Vol.2013-ARC-205 No.6
Vol.2013-OS-125 No.6
2013/4/25

分散フレームワーク Alice の DataSegment の更新に関する改良

杉 本 優[†1] 河 野 真 治[†2]

Alice は、データを Data Segment、タスクを Code Segment という単位に分割して記述する分散フレームワークである。Alice を使用して水族館の例題を通して分散フレームワークとしての記述能力を確認した。一方で、bitonic sort などの並列処理では、Data Segment の更新のオーバーヘッドが大きいことがわかった。本論文ではこれを解決する手法として Data Segment の flip を提案し、効果の測定を行った。

Yu SUGIMOTO[†1] and Shinji KONO [†2]

Alice is an distributed programming framework, which uses Data Segment and Code Segment as programming units. We checked Alice has an ability to write distributed program using aquarium example. In bitonic sort example refined Data Segment update is slow. So We propose a new API “flip” to solve the problem and show its variation.

1. 研究背景と目的

本研究室では、データを Data Segment、タスクを Code Segment という単位に分割して記述する分散ネットフレームワーク Alice の開発を行なっている。Alice はノード間の Data Segment の送受信 API が提供されている。メニーコアのマシンが主流になっている背景から SEDA Architecture[3] を採用しており、マルチコア上でのスループットの向上を期待している。

以前、Alice が分散フレームワークとしての記述能力を確認するために、水族館の例題[8] の作成を行った。その結果より、Alice には分散プログラムを記述するのに必要な API が備わっていることが確認できている。また、並列環境に対応していることを確認するため、bitonic sort を作成した。しかし、Data Segment の更新のオーバーヘッドにより、期待した効果を得られなかった。

本研究では Data Segment の更新オーバーヘッドを解決する手段として新しく API を提案し、効果の測定を行う。

2. 分散ネットフレームワーク Alice

Alice[6] は、本研究室で開発を行なっている分散管理フレームワークである。Cell 用の並列フレームワーク Cerium[2][7] に似たタスク管理機構と Linda[1] を相互接続した分散フレームワークである Federated Linda[4][5] に似た Data Semgnet の通信構造をもつ。

まず、Alice を使用するに必要な Data Segment、Code Segment について説明を行う。

2.1 Data Segment API

Data Segment は数値や文字列などのデータを構造体的に保持する。Alice は Data Segment をデータベース的に扱う。しかし、Alice は通常のデータベースとは異なり Key 毎にキューがある。Data Segment はキューに put した順番に取得することができる。これは Linda に準じた設計となっている。

Data Segment を管理するのが Data Segment Manager(以下 DSM) である。ノード毎にキーを持ち他のノードには Remote DSM 経由でアクセスすることができる。つまり、Remote DSM は他のノードの Local DSM の proxy である。(図 1) 他のノードに対するアクセスはキューによって、ノード内部で逐次化される。それ以外は、すべて Java の Thread pool により並列実行される。 以下が用意されている Data Segment API である。これらを用いて Data Segment の送受信を行う。

- `void put(String key, Value val)`
- `void update(String key, Value val)`
- `void peek(Receiver receiver, String key)`
- `void take(Receiver receiver, String key)`

2.1.1 put

`put` はデータを追加するための API である。Key

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

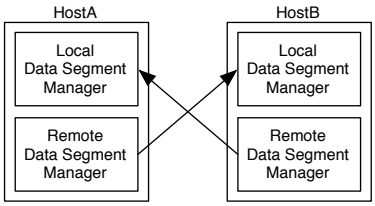

図 1 Remote DSM は他のノードの Local DSM の proxy

Value Store のキューに追加される。 (図 2)

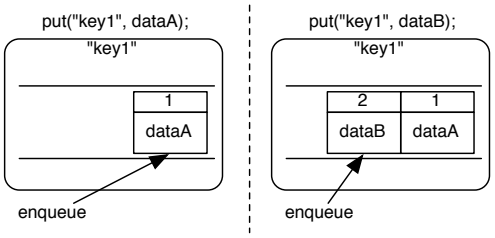

図 2 put はデータを追加する

2.1.2 update

update はデータを更新するための API である。キューの先頭を置き換える。(図 3)

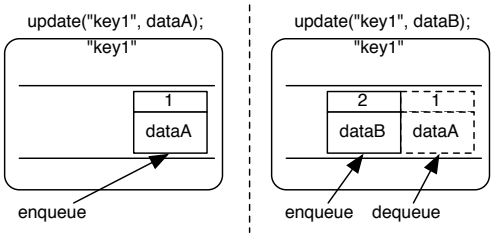

図 3 update はキューの先頭を書き換える

2.1.3 peek

peek はデータを取得する。(図 4)

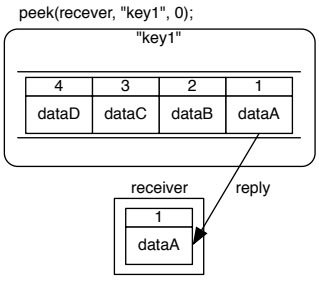

図 4 peek はデータを取得する

Data Segment が無ければ Code Segment の待ち合わせが起きる。(図 5)

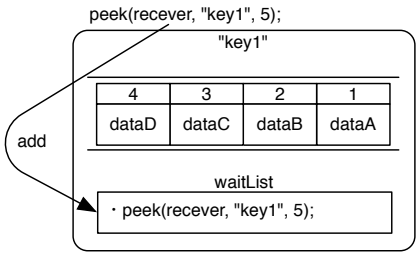

図 5 希望のデータが無いときは保留する

put、update により Data Segment の更新があれば、peek が直ちに実行され、待ち合わせを行なっていた Code Segment が active queue に移される。

2.1.4 take

take もデータを取得するための API である。(図 6)peek との違いは取得された Data Segment は Key Value Store のキューから取り除かれる。

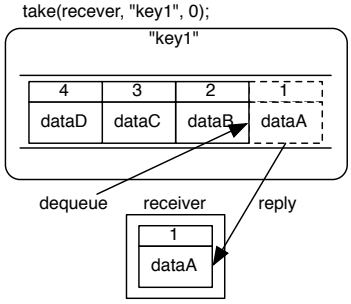

図 6 take はデータを読み込む

2.2 Code Segment

Code Segment はタスクのことである。Code Segment をユーザーが記述するときに Code Segment 内で使用する Data Segment の作成を記述する。Code Segment には Input Data Segment と Output Data Segment を作る API が存在する。

Input Data Segment は Remote か Local を指定する必要がある。さらに、どの Data Segment を取るかをキーで指定する。実際に setKey のメソッドを呼んだ時に指定する。Code Segment が active になるためには、Input Data Segment が全て揃う必要がある。

Output Data Segment も Remote か Local を指定する。Input と同様にどのキーに対して Data Segment を追加するか指定する。実際には put または update メソッドを呼ぶタイミングでおこなう。

これらの Input/Output が Code Segment 間の依存関係を自動的に記述することになる。

2.2.1 Code Segment の実行方法

Alice には、Start Code Segment (ソースコード 1) という C の main に相当するような最初に実行される Code Segment がある。

Vol.2013-ARC-205 No.6
Vol.2013-OS-125 No.6
2013/4/25

ソースコード 1　StartCodeSegment の例

```
1  public class StartCodeSegment extends CodeSegment {
2
3      @Override
4      public void run() {
5      System.out.println("run␣StartCodeSegment");
6
7      TestCodeSegment cs = new TestCodeSegment();
8      cs.input1.setKey("local", "key1");
9
10     System.out.println("create␣TestCodeSegment");
11
12     ods.update("local", "key1", "String␣data");
13     }
14
15 }
```

Start Code Segment はどの Data Segment にも依存しない。つまり Input Data Segment を持たない。この Code Segment を main メソッド内で new し、execute メソッドを呼ぶことで実行を開始させることができる。(ソースコード 2)

ソースコード 2　Start Code Segment を実行させる方法

```
1  public class TestLocalAlice {
2      public static void main(String args[]) {
3      new StartCodeSegment().execute();
4      }
5  }
```

2.2.2　Code Segment の記述方法

Code Segment をユーザーが記述する際には Code Segment を継承して記述する。(ソースコード 3) その CodeSegment は InputDataSegmentManager と OutputDataSegmentManager を利用することができる。

InputDataSegmentManager は Code Segment の `ids` というフィールドを用いてアクセスする。

ソースコード 3　CodeSegment の例

```
1  public class TestCodeSegment extends CodeSegment {
2      Receiver input1 = ids.create(CommandType.PEEK);
3
4      @Override public void run() {
5      System.out.println("index␣=␣" + input1.index);
6      System.out.println("data␣=␣" + input1.val);
7
8      if (input1.index == 10) System.exit(0);
9
10     TestCodeSegment cs = new TestCodeSegment();
11     cs.input1.setKey("local", "key1", input1.index);
12     ods.update("local", "key1", "String␣data");
13     }
14 }
```

- `Receiver create(CommandType type)`

create でコマンドが実行された際に取得される Data Segment が格納される受け皿を作る。引数には CommandType が取られ、指定できる CommandType は `PEEK` または `TAKE` である。

- `void setKey(String managerKey, String key)`

setKey メソッドにより、どこの Data Segment のある key に対して peek または take コマンドを実行させるかを指定することができる。コマンドの結果がレスポンスとして届き次第 Code Segment は実行される。

OutputDataSegmentManager は Code Segment の `ods` というフィールドを用いてアクセスする。OutPutDataSegmentManager は `put` または `update` を実行することができる。

- `void put(String managerKey, String key, Value val)`
- `void update(String managerKey, String key, Value val)`

3.　Message Pack

Data Segment のデータの表現には Message Pack を利用している。Message Pack に関して Java におけるデータ表現は以下の 3 つある。

- 一般的な `Java` のクラスオブジェクト
- `MessagePack for Java` の `Value` オブジェクト
- `byte[]` で表現されたバイナリ

Data Segment は、MessagePack for Java の Value オブジェクトを用いてデータが表現されている。MessagePack は Java のように静的に型付けされたオブジェクトではなく、自己記述なデータ形式である。MessagePack for Java の Value オブジェクトは MessagePack のバイナリにシリアライズできる型のみで構成された Java のオブジェクトである。そのため、Value も自己記述式のデータ形式になっている。

Value オブジェクトは通信に関わるときには、シリアライズ・デシリアライズを高速に行うことができる。ユーザーはメソッドを用いてオブジェクト内部のデータを閲覧、編集することができる。

ユーザーが一般的なクラスを IDL(Interface Definition Language) のように用いてデータを表現することができる。この場合、クラス宣言時に@Message というアノテーションをつける必要がある。ただし、MessagePack で扱うことのできるデータのみをフィールドに入れなければならない。

4.　現状の Alice の問題点

Alice を用いた例題を通して、様々な問題点が明らかになった。API のシンタックス的問題、永続性の問題、実行速度の問題など解決すべき問題は多々ある。特に実行速度の問題では分散環境をテストする例題として作成された Ring の例題 (トポロジーを円状に構成し、メッセージが 1 周する時間を計測する) ではシングルスレッドで実装されている Federated Linda に実行速度で及ばない。また、並列環境をテストする例題として作成した bitonic sort の例題も期待した結果を得ることができなかった。そこで、実行速度の改善

Vol.2013-ARC-205 No.6
Vol.2013-OS-125 No.6
2013/4/25

を行うために、オーバーヘッドの洗い出しを行った。その結果以下のような原因が見つかっている。

HashMap の多用　Alice では Data Segment Manager のマネージャーキーによる探索、Data Segment のキーによる探索など HashMap を多用している。この探索を行う際に排他制御を行うためかかる時間が多少ではあるが、オーバーヘッドになっていると予想される。

Message Pack の convert / unconvert　現在の実装では put または update を呼ぶと Message Pack により Value 型へと変換が行われている。また、peek,take で取得した Data Segment に対して Code Segment 内で Value 型から変換元の型への変換を行う必要がある。この作業もオーバーヘッドになる。また、配列の要素数が増えると変換に時間が多くかかるので、この作業はできるだけ避けたい。

SEDA Architecture　Federated Linda に比べて、通信のレスポンスが遅い原因には SEDA Architecture が挙げられる。SEDA とはマルチコアスレッドを用いて大量の接続を管理し、受け取ったデータを処理ごとに分けられたステージと呼ばれるスレッドに投げ、処理が終わると次のステージにデータを伝搬させて行く処理方式である。しかし、SEDA はスループット重視の実装であり、レスポンス重視ではない。逆に多段のパイプラインのせいでレスポンスは遅れてしまう。また、データを次のステージへ伝搬させる際に LinkedBlockingQueue を使用しているが enqueue 時、dequeue 時にロックを掛けるのでオーバーヘッドの原因と思われる。LinkedBlockingQueue は片方向の連結リストをベースとした Queue 実装である。enqueue / dequeue の操作時の排他制御にはそれぞれ別々のロックオブジェクトが使用されている。そのため、enqueue と dequeue が重なってもロック解除待ちは発生しないが、そのかわりに連結リストの Node オブジェクトの生成操作などが発生してしまうため、enqueue 操作の処理コストが高い。さらに、非力なマシーンでは SEDA の効果を得られず、スレッドを切り替えが頻繁に起こりオーバーヘッドになってしまう。

Data Segment の再構成　取得された Data Segment に変更を行い Key Value Store へ追加する際に、Output Data Segment が毎回新しく作成される。そして、変更された値のコピーが行われる。このコピーに時間がかかっているのではないかと推測される。この無駄なコピーを無くすことで速度改善ができるのではないかと思われる。

5. 改 善 案

5.1 Message Pack

Alice では Data Segment を Value 型という、Message Pack が提供している型で保存している。Value というクラスは動的に型付けされたオブジェクトを表現することができるため、Ruby や Python のような動的型付けの言語のオブジェクトと同様の扱いをすることができる。分散プログラムのアプリケーションはサーバとクライアントの Version が同じとは限らない。サーバ側が更新され、扱う Data Segment が変更された場合であっても、旧 Version との互換性が要求される。Alice は、この問題を Message Pack の Value 型を用いることで互換性をもたせようとしている。しかし、Version の問題が起こらない Local の場合、Data Segment を Value 型に変換し、また任意の型に戻すという操作を行う必要はなく、この操作は単なるオーバーヘッドにしかならない。

従って、Data Segment の送信先が Remote であるならば Value 型に変換を行い、Local であるならば変換しないという具合に改善をすれば、Local における Message Pack のオーバーヘッドを減らすことができる。

5.2 SEDA Architecture

Local においては put や peek に沿った Command を作成するステージ (Code Segment が実行されているスレッド)、受け取った Command を処理するステージ、Code Segment に Data Segment をセットするステージの三段のパイプラインで構成されている。これを全て同一のステージにまとめ、Local の環境下において SEDA を使用せずに処理を行うように変更する。

5.3 Data Segment の再構成

Data Segment の更新におけるオーバーヘッドを減らす方法として Cerium でも良好な結果を得ている flip を提案する。Cerium における flip は Input Data Segment と Output Data Segment を swap させる API である。(ソースコード 4)

ソースコード 4　Cerium における flip

```
1 void swap() {
2    void * tmp = readbuf;
3    readbuf = writebuf;
4    writebuf = tmp;
5 }
```

`readbuf` が Input Data Segment、`writebuf` が Output Data Segment である。Output が input の書き換えならば swap を行い、2 つの領域を入れ替えることで無駄な memcopy を防ぐことができる。

Alice においても Cerium と同様に flip を実装することで、無駄なデータのコピーを防ぐ。(ソースコード 5)

Cerium において Output Data Segment は Task が実行された段階ですでに用意されている。そのためデータを Output Data Segment に書き込む前に flip を呼ぶ。Alice では put または update を呼んだ段階で Output Data Segment が作られるため、ソースコー

Vol.2013-ARC-205 No.6
Vol.2013-OS-125 No.6
2013/4/25

ソースコード 5　Alice における flip

```
1  public class OutputDataSegment {
2      public void flip(Receiver receiver) {
3          DataSegment.getLocal().putObject(receiver.key,
               receiver.getObj());
4      }
5  }
```

ソースコード 6　flip の使用例

```
1  public class SortPhase extends CodeSegment{
2      private Receiver info = ids.create(CommandType.PEEK
           );
3
4      public SortPhase(String key){
5          info.setKey(key);
6      }
7
8      @Override
9      public void run() {
10         DataList list1 = info.asClass(DataList.class);
11         Sort.quickSort(list1);
12         ods.flip(info);
13     }
14 }
```

ド 6 のように Input Data Segment である Receiver を flip メソッドに引数として渡すことで、無駄なコピーを減らす。

6. 実　　験

6.1 実 験 環 境

SEDA がコア数の少ないマシーンではうまく動作しないことを考慮して、メニコア環境で実験を行った。

表 1　実行環境の詳細

CPU	Intel(R) Xeon(R) X5650 @2.67GHz
物理コア数	12
論理コア数	24
CPU キャッシュ	12MB
Memory	16GB

6.2 実 験 概 要

今回それぞれの改善案の効果を調査するために以下の 3 つの実験を行った。

6.2.1 SEDA の有無

Local から Data Segment を取得する Code Segment を 10000 回実行される時間を計測する。SEDA を使用した場合と、しない場合の 2 つの比較を行い、その効果を測定する。

表 2　SEDA の有無の比較

SEDA	あり	なし
実行時間 (ms)	27.72	7.53

SEDA を使わずにコマンドを処理する方が約 3.7 倍差が見られた。(表 2)

6.2.2 flip の効果の測定

Local に Data Segment を 10000 回追加するのにかかる時間を計測する。flip コマンドを使用して追加する場合と、put コマンドを使用して追加する場合の 2 つの比較を行う。

表 3　flip の結果

Command	flip	put
実行時間 (ms)	61.12	65.24

flip を使う方が若干ではあるが速度改善が見られる。(表 3)

6.2.3 bitonic sort における効果の測定

bitonic sort により、100 万の要素をもつ配列の Sort にかかる時間を計測する。分割数は 10 個で行った。

表 4　bitonic sort の結果

	改善前	改善後
実行時間 (ms)	199.38	184.64

6.3 考　　察

実験の結果より今回の改善により、約 10%程 Alice の速度改善を行うことができた。(表 4) この差のほとんどが SEDA によるものと推測される。LinkedBlockingQueue を使った SEDA の実装は、コストが高くレスポンスを求めるには不向きであることがわかった。Cerium のようにスケジューラーによって Code Segment に Data Segment をセットしたほうがレスポンスを向上できると思われる。

7. ま と め

今回の改善は Alice の Local における並列処理の速度を向上させるためのものであった。その結果約 10%程処理速を改善することができた。しかし、まだまだ十分な速度であるとは言いがたい。別の実験から Code Segment 生成からの実行されるまでに、オーバーヘッドがあるとの実験結果が出ている。Code Segment 側のオーバーヘッドを取り除くことで、更なる速度改善が見込まれる。Local においては SEDA は使用しないが、Remote に Data Segment の更新する際にはまだ SEDA を使用している。今回の実験により Remote においても SEDA を使用しないことでレスポンスの向上が見込まれるので、実験を行い確認したい。Remote の処理速度としては少なくともシングルスレッドの Federated Linda と同等の速度を目指している。

また、Alice が抱える問題は API のシンタックス的な問題や拡張性の問題、永続性の問題などが現在判明している。これらの問題を解決し、Alice が信頼性とスケーラビリティーを持つように改良を行なっていく必要がある。

Vol.2013-ARC-205 No.6
Vol.2013-OS-125 No.6
2013/4/25

参 考 文 献

1) Sudhir Ahuja, Nicholas Carriero, and David Gelernter. Linda and friends IEEE Computer, Aug. 1986.
2) Shinji KONO. Cerium. `http://sourceforge.jp/projects/cerium/`, March 2008
3) Matt Welsh, David Culler, and Eric Brewer. Seda: an architecture for well-conditioned, scalable internet services *SIGOPS Oper. Syst. Rev.* Vol.35, No.5, pp. 230-243, 2001.
4) 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.
5) 赤嶺一樹, 河野真治. Meta Engine を用いた Federated Linda の実験. 日本ソフトウェア科学会第 27 回大会 (2010 年度) 論文集, Sep 2010.
6) 赤嶺一樹, 河野真治.Data Segment API を用いた分散フレームワークの設計. 日本ソフトウェア科学会第 28 回大会 (2011 年度) 論文集, Sep 2011.
7) 多賀野海人, 小林佑亮, 宮國渡, 河野真治 (琉球大). Cell Task Manager Cerium の SPU 内データ管理. 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, April 2009.
8) 多賀野海人, 小林佑亮, 宮國渡, 河野真治 (琉球大). Code Segment と Data Segment によるプログラミング手法. 情報処理学会プログラミング・シンポジウム, January 2013.