには、通常の通信に制限をかけることが必要である。

デバッグプロトコルには、個々の Tuple Space の情報収集とともに、スナップショットを取る機能が必要である。スナップショットが取れれば、そのイメージを調べることによりデッドロック検出も可能となる。

Scalability の検証では、通信の集中を見つける必要がある。そのためには、タプルスペース側での通信の log の監視を行なう必要がある。すべての log 情報を一ヶ所に集めることなく、通信の集中を調べる機能が必要である。

9. デバッグプロトコル

分散環境で情報を集めるには、デバッグのための通信そのものが分散プログラムになる。ここでは、PC クラスタ上でのシミュレーションを想定して、ターゲットの通信を妨げないデバッグ通信を考える。200 台規模では、単一のリング上の通信が良いと考えている。

メタエンジン上にリングを構成し、デバッガフロントエンドは、そのリングを通して、コマンドを送ったり情報を収集したりする構成である (図 5)。

10. Simulation

デバッグプロトコルを実装するために、PC クラスタによるシミュレーションを行なった。メタエンジン上にリングを構成し、その周回時間をパケットの大きさを変えて調べる。これにより、リングを使うことによるデバッグのユーザへの応答性能と、デバッグを行なう情報を交換する時のパケットの適切な大きさを調べることができる。これらの数値は、その時その時の技術に依存している。

本稿では分散通信に影響を最低限にするために、Ring で性能を評価する。3 台の Linda Server 間で Meta Engine がデータをやり取りする場合の UML シーケンス図は図 5 のようになる。

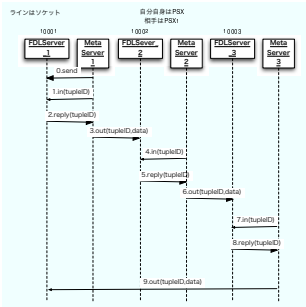

図 5 3 台間の通信

Ring では通信パケットは一つのみであり、デバッグ対象への影響が小さい。しかし、スナップショットや一時停止などのデバッグ操作をするためには、全ノードを周回する必要がある。

ここでは、通信パケットの大きさを変えて、3〜100 までの台数でデータが 1 周 (図 6) する時間、および 1000 周 (図 7) した時に掛かった時間を測定する。前者では接続の手間を含む通信時間、後者では通信のみの時間を計ることが出来る。

実験は、琉球大学情報工学科のクラスタ上 (Core Duo 2GHz, メモリ 1GB) で、クラスタジョブ管理システム Torque を用いて行なった。ネットワークは AlaxalA Gigabit Ethernet Switch で構成されている。クラスタ自体は 180 台あるが、安定して動作する 100 台までを使用して測定を行なった。

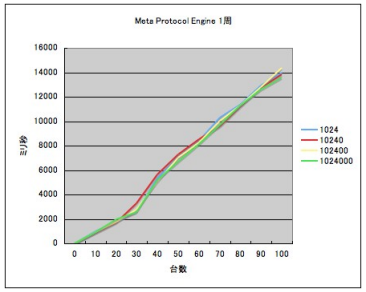

図 6 接続を含む一周の時間

X 軸が台数、Y 軸がミリ秒、ラインの色が通信するデータサイズを表す。両図から見てわかる通り、データの量にはあまり依存する事はなくほぼ同じラインを形作っている。データを 1 周のみした場合は 1 サイクルあたり約 14000ms、一台あたり約 140ms 掛かっている計算になる。これは、TCP の接続時間がかなり大ききことを示している。1MB 程度の通信を隠すほど接続時間のオーバヘッドは大きい。14 秒はインタラクティブなデバッガとしては容認できないと思われる。従って、毎回、新しく接続するような HTML のような通信を採用することはできないことがわかる。

それに対し 1000 周した際に掛かった時間は、1 サイクルおよそ 60ms、一台につき約 0.6ms となっている。これより、一度、接続してしまえば、Meta Engine での通信は実際に 100 台程度のデバッグに使用するのに十分な性能を持っていることが確認出来た。

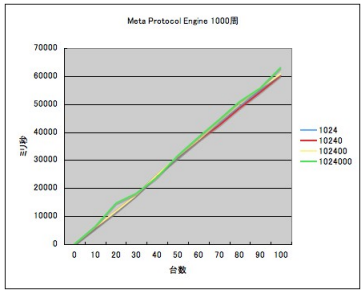

図 7 千周の平均周回時間

パケット 1KB から 100KB までの差は 2 倍程度であり、それ以上はパケットサイズにリニアに時間がかかる。従って、数十 KB 程度以下にデータを抑えることは、応答時間的には意味がない。

11. 比　　較

本稿ではデバッグを行う為に通常通信とは他に、Meta で通信する Meta Protocol Engine を実装し評価した。今回は、スナップショットなどの実際のデバッグ機能を実装することは出来なかった。通信の実験においても、同クラスタ上で別の Ring 通信や他の MetaLinda 通信等があった場合の干渉の程度などを測定する必要があると考えられる。

参　考　文　献

1) 上里 献一 and 河野 真治. Suci ライブラリのスナップショット API を利用した並列デバッグツールの設計. 日本ソフトウェア科学会第 20 回大会論文集, Sep 2003.
2) Ittai Abraham, Noam Nisan, Cyril Gavoille, Dahlia Malkhi, and Mikkel Thorup. Compact name-independent routing with minimum stretch. In *In Proceedings of the 16th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2004*, pp. 20–24. ACM Press, 2004.
3) Sudhir Ahuja, Nicholas Carriero, and David Gelernter. Linda and friends. *IEEE Computer*, Aug. 1986.
4) 安村 恭一 and 河野 真治. 動的ルーティングによりタプル配信を行なう分散タプルスペース Federated Linda. 日本ソフトウェア科学会第 22 回大会論文集, Sep 2005.
5) 安村恭一, 河野真治. 大域 ID を持たない連邦型タプルスペース Federated Linda. 情報処理学会システムソフトウェアとオペレーティング・システム研究会, May 2005.

6) 上里 献一 (琉球大)，河野真治 (琉球大，科学技術振興事業団さきがけ研究 21(機能と構成)). スナップショットを用いた PC Cluster 用デバッグツール. 情報処理学会システムソフトウェアとオペレーティング・システム研究会, May 2003.
7) 淵田 良彦, 河野 真治 . 連邦型タプルスペースを使ったコンパクトルーティングの実験 . 第 62 回情報処理学会・プログラミング研究会, Jan 2008.