分散プログラムにおけるデバッグツールの設計と実装

035713J 小野雅俊　指導教員：河野真治

1　スケーラビリティをデバッグする

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為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, 2] 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　Federated Linda

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、スケーラビリティを保つ。

2.1　Federated Linda Protocol

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。

3　Protocol Engine

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。

例）ポートを変えた、二つの Local Linda Server 間のデータをやり取りする場合

```
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(localhost,10000);
sendpsx = fdl.open(localhost,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();
```

4 Meta Protocol Engine

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 1)。

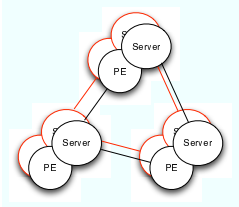

図 1: Meta Protocol Engine

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。

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 2 のようになる。

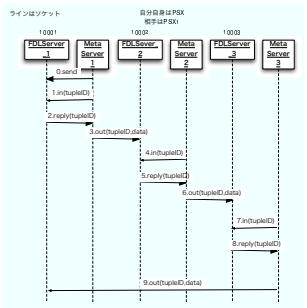

図 2: 3 台間の通信

5 測定

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。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。これは o(n) であり、十分にスケーラビリティがあるとは言えない。しかし、もっとも影響が少ない方法なので、どの程度まで使えるかを測定することには意味がある。

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 3) する時間、および 1000 周 (図 4) 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。

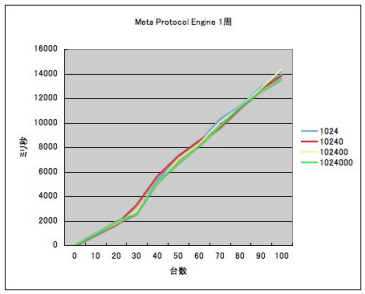

図 3: 接続を含む一周の時間

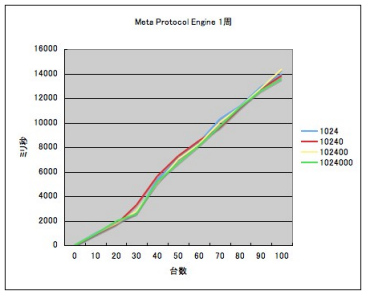

図 4: 千周の平均周回時間

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り、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。

6 まとめと課題

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, 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

[2] 渕田 良彦, 河野 真治 分散プログラミングモデル Federated Linda と分散デバッグ