分散フレームワーク Alice の圧縮機能

照屋 のぞみ[†1]　杉 本　　優[†2]　河 野 真 治[†3]

当研究室ではデータを Data Segment、タスクを Code Segment という単位で分割して記述する手法を提唱しており、そのプロトタイプとして並列分散フレームワーク Alice を開発している。　本研究では Alice に実用的なアプリケーションを作成するために必要な動的なトポロジーを管理する機能と Alice の制御を行えるメタ計算を追加した。また、通信時における Data Segment の多態性を実現するため、Data Segment を Object 型、MessagePack を使った ByteArray 型、圧縮された ByteArray 型の３つの形式で表現できるメタ計算の設計と実装、性能評価を行った。

Nozomi TERUYA,[†1] Yu SUGIMOTO[†2] and Shinji KONO [†3]

Alice is a prototype framework for distributed programming, which uses Data Segment and Code Segment as programming units. We checked Alice has an ability to write dis- tributed program using aquarium example, distributed database Jungle and share screen system AliceVNC . In this paper, we add functions which control dynamic topology and Alice computation. And we show Alice has an ability to write useful application. Furthermore we improve Alice performance. So Alice works 12

1. 研究背景と目的

当研究室ではデータを Data Segment、タスクを Code Segment という単位で分割して記述する並列分散フレームワーク Alice の開発を行っている。Alice では分散環境の構築に必要な処理を Meta Computation として提供することで、スケーラブルな分散プログラムを信頼性高く記述できる環境を実現している。

Alice が分散プログラムを記述する能力を有することは確認された。だが、実用的な分散プログラムを作成するためには、ノードの切断・再接続時に対応した処理を行いたい場合や圧縮されたデータ形式で通信を行いたい場合がある。

本研究では、 Alice の Computation の 制御を行う Meta Computation を実装した。プログラムに Alice の制御を行うメタプログラムを記述することにより切断や再接続の状況に応じた処理を元のコードを変更することなく指定することができる。また、圧縮機能を持った DataSegmentManager を追加することにより Data Segment の多態性を実現した。そして、TreeVNC と TreeVNC を Alice を用いて実装した AliceVNC の比較をコードの観点から評価を行った。

†1 琉球大学工学部情報工学科
Information Engineering, University of the Ryukyus.
†2 琉球大学大学院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.
†3 琉球大学工学部情報工学科
Information Engineering, University of the Ryukyus.

2. 分散フレームワーク Alice の概要

2.1 Data Segment と Code Segment

Alice はデータを Data Segment、（以下 DS）タスクをと Code Segment（以下 CS）という単位に分割してプログラミングを行う。DS は Alice が内部にもつデータベースによって管理されている。DS に対応する一意の key が設定されており、その key を用いてデータベースを操作する。

CS は実行に必要な DS が揃うと実行されるという性質を持ち、入力された DS に応じた結果が出力される。CS を実行するために必要な入力 DS は InputDS、CS が計算を行った後に出力される DS は Output DS と呼ばれる。データの依存関係にない CS は並列実行が可能であるため、並列度を上げるためには CS の処理内容を細かく分割して依存するデータを少なくするのが望ましい。

2.2 Data Segment

複数のスレッドから１つのデータに変更を行うためには、データの不整合を防ぐための lock が必要になる。複数の関係のない要素を１つのデータオブジェクトで表現した場合、全ての操作で lock が必要になる。この lock がスケラビリティーを低下させる。つまり

データのサイズも並列計算には重要である。

Alice はデータを細かく分割して記述する。その細かく分割されたデータを DS と呼ぶ。実際には特定のオブジェクトにマッピングされ、マッピングされたクラスを通してアクセスされる。

2.3 Data Segment Manager

DS は実際には queue に保存される。queue には対になる key が存在し、key の数だけ queue が存在する。この key を指定して DS の保存、取得を行う。queue の集合体はデータベースとして捉えられる。このデータベースを Alice では DS Manager（以下 DSM）と呼ぶ。DSM には Local DSM と Remote DSM が存在する。Local DSM は各ノード固有のデータベースである。Remote DSM は他のノードの Local DSM の proxy であり、接続しているノードの数だけ存在する。（図 1）Remote DSM に対して書き込むと対応するノードの Local DSM に書き込まれる。

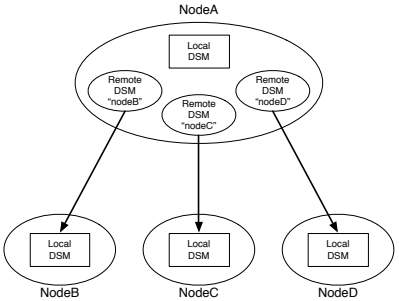

図 1 Remote DSM は他のノードの Local DSM の proxy

2.4 Data Segment API

以下の Data Segment API を用いてデータベースにアクセスする。put と update は DS を追加する際に、peek と take は DS を取得する際に使用する。

- `void put(String managerKey, String key, Object val)`

DS を queue に追加するための API である。第一引数で指定した DSM の中の、第二引数に対応する queue に対して DS を追加している。

- `void update(String managerKey, String key, Object val)`

update も queue に追加するための API である。put との違いは、先頭の DS を削除してから DS を追加することである。そのため API 実行前後で queue の中にある DS の個数は変わらない。

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

take は DS を読み込むための API である。読み込まれた DS は削除される。要求した DS が存在しなければ、CS の待ち合わせ （Blocking）が起こる。put や update により DS に更新があった場合、take が直ちに実行される。

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

peek も DS を読み込む API である。take との違いは読み込まれた DS が削除されないことである。

3. Code Segment

Alice 上で実行されるタスクの単位が CS である。ユーザーは CS を組み合わせることでプログラミングを行う。CS をユーザーが記述する際に、内部で使用する DS の作成を記述する。

Input DS と Output DS は CS に用意されている API を用いて作成する。Input DS は、Local か Remote か、また key を指定する必要がある。CS は、記述した Input DS が全て揃うと Thread pool に送られ、実行される。

Output DS も Local か Remote か、また key を指定する必要がある。Input の場合は setKey を呼ぶ際、Output の場合は put(または update) の際にノードと key の指定を行っている。しかし、どの時点でノードと key の指定を行えばよいか、どのような API を用意するべきかは、議論の余地がある。

4. Code Segment の記述方法

CS をユーザーが記述する際には CS を継承して記述する (ソースコード 1 ,2)。継承することにより Code Segment で使用する API を利用する事ができる。

Alice には、Start CS (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 CS がある。Start CS はどの DS にも依存しない。つまり Input DS を持たない。この CS 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 1 は、5 行目で次に実行させたい CS（ソースコード 2）を作成している。8 行目で Output DSM を通して Local DSM に対して DS を put している。Output DSM は CS の `ods` というフィールドを用いてアクセスする。Output DSM は `put` と `update` を実行することができる。TestCodeSegment はこの”cnt”という key に対して依存関係があり、8 行目で update が行われると TestCodeSegment は実行される。

```
public class StartCodeSegment extends
      CodeSegment {

    @Override
    public void run() {
        new TestCodeSegment();

        int count = 0;
        ods.update("local", "cnt", count);
    }

}
```

Code 1　StartCodeSegment の例

```
public class TestCodeSegment extends
      CodeSegment {
    private Receiver input1 = ids.create(
          CommandType.TAKE);

    public TestCodeSegment() {
        input1.setKey("local", "cnt");
    }

    @Override
    public void run() {
        int count = input1.asInteger();
        System.out.println("data␣=␣" + count);

        if (count == 10)
            System.exit(0);

        new TestCodeSegment();
        ods.update("local", "cnt", ++count);
    }
}
```

Code 2　CodeSegment の例

ソースコード 2 は、0 から 10 までインクリメントする例題である。2 行目で取得された DS が格納される受け皿を作る。Input DSM がもつ create メソッド使うことで作成できる。

- `Receiver create(CommandType type)`

引数には CommandType が取られ、指定できる CommandType は `PEEK` または `TAKE` である。Input DSM は CS の `ids` というフィールドを用いてアクセスする。

4 行目から 6 行目はコンストラクタである。コンストラクタはオブジェクト指向のプログラミング言語で新たなオブジェクトを生成する際に呼び出されて内容の初期化を行う関数である。

TestCodeSegment のコンストラクタが呼ばれた際には、

(1) TestCodeSegment が持つフィールド変数 Receiver input1 の定義が行われる。
(2) 次に CS のコンストラクタが呼ばれ、CS が持つフィールド変数の定義と初期化が行われる。
(3) `ids.create(CommandType.TAKE)` が行われ、input1 の初期化が行われる。
(4) 最後に TestCodeSegment のコンストラクタの 5 行目が実行される。

5 行目は Input DSM がもつ setKey メソッドにより Local DSM から DS を取得している。

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

setKey メソッドにより、どの DSM 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できる。コマンドの結果がレスポンスとして届き次第 CS は実行される。

run メソッドの内容としては 10 行目で取得された DS を Integer 型に変換して count に代入している。16 行目で もう一度 TestCodeSegment の CS が作られる。17 行目で count の値をインクリメントして Local DSM に値を追加する。13 行目が終了条件であり、count の値が 10 になれば終了する。

4.1　Computation と Meta Computation

Alice の Computation は、key で指し示される DS を待ち合わせて CS を実行させると定義できる。それに対して、Alice の Meta Computation は、Alice の Computation を支えている Computation のプログラミングと定義できる。

例えば、トポロジーを指定する API は Meta Computation である。Alice が動作するためにはトポロジーを決める必要がある。つまりトポロジーの構成は Alice の Computation を支えている Computation とみなすことができる。トポロジーが決定するとそのトポロジーを構成する計算が行われる。トポロジーを指定する API はその構成の計算をプログラミングして変更するものである。他にも再接続の動作を決める API や切断時の動作を決める API は Meta Computation である。

これらの Meta Computation が Alice の Computation に影響することはない。プログラマーは CS を記述する際にトポロジーや切断、再接続という状況を予め想定した処理にする必要はない。プログラマーは目的の処理だけ記述する。そして、切断や再接続が起こった場合の処理を記述し Meta Computation で指定する。このようにプログラムすることで、通常処理と例外処理を分離することができるため、シンプルなプログラムを記述できる。

5.　Meta Data Segment

DS は、アプリケーションに管理されているデータのことである。アプリケーションを構成する CS によってその値は変更される。それに対して Meta DS は、分散フレームワーク Alice が管理しているデータである。Alice を構成する CS によってのみ、その値は変更される。一部の Meta DS はアプリケーションに利用することができる。

例えば、"start" という key では、ノードが Start CS を実行可能かどうかの状態を表す。他にも "_CLIST" という key では、利用可能な Remote DS の一覧が

管理されている。ユーザーはこの一覧にある名前を指定することで、動的に DS の伝搬などを行うことができる。

また、Input DS に付随しているものもある。Input DS は CS 内部で Receiver という入れ物に格納される。ユーザーは、Receiver に対して操作することで DS を入手できる。この Receiver には、from というフィールドがあり、この DS を誰が put したという情報が入っている。この情報をデータの伝搬する際に利用することで、DS を put したノードに送り返すことを防ぐことができる。

Meta DS は DS 同様に DS API を用いて取得できる。

6. Meta Code Segment

CS はアプリケーションを動作させるために必要なタスクであり、ユーザーによって定義される。それに対して Meta CS は Alice を構成するタスクである。つまり Meta CS の群は Alice の Computation と言い換えることができる。一部のみユーザーが定義をすることができ、Alice の挙動を変更することができる。

7. AliceVNC

AliceVNC は、当研究室で開発を行っている TreeVNC を Alice を用いて実装された、授業向け画面共有システムである。Alice が実用的なアプリケーションを記述する能力をもつことを確認するために作成した。

授業で VNC を使う場合、１つのコンピュータに多人数が同時につながるため、性能が大幅に落ちてしまう (図 2)。この問題をノード同士を接続させ、木構造を構成することで負荷分散を行い解決したものが TreeVNC である (図 3)。TreeVNC は、TightVNC のソースコードを利用して開発されている。

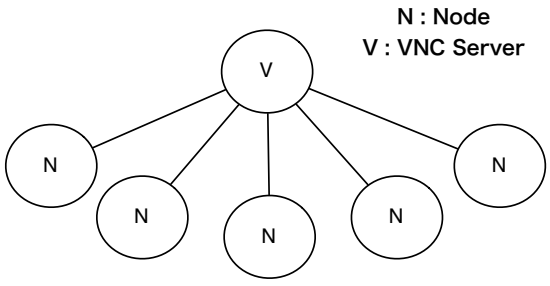

図 2 VNC の構造

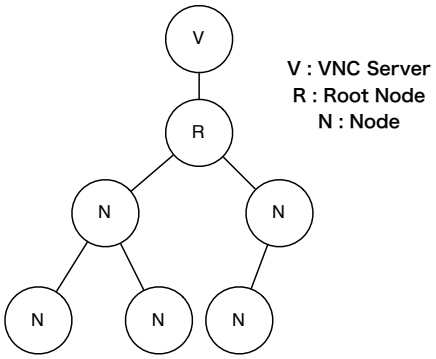

図 3 TreeVNC, AliceVNC の構造