Computer Science Tripos - Part II Project Proposal

NetCache: An accelerated, network-assisted Key-Value Store

Thanh Bui

Supervisor: Dr Noa Zilberman

Department of Computer Science
University of Cambridge

Originator: Dr Noa Zilberman

Director of Studies: Dr Graeme Jenkinson

Project Overseers: Dr Markus Kuhn, Dr Ewal and Dr Nada Amin

Downing College October 19, 2018

Introduction

Background

In-network computing is an emerging research area in systems and networking, where applications traditionally running on the host are offloaded to the network hardware (e.g., switch, network interface card (NIC)). Examples of applications offloaded in the past include consensus protocols (NetPaxos[1]), sorting and even a game (Tic-Tac-Toe). Key-Value Store (KVS) is also among the popular type of in-network applications. In particular, the performance of KVS applications is shown to be very sensitive to load within the data center and latency. As the load increases, it becomes more difficult to maintain KVS applications' performance without exceeding the host capacity.

Therefore, it is particularly interesting, and indeed useful, to be able to offload KVS applications to the network, as this will reduce the load placed on the host, as well as allow the application to scale to a distributed system.

The Project

The goal of this project will be to offload the KVS application to run on a NetFPGA platform. The main idea is to perform as much computation as possible in the FPGA.s

For this project, I will be using the P4 programming language[2]. It is a language designed to allow the programming of packet forwarding planes. Besides, unlike general purpose languages such as C or Python, P4 is domain-specific with a number of constructs optimized around network data forwarding, hence is well suited to such a network application.

Starting Point

Platform & Language

This project will work mainly with a NetFPGA SUME[3] board, using P4 programming language. Apart from the platform and the language, I will be building the application from scratch.

[1]https://dl.acm.org/citation.cfm?id=2774999
[2]https://p4.org/
[3]https://github.com/NetFPGA/NetFPGA-SUME-public/wiki

I have no prior experience with either NetFPGA or P4, but this will be mitigated through self-learning in which I will make use of the online tutorials, Google's resources and the P4 community documentation, as well as the experience of my supervisors.

Computer Science Tripos

The relevant Tripos courses that can serve as a starting point for this project are primarily: *Computer Networking* and *Principles of Communications*. Since the courses are introductory, I will also consult Part III's *High Performance Networking* course. I also plan to bridge any knowledge gap through extensive personal reading as well as help from my project supervisors.

Resources Required

For this project I will be using my own computer, a 2017 MacBook Pro with a 2.3 GHz Intel Core i5 processor and 16 GB of RAM, that runs macOS Mojave. I accept full responsiblity for this machine, and I have made contingency plans to deal with hardware and/or software failures. Should that machine suddenly fail, I have another 16Gb of RAM computer with a 2.6 GHz Intel Core i7 processor that runs Ubuntu 18.04 LTS. If all else fails, I can continue to work on an MCS machine. Backups will be done weekly to Microsoft OneDrive and my external hardrive, and all my codes will be uploaded to GitHub for version control.

For the hardware prototype, I will require a NetFPGA SUME board. I will need access to a machine that has the SUME installed, and a 10G NIC. These will be supplied by my supervisor. I will be using my own machine for development, with remote access to a server with the SUME board inside.

I will also require access to the lab network in order to *ssh* to the server with the SUME board. Development on my machine will require VPN access, for using floating licenses of the development tools. Lastly, I will require a second server for testing some of the extensions of the project.

Work to be Done

The main core component of the project is to be able to run the KVS on the NetFPGA board. In order to do that, the following sub-tasks must be done:

1. Since the project is done on NetFPGA using P4 programming language, both of which I am not familar with, I first have to study them thoroug‘hly and be proficient working with the platform.

2. The next stage will be to design the architecture for the application. Generally, it will be to try to map the application to a match-action table.

3. The third stage involves implementing the architecture. Firstly, it will be coded and simulate to ensure correctness. Once it runs smoothly in the software, it will then be compiled to the hardware. The aim for this stage is to get a working prototype that uses a single protocol (binary/ascii) and a single slab size.

4. Once a simple prototype is up and working, further extensions need to be done to allows the prototype to support a variety of parameters/conditions (which will be discussed in **Possible Extensions** section).

Success Criteria

This project will be deemed a success if I managed to study and design an architecture for the application, as well as succeeded to implement that design. More concretely:

1. I am able to map the KVS application to a match-action pipeline architecture.

2. I have a working prototype. In other words, my design runs on the hardware.

3. The project is able to support the real application

4. The performance of the KVS is improved.

Possible Extensions

If the core parts of the project are successful and completed within a reasonable time, I shall then try to investigate and implement further extensions. Some possible options are:

1. Extending the application, which currently only supports fixed slab size, to support multiple slab sizes.

2. Extending to support more than one protocol (e.g. ASCII)

3. Extending the application to forward to a host on a miss

4. Considering cache extensions and large-size queries

Timetable

Planned starting date is 19/10/2018.

1. **Michaelmas weeks 2–4 [19/10–5/11]:** Preparatory reading on the P4 programming language and setting up the NetFPGA platform. Going through the tutorials and experimenting with some examples.

2. **Michaelmas weeks 5–6 [6/11–19/11]:** Study the application: its protocols (e.g., binary or ASCII, TCP or UDP) and modes of operation (e.g. set, get, delete, etc.).
   Milestone: Understand the application architecture. Be able to map the application to a match-action pipeline.

3. **Michaelmas weeks 7–8 [20/11–28/11]:** Start implementation of the design: coding and simulation to ensure the architecture works, then transfering to hardware and testing.
   Milestone: Basic design should work with minimal bugs left for the vacation .

4. **Michaelmas vacation:** Finish the design and start writing progress report.
   Milestone: Have a working prototype, a completed progress report and a presentation for demonstration purposes.

5. **Lent weeks 1–2 [17/1–30/1]:** Start working on extending the application to support multiple slab sizes and key sizes.

6. **Lent weeks 3–4 [31/1–13/2]:** Working on extending the application to support another protocol, possibly ASCII.
   Milestone: The prototype continues to work well with the extensions implemented.

7. **Lent weeks 5–6 [14/2–27/2]:** Working on extending the application to forward to a host on a miss.

8. **Lent weeks 7–8 [28/2–13/3]:** Possible overflow from the previous weeks. Clean up codes and repository. Possible consideration of cache extensions and large-size queries.
   Milestone: Completed prototype with core components and at least 3

extensions in place.

9. **Easter vacation:** Writing dissertation main chapters. Possible overflow of the fourth extension.

10. **Easter term 1–2 [25/4–8/5]:** Further evaluation and complete dissertation. Proof reading and then an early submission so as to concentrate on examination revision.

11. **Easter term 3 [9/5–17/5]:** Buffer week.