

DataSegment API を用いた分散フレームワークの設計

赤嶺 一樹, 河野真治

本研究室では、分散フレームワークとして、Linda を拡張した FederatedLinda を開発してきた。しかし、分散用の API として、in/out/read だけでは足りないことが判明した。また、分散 KVS には、read/write/update といった API があるが、同期する機構は備わっていないため、ゲームなどのリアルタイムプログラム用途にそのまま利用することはできない。そこで、Linda の待ち合わせ可能な API と一般的な KVS の持つデータベースの API の中間になるような DataSegment API を設計する。本研究室では、その DataSegment API を用いた分散フレームワークを開発する。

1 スケーラブルな分散フレームワークの提案

ブロードバンド環境やスマートフォンなどをはじめとしたモバイル端末の普及により、ネットワークに接続している端末が増えている。それにしがつて、ネットワーク上におけるサービスの巨大化は必至となっている。そこで、サービスのスタートアップ時期には少量のリソースで運用でき、ユーザー数が増える時期には、リソースを追加するのみでサービスの質を維持できるといった、スケーラビリティを備えたシステムが求められている。そのようなシステムを開発するためには、様々なプロトコルを提案し、実装し、検証していく必要がある。そのため、提案したプロトコルを玩具的に実験できるフレームワークが求められている。本研究室ではこれまでに、Linda を拡張した FederatedLinda を開発し、スケーラビリティのあるシステムについて考えてきた。すると、その開発サイクルの中でいくつかの問題点があることが分かった。また、本研究室が開発している、

並列フレームワーク Cerium に新しく実装する予定の DataSegment と CodeSegment を用いた並列計算に関するアイデアが、分散アプリケーションにも応用できるのではないかと考えた。上記の理由から、新しい分散フレームワークを提案する。本論文では、DataSegment と CodeSegment を用いた分散フレームワークに関して提案する。

2 FederatedLinda

2.1 Linda とは

Linda は、タプルスペースという ID で区画されたデータストアに、以下の API (表??) を用いてデータを出し入れすることによって、外部との通信を行う分散プログラミングモデルである。

表 1 Linda API

in(id)	タプル空間から取り出す。 タプル空間にタプルは残らない。
rd(id)	タプル空間から取り出す。 タプル空間にタプルが残る。
out(id,data)	タプル空間にタプルを入れる。

Design of Distributed Application Framework with DataSegment API

Kazuki Akamine, Shinji Kono, 琉球大学理工学研究科 情報工学専攻, Information Engineering Course, Faculty of Engineering Graduate School of Engineering and Science, University of the Ryukyus.

2.2 Federated Linda とは

Federated Linda は Linda サーバーを複数台、相互に接続することによって、分散プログラミングを実現する。各サーバーは、接続した Linda サーバー内のタブルスペースへデータの in/out を行うことによって、データを伝搬する。

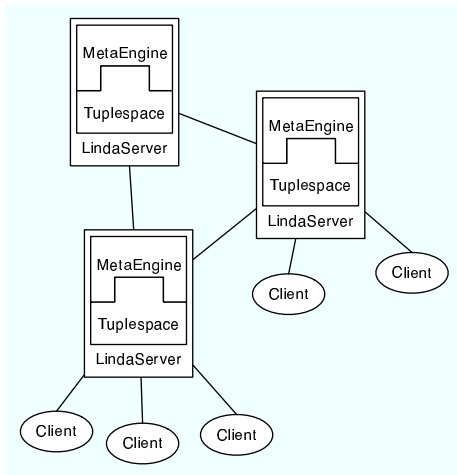


図 1 Federate Linda の接続モデル。組み込まれた Meta Engine がタブルスペースを操作し、外部のサーバーへデータを伝搬する

2.3 Meta Engine とは

Federated Linda は、サーバー間に設置された、Protocol Engine と呼ばれるプログラムによって、タブルスペースの操作や、他サーバーへのタブルの伝搬などを行っており、タブルスペースとは別のプロセスとして、サーバー上に存在していた。しかし、別のプロセスであるため、タブルスペースへのアクセスには同一サーバー上であっても、ソケット通信を用いていた。そこで、本研究室では、Meta Engine と呼ばれるプログラムを提案し実装してきた。Meta Engine は、タブルスペースと同一プロセス上に組み込まれた Protocol Engine である。(図??) すなわち、タブルスペースと同じメモリ空間にあるため、ソケット通信を用いることなく直接 Linda の API を使用して、

タブルスペースにアクセスすることが出来る。

2.4 Federated Linda の問題点

大きく分けて分散アプリケーションは、次の 3 つのパートで構成することができる。

1. **Configuration** 各ノードの接続やルーティングを行う。
2. **ProtocolEngine** Database へアクセスし、他のノードにデータを伝搬する。
3. **Database** 通信した内容や計算結果などの各種データの保存領域。

しかし、FederatedLinda を利用したプログラマーが記述できる部分は ProtocolEngine だけであるため、Configuration に関するコードをそこに記述しなくてはならない。そのため、各ノード間の接続やルーティングを利用者が管理する必要が出てくる。また、FederatedLinda の ProtocolEngine には、次の 2 つの実行方法がある。

1. **Polling base** メインループで定期的に ProtocolEngine を実行し、TupleSpace からデータを取得し、確認する。
2. **Callback function base** とある Tuple が更新される度にその Tuple に設定された Callback function が実行される。

Polling base の場合は、通信が行われる都度、記述した Tuple の内容をチェックするため、負荷が大きくなる。そのため、Callback function base を用いて記述する方が効率はよくなる。しかし、Callback function 同士の繋がりがツリー状に構成されるため、利用者がそのツリー同士の接続管理を行わなくてはならない。また、Callback function 間のデータの共有も難しい。Callback function で記述されたコード群をシーケンシャルに読むことも難しくなる。

3 まとめと今後の課題

謝辞

参考文献

- [1] test