FederatedLinda における デバッグツールの設計と実装

065701J 赤嶺一樹　指導教員：河野真治

平成 22 年 2 月 16 日

1　概要

分散プログラミングを行うにあたり、直面する難しい問題の一つに、自然にスケールするシステムを設計することが挙げられる。ここで言うスケールとは、リソースを増やしてサービス能力を直線的に増やす事である。

本研究室では、そのようなスケーラビリティを備えた分散プログラムを記述できるようなプログラミングモデルを提供するために FederatedLinda を開発した。

しかし、分散プログラムは複数のプログラムが他のプロセスで同時に走り、それらは相互に関わりをもつため、通常の方法でデバッグを行うことは難しい。そこで、本研究では、リング型のデバッグ用トポロジーを他のトポロジーの裏で構築し、リング上を一周するトークンによって、指定したタプルの現在の情報を取得するといったデバッグの基本機能の設計と実装を行った。

2　FederatedLinda

Linda は、 in(), out(), rd() といった API を用いてタプルスペース上のデータを更新する。しかし、これでは 1 つのサーバーへ処理が集中するため、スケールするモデルとはいえない。そこで、FederatedLinda という、連邦型タプルスペースを提案し、実装してきた。

FederatedLinda は ProtocolEngine を用いてデータの伝搬を行うが、同一サーバー上でも、プロセスが別なためメモリ空間を共有できない。そこで、サーバーのプロセスと同一の MetaProtocolEngine を用いることにした。

3　デバッグツールの概要

今回の研究では、ツリー型のトポロジーと、デバッグ用のリング型トポロジーを構築して実験を行った。

図 1 はその接続の具体例である。実線と破線は、それぞれツリーとリングの接続を表している。

ツリーでは、親から送られてきたデータは子に、子から送られてきたデータは親へと伝搬される。一方、リングでは、左から入ってきたデータは右に、右から入ってきたデータは左へと伝搬される。

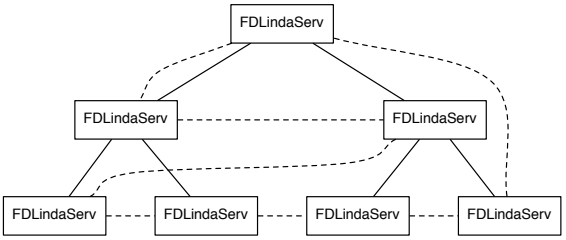

図 1: ツリーとデバッグ用のリング

3.1　タプルの伝搬

FederatedLinda サーバー間のタプルの伝搬は、タプルへの in(), out() をベースに行われる。データをやり取りするとき、1 コネクションあたり、1 タプルを割り当てることによって、どこのサーバーから送られてきたかを MetaProtocolEngine は知ることができる。そのため、接続は Poll を行うタプル番号で管理され、ホスト名とポート番号を意識してコードを書くのは、最初の接続のときのみである。

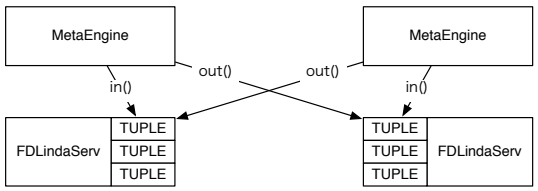

図 2: in() と out() によるタプルの伝搬

MetaProtocolEngine は、まず、接続を行ったら、その接続に対応するタプルに in() を行う。in() にはコールバッククラスを渡し、外部から out() によってデータが送られてきたときにそのコールバッククラスのメソッドを実行するように設定する。(図 2) そのコールバックに、ルーティングテーブルにしたがって、データを送信するようにコードを記述することによって、タプルの伝搬が行われるようになる。

3.2 ルーティング

起動した FDLindaServ は ConfigurationManager に起動を通知し。実験に十分な台数が集まったら、各々の接続先とルーティングテーブルを各 FDLindaServ へと XML 形式で送信する。接続が完了したら、サーバーへと完了を通知する。

3.3 デバッグ用トークン

デバッグ用の命令を Manager がリングの先頭ノードに出力すると、先頭ノードはコマンドを解釈して、データを算出し、トークンにのせ、隣のノードへと伝搬する。Manager と 1 対多のデバッグツールでは、一部にアクセスが集中してスケールすることはない。しかし、この方法をとると、集中を避けることができる (図 3)

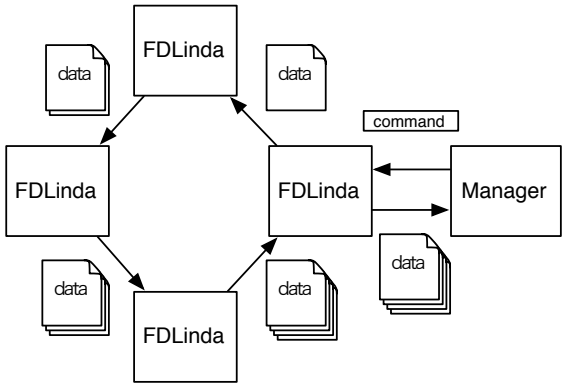

図 3: ノードのデータをトークンに付加して伝搬

4 評価

4.1 デバッグツールの検証と結果

今回の実験では、評価に TORQUE によって管理されたクラスタ 30 台を用いた。

まず、クラスタを用いてツリートポロジーを形成し、その後ろにデバッグ用のリングトポロジーを形成する。

ツリートポロジー上で、一番上の親ノードから、末端の子のノードへと、タプルを伝搬させ、データが 100 周するまでの時間を計測する。

また、その裏に張られているリングトポロジー上でデバッグ用のトークンタプルをループさせた場合のロス時間を計測する。

表 1 は、上記の実験の結果である。ツリーを 100 周した合計時間を平均して 1 周あたりの時間を求めた。

以上の結果より、デバッグエンジンを用いた時のロスタイムは、実用的な範囲に収まっている事が分かる。

表 1: デバッグエンジンの有無による影響

デバッグエンジンの有無	無	有
1 周にかかる時間 [ms]	11.38	12.35

また、デバッグ用のトークンは、 1 台あたり、100 Bytes の文字列 (XML) が付加される。この内容は冗長なため、圧縮して送信すれば、さらなるパフォーマンスを期待できる。

4.2 TCP No Delay の効果の検証

また、 TCP No Delay による、タプルの転送速度の差を検証した。使用したクラスタの台数は 46 台である。

リングに流したデータには、 4096 Bytes のデータを用い、 100 周したうちの平均時間より、 1 周にかかる時間を算出した。

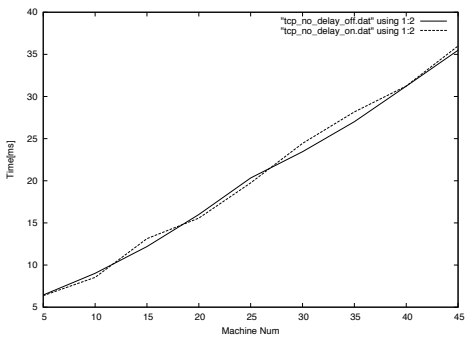

図 4: TCP No Delay の検証

上記のことより、TCP No Delay によって、パフォーマンスに大きな差を得られることはない。

5 まとめ

先行の研究では、Manager に集中するタイプのデバッグツールであったため、スケールするものではなかった。しかし、今回の Ring を用いる方式のデバッグツールは、隣にデータを伝搬していくのみの単純な処理で構成されるため、一部のサーバーにアクセスが集中することはなくなり、スケーラブルなデバッグツールとなった。

参考文献

[1] Yoshihiko Fuchita 分散プログラミングモデル Federated Linda と分散デバッグ開発

[2] Masatoshi ONO 分散プログラムにおけるデバッグツールの設計と実装