[ポスター発表] 研究報告

プログラマブルスイッチを用いたクラスタ構築のためのコンパイラ設計

着本光大[1]　柏崎礼生[2]　井口信和[3,4]

Compiler Design for Build Cluster Using Programmable Switch

1. はじめに

プログラマブルスイッチ等のプログラマブルなネットワークデバイスの登場により, 独自のネットワークプロトコルをハードウェア (以下, データプレーン) 上に実装することが可能となった. データプレーン上に実装することで, CPU 上での実行速度と比較しより高速な動作が期待できる. またネットワークプロトコルだけではなく, In-Network Computing と呼ばれるアプリケーション処理をデータプレーン上にオフロードする試みもある. ムーアの法則の終焉により CPU の性能は頭打ちとなっており, データプレーンへのオフロードはアプリケーションの高速化につながる有効な手段の一つである. 実装例として, DNS[1] や Key-Value Store[2], 機械学習を用いた Firewall[3], 分散合意アルゴリズムの Raft[4] 等がある.

データプレーンには厳しいリソース制約が存在する. パケットを処理するパイプラインステージの数は限られており, 複雑なプログラムの場合は最後まで実行できない可能性がある. この問題はパケットを再循環させることで解決できるがスループットは低下する. またステートフルメモリがわずかしかないため保持できるグローバル変数が限られる.

そこでリソース制約を, 複数のプログラマブルスイッチを用いたクラスタ構成で解決することを検討する. クラスタ構成とは, 複数台のノードを接続し 1 つのシステムを構築することで, 拡張性や可用性を得ることができる構成である. プログラムを分離しクラスタ構成で動作させることで, プログラマブルスイッチ単体では実行できなかったプログラムのオフロードが可能となる. しかしクラスタを構築するには, リソース制約を考慮しながらデータプレーンプログラムを分離する必要がある. またデータプレーンプログラミング言語は, パケット処理に特化しているためアプリケーション開発には向いていない. そのためクラスタ構成の開発難易度は高い. そこで本研究では, C, C++, Rust 等の汎用プログラミング言語で書かれたプログラムを, リソース制約に応じてプログラムを分離し複数のデータプレーンプログラムに変換するコンパイラ NeD(以下, NeD) を設計する. データプレーンプログラミング言語には P4(Programming Protocol-independent Packet Processors)[*1]を使用する. NeD を用いることで, 汎用プログラミング言語で開発したアプリケーションをクラスタ構成で動作させることが可能となる.

2. P4

P4 は PISA に基づいて, どのようにパケットを処理するかを記述するドメイン固有言語である. PISA(Protocol-Independent Switch Architecture) は, Parser, Match-Action, Deparser から構成されるプログラマブルスイッチのためのパケットパイプランアーキテクチャである. Parser は受信パケットを読み込み, 定義したヘッダフィールドと一致するか照合する. Match-Action はテーブルを参照しマッチ条件を元にアクションを決定する. マッチ条件には最長一致 (Longest Prefix Matching) や完全一致を選択できる. Deparser は処理したパケットを再構築し指定したポートから送信する.

P4 にはデバイス毎に固有のアーキテクチャ定義が提供されており, それらを用いることでステートフルメモリやカウンタにアクセスすることができる.

[1] 近畿大学大学院総合理工学研究科
Graduate School of Science and Engineering Research, Kindai University
[2] 国立情報学研究所
National Institute of Infomatics
[3] 近畿大学理工学部情報学科
Department of Infomatics, Faculty of Science and Engineering Research, Kindai University
[4] 近畿大学情報学研究所
Cyber Infomatics Research Institute, Kindai University

[*1] https://p4.org/

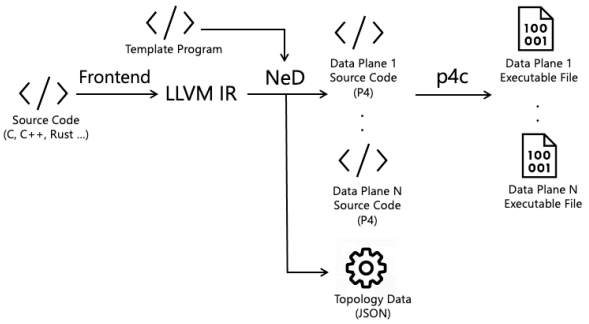

図 1 NeD アーキテクチャ
Fig. 1 NeD Architecture

3. NeD

3.1 概要

NeD のアーキテクチャを**図 1** に示す. NeD は LLVM IR とテンプレートプログラムを入力とし, P4 プログラムとトポロジデータを出力するコンパイラである.

LLVM IR はコンパイラ基盤である LLVM[*2]で用いられる中間表現である. LLVM IR を入力とすることで, LLVM のフロントエンドとして実装されているプログラミング言語を入力として扱うことが可能となる. また LLVM が持つ最適化機構を利用することで, プログラム内の不要な命令を検出しコンパイル時に削除することができる.

テンプレートプログラムは, フォワーディング処理のみを実装した P4 プログラムである. NeD はアプリケーションのオフロードに焦点を当てているため, フォワーディング処理は生成しない. NeD は与えられたテンプレートプログラムに, LLVM IR から生成したアクションを埋め込み P4 プログラムとして出力する.

トポロジデータはクラスタを構築するためのネットワークトポロジと, ノードと P4 プログラムの対応付けをした JSON ファイルである.

3.2 NeD ヘッダ

NeD は `NeD_Header` と `NeD_Body` の 2 種類のヘッダ定義を生成する.

`NeD_Header` は, `end_flag`, `id`, `address`, `loop_count` の 4 つのフィールドを持つ. `end_flag` はプログラムが終了したかどうかを示す 1bit のフラグである. パケット受信時にフラグが 1 だった場合, プログラムを終了したと見なしパケットをフォワーディングする. `id` は NeD パケットを判別するための識別子である. `address` はパケット受信時に実行するアクションを判別するための識別子である. `loop_count` はループ処理実行時に, 現在のループ回数を保存するフィールドである. 無限ループを防止するために, コンパイル時に指定した最大ループ回数に `loop_count` が達したときパケットは破棄される. `id`, `address`, `loop_count` のフィールドサイズはコンパイル時に決定する.

`NeD_Body` はアクションの引数や戻り値をフィールドに持つヘッダであり, 外部からのアクション呼び出しに用いる. そのためフィールドは, アクションが生成されるコンパイル時に決定する.

3.3 コンパイルプロセス

まずプログラム全体の使用メモリ容量, 命令数から必要ノード数を決定する. 次にプログラムを分離しオフロード先のノードを選択する. プログラムは関数単位で分離される. このときリソース制約を満たせない関数がある場合, 制約を満たすまで関数を分離する. ノード選択は, プログラム全体でノードをまたぐ関数呼び出しが最小回数になるように選択される. これはノード間通信のオーバーヘッドを最小化するためである. NeD は関数呼び出しの依存関係を抽出し, 呼び出し回数の多いペアを同一ノードに配置する.

ノード選択が完了後, LLVM IR 命令を P4 命令へと変換する. 関数はアクションとして扱われる. アクションにはマッチ条件としてアドレスが割り当てられ, アドレスとアクションを対応付けたテーブルエントリが生成される. 生成したアクションの引数と戻り値をフィールドにした `NeD_Body` ヘッダも同時に生成される. 生成した P4 プログラムはテンプレートプログラムに埋め込まれ出力される.

4. まとめ

本研究ではデータプレーンのリソース制約の解決のために, 複数のプログラマブルスイッチを用いたクラスタ構成を検討した. そしてクラスタ構成を自動で構築するために, 汎用プログラミング言語から複数の P4 プログラムを生成するコンパイラ NeD を設計した.

今後は設計したコンパイラの動作確認実験を予定している. また NeD で生成したクラスタをシミュレータ上で実行し, スループット等の性能評価実験の実施を検討している.

参考文献

[1] Woodruff, J., Ramanujam, M. and Zilberman, N.: P4DNS: In-Network DNS, 2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), pp.1-6, (2019).
[2] Tokusashi, Y., Matsutani, H. and Zilberman, N.: LaKe: The Power of In-Network Computing, 2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig), pp.1-8, (2018).
[3] Qin, Q., Poularakis, K., Leung, K. K. and Tassiulas, L.: Line-Speed and Scalable Intrusion Detection at the Network Edge via Federated Learning, 2020 IFIP Networking Conference (Networking), pp.352-360, (2020).
[4] Zhang, Y., Han, B., Zhang, Z., and Gopalakrishnan, V.: Network-Assisted Raft Consensus Algorithm, SIGCOMM Posters and Demos 2017, pp.94-96, (2017).

[*2] `https://llvm.org/`