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　はじめに

分散プログラミングが多数開発されている近年、プログラム作成時にクラスタ上の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する必要があり、またそれを踏まえた上でのスケーラビリティを確保することが必要とされてきている。
スケーラビリティとはリソースを増やしてサービス能力を（理想では）直線的に拡張出来る能力であり、負荷の増加に関してはリソースの追加のみで、アプリケーション自体を変更する必要性がない事を示す。本稿は分散プログラムに河野先生が提案、作成した Federated Linda を用い、スケーラビリティなデバッグを行う為に通常のタプル通信の他にメタな通信を行うプロトコルエンジンの設計と実装について検討する。

2　分散プログラムの問題点

デバッグを行う際に通常の通信を用いて行うのでは、本来の通信に影響を及ぼす恐れがある。よって、デバックプロトコル自身がスケーラビリティでなければいけない。つまり、サーバーの台数が変動した時に、デバッグプロトコルを別途用意する事なく同じ様に変動しなければならない。本稿では snapshot を Ring を使って取る為、Ring の性能テストを評価する。

3　Federated Linda

Federated Linda とは、複数のタプル空間を相互に接続する事によって分散プログラムを実現するモデルである。Linda とは同じ構造ではあるが、Linda が一つのタプル空間を共有するのに対し、Federated Linda はタプル空間通しで in/out を行う。
クライアントのアクセス数が増えたとしても、タプルスペース等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。

本稿では、Linda Server と Protocol Engine が一体化した type3 で実装を検討する。（図 1）

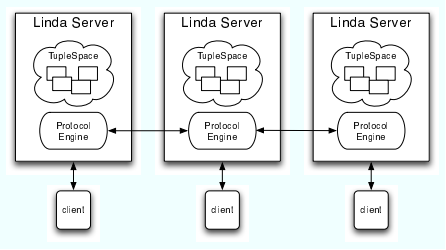

図 1: type3

3.1　Federated Linda Protocol

Federated Linda は以下の様にして通信を行う。

- open(host,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。
- in(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。返り値は PSXReply で用いるユニークな番号となる。
- out(id,data)
  引数で指定した id でデータを送信する。data は Bytebuffer でなければならない。
- sync()
  接続している Linda Server とタプルの送受信を行う。

4　Protocol Engine

type2 で実装されている 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();
```

5 Meta Protocol Engine

type3 ではスケーラビリティを実現する為に Linda Server に直接 Protocol Engine を実装する。この Engine を Memta Protocol Engine と呼ぶ。
通常の FederatedLinda と同様 in/out 出来るプロトコルを持ち、Protocol Engine と違う所はサーバー自身に直接アクセスして、データを取得する事が出来る事である。Linda Server のメインループにメソッドを追加する事により、Server が立ち上がると同時に Meta Protocol Engine を起動する事が出来る。

6 測定

Meta Protocol Engine での通信テスト、3 100 までの台数でデータが 1 周（図 2）および 1000 周（図 3）した時に掛かった時間を測定する。
測定環境はクラスタ（cls001.cs.ie.u-ryukyu.ac.jp cls180.cs.ie.u-ryukyu.ac.jp）で Torque を用いる。
出来得るならば、180 台までの測定を行いたかったが、使えないクラスタの台数が不安定に変動するので、安定して測定出来る 100 台までを測定する事とした。
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。
これは、1 周のみだと通信の際の open/close の時間が大きく影響するためだと思われる。

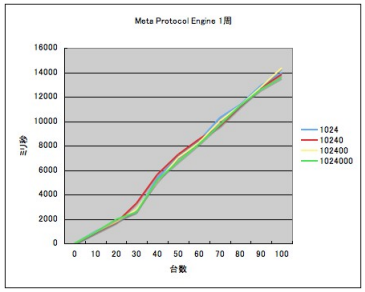

図 2: 一周した Time

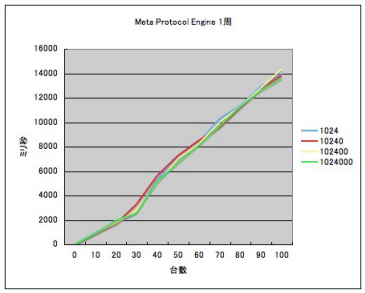

図 3: 1000 周した Time

7 終わりに

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 を提案した。
その結果、別途デバッグ用 Protocol Engine を用意する事なく Linda 間で通信が出来るため、よりスケーラブルなデバッグ環境が実現出来る事が分かった。今後の問題としては、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に、デバッグプログラムと干渉する事はないか、本 Meta Protocol Engine で実際にデバッグタプルを通信した場合に通常の通信との干渉するかどうかなどが、挙げられる。

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