分散フレームワークChristieの設計

2018/2/20

照屋のぞみ 並列信頼研

Christieの設計目標

・ 安定したネットワークサービスを提供するためには、分散プログラムに信頼性とスケーラビリティが求される
・信頼性とは定められた環境下で安定して仕様に従った動作を行うこと
・仕様の記述のしやすさ、可読性
・拡張時に仕様変更を抑えられること
・スケーラビリティとはサービス利用者が増加したとき単純にノードを追加するだけで線形に性能を向上させる能力

DS/CSに基づく分散フレームワーク

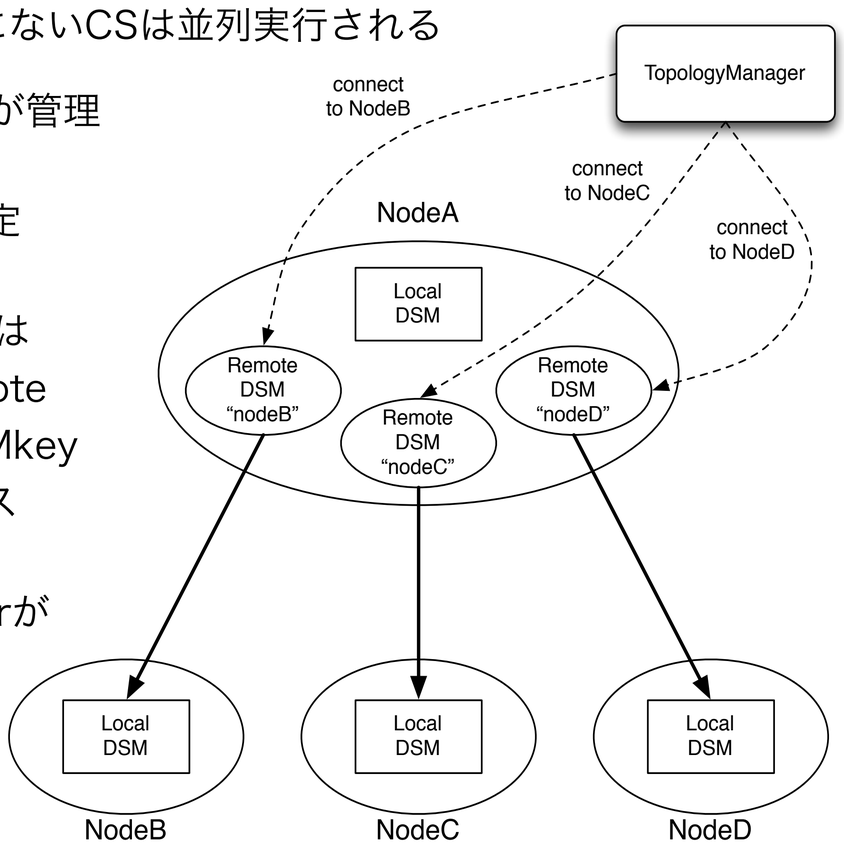
・データを Data Segment(DS) 、タスクを Code Segment(CS) という単位に分割して依存関係を記述することでプログラミングする
・CSはInput DS(入力されるDS)とOutput DS(出力されるDS)を持ち、keyで指定したInputDSが全て揃うと実行される。
・データの依存関係にないCSは並列実行される

・DSはDSManagerが管理

・DSはkeyにより指定

・他ノードのDSMへは proxyであるRemote DSMを立ててDSMkey を指定してアクセス

・TopologyManagerが ノードを構成する



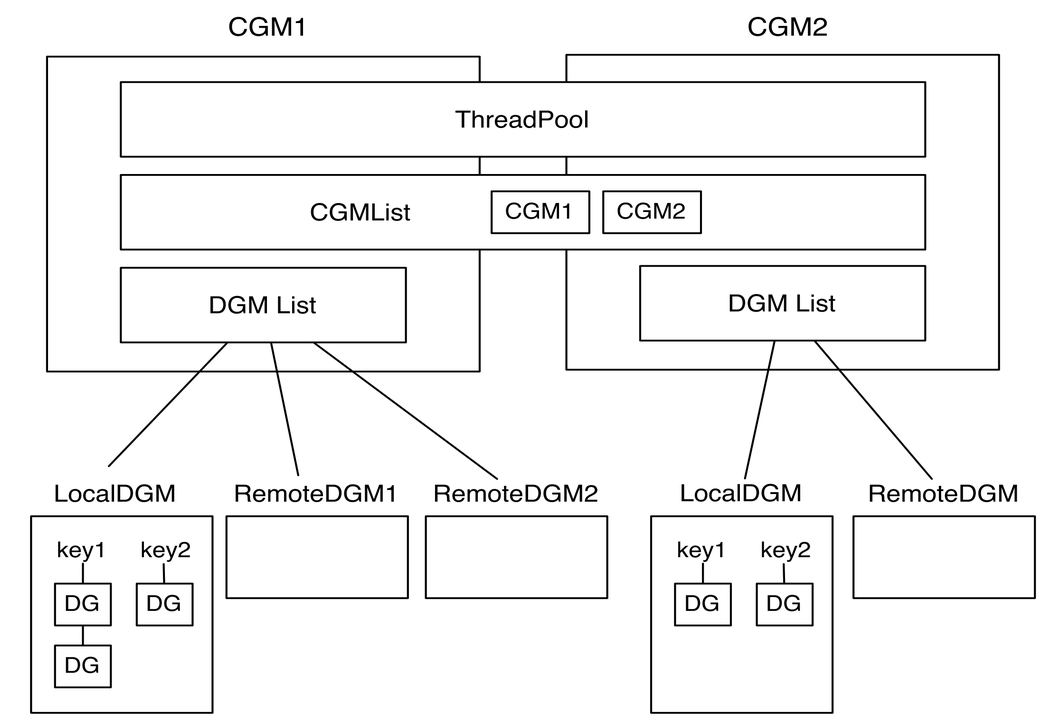
従来実装(Alice)の問題点

- **LocalDSMがstaticで書かれており複数立ち上げできない**
  - NAT越えなどの機能拡張が困難
  - 分散のテストが困難
- **APIシンタックスの分離**
  - setKeyがCS外からも呼べる
  - CSのコードを見てもどのkeyで待ち合わせを行っているか不明
  - createとsetKeyの順序を間違えるとエラーとなる
- **DSの型の整合性を実行時にしかチェックできない**
  - Receiver型は任意の型をputで格納できる
  - データを取り出すときはasClassで型を指定する
  - この整合性をコンパイル時にはチェックできない

```
public class TestCodeSegment extends CodeSegment {
    private Receiver input1 = ids.create(CommandType.PEEK);
    private Receiver input2 = ids.create(CommandType.TAKE);

    public TestCodeSegment() {
        input1.setKey("local", "count");
        input2.setKey("remote", "name");
    }

    @Override
    public void run() {
        int count = input2.asClass(Integer.class);
        int name = input2.asClass(String.class);
        System.out.println(count + " : " + name);

        new TestCodeSegment();
    }
}
```

Christieの基本設計

・CodeGear(CG)/DataGear(DG)でプログラミング
・CodeGearManagerがDataGearManagerを管理
・CGMごとにLocalDGMがあり、CGMは複数立ち上げ可能
・CGMをCG間で引数で持ち歩くことで、CGMからCGを作るAPIにアクセスする
・複数のLocalDGM同士のやりとりはRemoteDGMを介してアクセス



ChristieのAPIシンタックス

・Javaのアノテーションを用いてInputDGの指定
 - アノテーションは必ずフィールドにつけなければならない
 - InputDGの生成、key指定、Take/Peekの指定の分離を防ぐ
・Reveiverではなく変数を直接書く
 - 変数名がそのままkeyとして扱われる
 - CSを見ればInputDGの型が分かる
 - 型を内部で保存するため、型を指定して取り出す必要がない

```
public class TestCodeGear extends CodeGear {

    @Take
    int count;

    @TakeFrom("remote")
    String name;

    @Override
    public void run(CodeGearManager cgm){
        System.out.println(count + " : " + name);
        cgm.setup(new TestCodeGear());
    }

}
```

まとめと今後の課題

・LocalDGMの複数立ち上げを可能にし、テストや機能拡張がしやすい環境を整えた
・アノテーションを用いたシンタックスによる信頼性の高い記述を実現
・１つのCS内での型の整合性を保証できた

・今後の課題
 - コンパイル時の型の整合性の保証
 - TopologyManagerの実装
 - Aliceとの速度比較
 - Jungleデータベースとの統合
 - GearsOSへの移行