修士 (工学) 学位論文

Master's Thesis of Engineering

分散フレームワークChristieの設計

Design of Distributed framework Christie

2018年3月

March 2018

照屋 のぞみ

NOZOMI TERUYA

琉球大学

大学院理工学研究科

情報工学専攻

Information Engineering Course

Graduate School of Engineering and Science

University of the Ryukyus

指導教員：教授　和田 知久

Supervisor: Prof. Tomohisa WADA

本論文は、修士 (工学) の学位論文として適切であると認める。

論　文　審　査　会

印

(主　査)　　和田　知久

印

(副　査)　　岡崎　威生

印

(副　査)　　名嘉村　盛和

印

(副　査)　　河野　真治

要 旨

スマートフォンやタブレット端末の普及率が増加している。それに伴いインターネット利用者数も増加しており、ネットワーク上のサービスには、信頼性とスケーラビリティーが要求される。ここでいう信頼性とは、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を指す。またスケーラビリティーとは、スケーラビリティとは、分散ソフトウェアに対して単純にノードを追加するだけで性能を線形的に上昇させることができる性質である。しかし、これらをもつ分散プログラムをユーザーが一から記述することは容易ではない。

これらの問題の解決のために、当研究室ではデータを Data Segment、タスクを Code Segment という単位で記述するプログラミング手法を導入した分散フレームワーク Alice の開発を実現した。Data Segment は整数や文字列や構造体などの基本的なデータの集まりである。Code Segment は入力となる Data Segment が全て揃ったら処理を開始し計算結果の Data Segment を出力するタスクである。

Alice が実用的な分散アプリケーションを記述でき、仕様の変更を抑えた信頼性の高い拡張を可能にするということは、水族館の例題や TreeVNC の例題から確認された。しかし、Alice では API 設計が直感的でなく、型の整合性がとれない問題があった。また、Alice に NAT 越えの機能を実装しようとした際、Data Segment Manager が 1 つしか持てないたために拡張が困難であることが分かった。

本研究では、Alice から得られた知見をもとに、分散フレームワーク Christie の設計を行った。

Abstract

目 次

図 目 次

表 目 次

リスト目次

第1章　分散プログラミングの信頼性向上

スマートフォンやタブレット端末の普及率が増加している。それに伴いインターネット利用者数も増加しており、ネットワーク上のサービスには、信頼性とスケーラビリティーが要求される。ここでいう信頼性とは、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を指す。またスケーラビリティーとは、スケーラビリティとは、分散ソフトウェアに対して単純にノードを追加するだけで性能を線形的に上昇させることができる性質である。しかし、これらをもつ分散プログラムをユーザーが一から記述することは容易ではない。

これらの問題の解決のために、当研究室ではデータを Data Segment、タスクを Code Segment という単位で記述するプログラミング手法を導入した分散フレームワーク Alice の開発を実現した。Data Segment は整数や文字列や構造体などの基本的なデータの集まりである。Code Segment は入力となる Data Segment が全て揃ったら処理を開始し計算結果の Data Segment を出力するタスクである。

Alice が実用的な分散アプリケーションを記述でき、仕様の変更を抑えた信頼性の高い拡張を可能にするということは、水族館の例題や TreeVNC の例題から確認された。しかし、Alice では API 設計が直感的でなく、型の整合性がとれない問題があった。また、Alice に NAT 越えの機能を実装しようとした際、Data Segment Manager が 1 つしか持てないたために拡張が困難であることが分かった。

本研究では、Alice から得られた知見をもとに、分散フレームワーク Christie の設計を行う。

第2章　分散フレームワーク Alice の概要

2.1　CodeSegment と DataSegment

Alice では Code Segment（以下 CS）と Data Segment（以下 DS）の依存関係を記述することでプログラミングを行う。

CS は実行に必要な DS が全て揃うと実行される。CS を実行するために必要な入力される DS のことを InputDS、CS が計算を行った後に出力される DS のことを Output DS と呼ぶ。

データの依存関係にない CS は並列実行が可能である（図 2.1 ）。CS の実行において DS が他の CS から変更を受けることはない。そのため Alice ではデータが他から変更され整合性がとれなくなることはない。

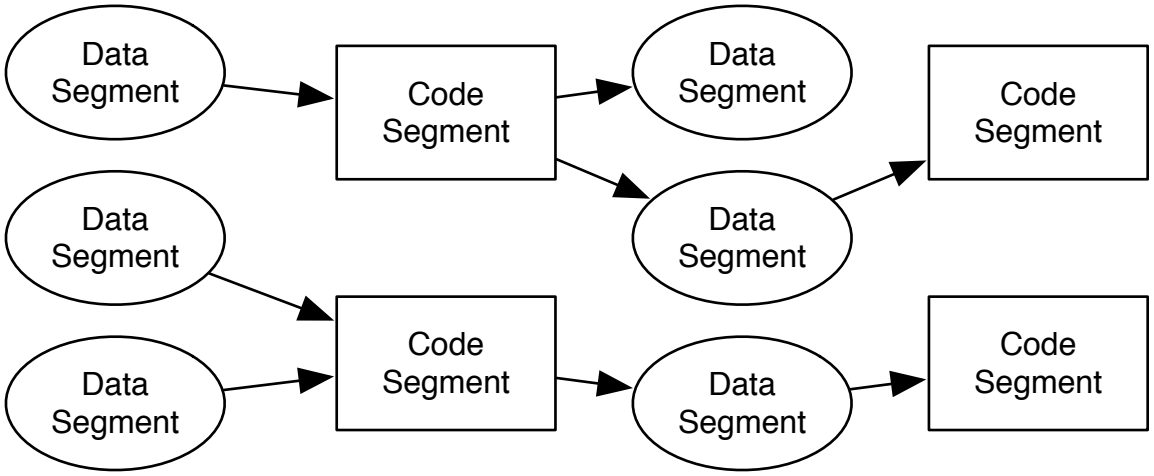

図 2.1: CodeSegment の依存関係

Alice は Java で実装されており、DS は Java Object に相当する。CS は Runnable な Object(void run() を持つ Object) に相当する。プログラマが CS を記述する際は、CodeSegment クラスを継承する。

DS は数値や文字列などの基本的なデータの集まりを指し、Alice が内部にもつデータベースによって管理されている。このデータベースを Alice では DS Manager と呼ぶ。

CS は複数の DS Manager を持っている。DS には対になる String 型の key が存在し、それぞれの Manager に key を指定して DS にアクセスする。一つの key に対して複数の DS を put すると FIFO 的に処理される。なので Data Segment Manager は通常のデータベースとは異なる。

2.2 DataSegmentManager

DS Manager（以下 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 は他ノードの Local DSM に対応する proxy であり、接続しているノードの数だけ存在する（図 2.2 ）。他ノードの Local DSM に書き込みたい場合は Remote DSM に対して書き込めば良い。

Remote DSM を立ち上げるには、DataSegment クラスが提供する connect メソッドを用いる。接続したいノードの ip アドレスと port 番号、そして任意の Manager 名を指定することで立ちあげられる。その後は Manager 名を指定して Data Segment API を用いて DS のやり取りを行うため、プログラマは Manager 名さえ意識すれば Local への操作も Remote への操作も同じ様に扱える。

2.3 Data Segment API

DS の保存・取得には Alice が提供する API を用いる。
put と update、flip は Output DS API と呼ばれ、DS を DSM に保存する際に用いる。
peek と take は Input DS API と呼ばれ、DS を DSM から取得する際に使用する。

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

DS を DSM に追加するための API である。第一引数は Local DSM か Remote DSM かといった Manager 名を指定する。そして第二引数で指定された key に対応する DS として第三引数の値を追加する。

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

update も DS を DSM に追加するための API である。put との違いは、queue の先頭の 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。

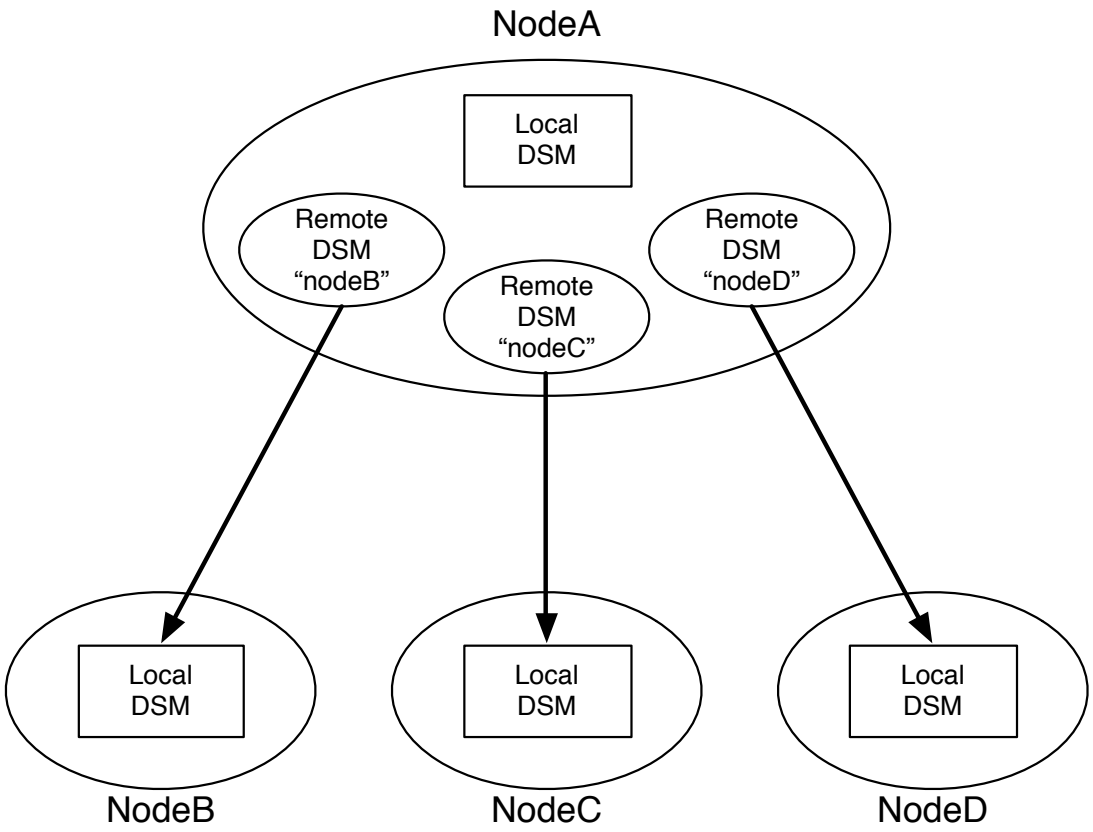

図 2.2: Remote DSM は他のノードの Local DSM の proxy

- `void flip(String managerKey, String key, Receiver val)`

flip は DS の転送用の API である。取得した DS に対して何もせずに別の Key に対し保存を行いたい場合、一旦値を取り出すのは無駄である。flip は DS を受け取った形式のまま転送するため無駄なコピーなく DS の保存ができる。

- `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。

2.4 CodeSegment の記述方法

CS をユーザーが記述する際には CodeSegment クラスを継承して記述する (ソースコード ?? , ??)。

継承することにより Code Segment で使用する Data Segment API を利用する事ができる。

Alice には、Start CS (ソースコード ??) 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 **??** は、5 行目で次に実行させたい CS（ソースコード **??** ）を作成している。8 行目で Output DS API を通して Local DSM に対して DS を put している。Output DS API は CS の `ods` というフィールドを用いてアクセスする。`ods` は `put` と `update` と `flip` 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 put が行われると TestCodeSegment は実行される。

CS の Input DS は、CS の作成時に指定する必要がある。指定は CommandType(PEEK か TAKE)、DSM 名、そして key よって行われる。Input DS API は CS の `ids` というフィールドを用いてアクセスする。Output DS は、`ods` が提供する put/update/flip メソッドをそのまま呼べばよかったが、Input DS の場合 `ids` に peek/take メソッドはなく、create/setKey メソッド内で CommandType を指定して実行する。

ソースコード**??**は、0 から 9 までインクリメントする例題である。2 行目では、Input DS API がもつ create メソッドで Input DS を格納する受け皿 (Receiver) を作っている。引数には `PEEK` または `TAKE` を指定する。

- `Receiver create(CommandType type)`

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。

5 行目は、2 行目の create で作られた Receiver が提供する setKey メソッドを用いて Local DSM から DS を取得している。

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

setKey メソッドは peek/take の実行を行う。どの DSM のどの 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。

実行される run メソッドの内容は

1. 10 行目で取得された DS を Integer 型に変換して count に代入する。
2. 12 行目で count をインクリメントする。
3. 16 行目で次に実行される CS が作られる。（この時点で次の CS は Input DS の待ち状態に入る）
4. 17 行目で count を Local DSM に put する。Input DS が揃い待ち状態が解決されたため、次の CS が実行される。
5. 13 行目が終了条件であり、count の値が 10 になれば終了する。

となっている。

2.5 TopologyManager

Alice では、ノード間の接続管理やトポロジーの構成管理を、Topology Manager という Meta Computation が提供している。プログラマはトポロジーファイルを用意し、Topology Manager に読み込ませるだけでトポロジーを構成することができる。トポロジーファイルは DOT Language [?] という言語で記述される。DOT Language とは、プレーンテキストを用いてデータ構造としてのグラフを表現するためのデータ記述言語の一つである。ソースコード??は 3 台のノードでリングトポロジーを組むときのトポロジーファイルの例である。

DOT Language ファイルは dot コマンドを用いてグラフの画像ファイルを生成することができる。そのため、記述したトポロジーが正しいか可視化することが可能である。

Topology Manager はトポロジーファイルを読み込み、参加を表明したクライアント（以下、Topology Node）に接続するべきクライアントの IP アドレスやポート番号、接続名を送る（図 2.3)。また、トポロジーファイルで lavel として指定した名前は Remote DSM の名前として Topology Node に渡される。そのため、Topology Node は Topology Manager の IP アドレスさえ知っていれば自分の接続すべきノードのデータを受け取り、ノード間での正しい接続を実現できる。

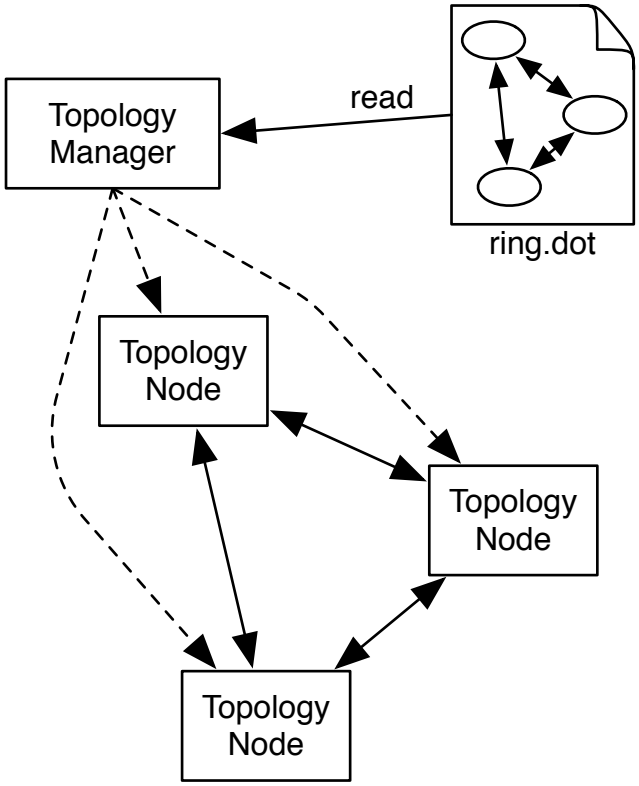

図 2.3: Topology Manager が記述に従いトポロジーを構成

また、実際の分散アプリケーションでは参加するノードの数が予め決まっているとは限らない。そのため Topology Manager は動的トポロジーにも対応している。トポロジーの種類を選択して Topology Manager を立ち上げれば、あとは新しい Topology Node が参加表明するたびに、Topology Manager から Topology Node に対して接続すべき Topology Node の情報が put され接続処理が順次行われる。そして Topology Manager が持つトポロジー情報が更新される。現在 Topology Manager では動的なトポロジータイプとして二分木に対応している。

第3章　Aliceの問題点

3.1　直感的でない API

3.2　型がわからない

3.3　DataSegmentManager を複数持てない

3.3.1　TopologyManager の NAT 超え機能

3.3.2　Jungle のテスト

第4章　分散フレームワークChristieの設計

4.1　Christieの基本設計

4.2　APIの改善

4.3　DataGearManagerの複数立ち上げ

第5章　Christieの評価

5.1　Akka

5.2　Corba

5.3　Erlang

5.4　Hazelcast

第6章　まとめ

第7章　今後の課題

第8章　謝辞

本研究の遂行、また本論文の作成にあたり、ご多忙にも関わらず終始懇切なる御指導と御教授を賜わりました河野真治准教授に深く感謝したします。

そして、数々の貴重な御助言と技術的指導を戴いた伊波立樹さん、並びに並列信頼研究室の皆様、本研究を遂行するにあたり参考にさせていただいた先行研究の Alice, Federated Linda, Jungle, TreeVNC の設計・実装に関わった全ての方々に感謝いたします。

また、本フレームワークの名前の由来となったクリスティー式戦車の生みの親、ジョン・W・クリスティーに敬意を評します。

最後に、日々の研究生活を支えてくださった米須智子さん、情報工学科の方々、そして家族に心より感謝いたします。

参考文献

[1] Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and Types*. Cambridge University Press, New York, NY, USA, 1989.

[2] Joachim (mathématicien) Lambek and P. J. Scott. *Introduction to higher order categorical logic*. Cambridge studies in advanced mathematics. Cambridge University Press, Cambridge, New York (N. Y.), Melbourne, 1986.

[3] Michael Barr and Charles Wells. *Category Theory for Computing Science*. International Series in Computer Science. Prentice-Hall, 1990. Second edition, 1995.

[4] Eugenio Moggi. Notions of computation and monads. *Inf. Comput.*, Vol. 93, No. 1, pp. 55–92, July 1991.

[5] M. P. Jones and L. Duponcheel. Composing monads. Research Report YALEU/DCS/RR-1004, Yale University, December 1993.

[6] Tokumori Kaito and Kono Shinji. The implementation of continuation based c compiler on llvm/clang 3.5. *IPSJ SIG Notes*, Vol. 2014, No. 10, pp. 1–11, may 2014.

[7] 信康大城, 真治河野. Continuation based c の gcc4.6 上の実装について. 第 53 回プログラミング・シンポジウム予稿集, 第 2012 巻, pp. 69–78, jan 2012.

[8] The agda wiki. `http://wiki.portal.chalmers.se/agda/pmwiki.php`. Accessed: 2016/01/20(Fri).

[9] Welcome to agda's documentation! — agda 2.6.0 documentation. `http://agda.readthedocs.io/en/latest/index.html`. Accessed: 2016/01/31(Tue).

[10] Welcome! — the coq proof assistant. `https://coq.inria.fr/`. Accessed: 2016/01/20(Fri).

[11] Ats-pl-sys. `http://www.ats-lang.org/`. Accessed: 2016/01/20(Fri).

[12] Spin - formal verification. http://spinroot.com/spin/whatispin.html. Accessed: 2016/01/20(Fri).

[13] Nusmv home page. http://nusmv.fbk.eu/. Accessed: 2016/01/20(Fri).

[14] The cbmc homepage. http://www.cprover.org/cbmc/. Accessed: 2016/01/20(Fri).

[15] Opencl — nvidia developer. https://developer.nvidia.com/opencl. Accessed: 2016/02/06(Mon).

[16] Cuda zone — nvidia developer. https://developer.nvidia.com/cuda-zone. Accessed: 2016/02/06(Mon).

[17] 翔平小久保, 立樹伊波, 真治河野. Monad に基づくメタ計算を基本とする gears os の設計. Technical Report 16, 琉球大学大学院理工学研究科情報工学専攻, 琉球大学工学部情報工学科, 琉球大学工学部情報工学科, may 2015.

[18] 徳森海斗. Llvm clang 上の continuation based c コンパイラ の改良. Master's thesis, 琉球大学 大学院理工学研究科 情報工学専攻, 2016.

[19] 小久保翔平. Code segment と data segment を持つ gears os の 設計. Master's thesis, 琉球大学 大学院理工学研究科 情報工学専攻, 2016.

[20] 河野 真治比嘉 健太. Continuation based c を用いたプログラムの検証手法.

[21] Benjamin C. Pierce. *Types and Programming Languages*. The MIT Press, 1st edition, 2002.

[22] B.C. Pierce. 型システム入門プログラミング言語と型の理論:. オーム社, 2013.

[23] Ulf Norell. Dependently typed programming in agda. In *Proceedings of the 4th International Workshop on Types in Language Design and Implementation*, TLDI '09, pp. 1–2, New York, NY, USA, 2009. ACM.

[24] John Backus. The history of fortran i, ii, and iii. *SIGPLAN Not.*, Vol. 13, No. 8, pp. 165–180, August 1978.

[25] Peter J. Landin. The mechanical evaluation of expressions. *Computer Journal*, Vol. 6, No. 4, pp. 308–320, January 1964.

[26] Alonzo Church. *The Calculi of Lambda-Conversion*. Princeton University Press, Princeton, New Jork, 1941.

[27] P. Hudak, S. Peyton Jones, and P. Wadler (editors). Report on the Programming Language Haskell, A Non-strict Purely Functional Language (Version 1.2). *ACM SIGPLAN Notices*, Vol. 27, No. 5, May 1992.

[28] N.G de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the church-rosser theorem. *Indagationes Mathematicae (Proceedings)*, Vol. 75, No. 5, pp. 381 – 392, 1972.