Extending the Software-defined Network Boundary

Oliver Michel
University of Colorado Boulder
oliver.michel@colorado.edu

Michael Coughlin
University of Colorado Boulder
michael.coughlin@colorado.edu

Eric Keller
University of Colorado Boulder
eric.keller@colorado.edu

ABSTRACT

Given that Software-Defined Networking is highly successful in solving many of today's manageability, flexibility, and scalability issues in large-scale networks, in this paper we argue that the concept of SDN can be extended even further. Many applications (esp. stream processing and big-data applications) rely on graph-based inter-process communication patterns that are very similar to those in computer networks. To our mind, this network abstraction spanning over different types of entities is highly suitable for and would benefit from central (SDN-inspired) control for the same reasons classical networks do. In this work, we investigate the commonalities between such intra-host networks and classical computer networking. Based on this, we study the feasibility of a central network controller that manages both network traffic and intra-host communication over a custom bus system.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*packet-switching networks, network topology*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*distributed applications*

General Terms

Design, Performance, Reliability

1. INTRODUCTION

Software-Defined Networking (SDN) has been highly successful in solving many of today's manageability, flexibility, and scalability issues in large-scale networks. In the years since SDN was introduced we have seen the software-defined network boundary extended from the physical network into the hypervisor. We argue that the concept of SDN can be extended even further. We believe that certain systems benefit from SDN-inspired control of all their communication which includes inter-process and inter-thread communication.

SIGCOMM'14, August 17–22, 2014, Chicago, IL, USA.
ACM 978-1-4503-2836-4/14/08.
http://dx.doi.org/10.1145/2619239.2631443.

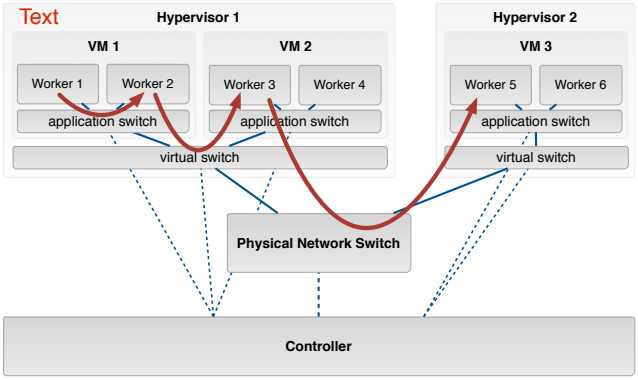

Figure 1: System Architecture

Many applications (*esp.* stream processing and big-data applications) rely on graph-based communication patterns that are very similar to those in computer networks. This network abstraction spanning over different types of entities (threads, processes, virtual hosts, and physical hosts) is highly suitable for and would benefit from central control for the same reasons classical networks do.

In this work, we investigate the commonalities between such intra-host networks and classical computer networking. Based on this, we study the feasibility of a central network controller that manages both network traffic and intra-host communication over a custom or modified bus or message queue system, as illustrated in figure 1. We believe that also in this scenario, central control can be beneficial to system performance, flexibility and elasticity as it allows for dynamic demand-driven re-wiring and re-balancing of graph-based systems regardless whether a given edge is spanning across hosts or is local to a single host.

2. SOFTWARE-DEFINED INTRA-HOST COMMUNICATION

2.1 Stream Processing

While there are several use-cases for extending central communication control into the host, this work is highly motivated by stream processing systems. While existing well-known frameworks like Hadoop focus on batch-style processing and analytics of data, a more recent trend focuses on processing of data generated in real-time [3]. Examples for such systems are Storm, Samza, and S4. In these systems, data-streams traverse a graph consisting of worker instances.

While nodes in this graph may correspond to actual virtual or physical hosts, this is not necessarily the case. Subsequently, edges in such a graph may stay within the boundaries of one host or span across the network.

Despite the fact that most stream processing frameworks have some notion of dynamically re-balancing the processing topology, this functionality is still generally inflexible, is not designed to dynamically respond to processing demand and, most importantly, does not support the dynamic introduction of new classes of workers (*e.g.,* an additional monitoring layer) at runtime. These limitations make it difficult for systems to respond automatically to changes in demand or introduce new functionality without halting the entire system. This is mainly due to static placement of message queue systems like RabbitMQ or Kafka in between workers.

Furthermore, a major challenge that arises with such systems is the influence network performance can have on the rate of processing obtainable. Much research has examined the impact network performance can have on Hadoop-style computation ([4] among others), especially in multi-tenant clouds, and proposed solutions to improve performance [1, 2]. Yet, little work has been done with regard to stream-processing frameworks.

2.2 The case for central control

In order to provide stream processing systems with an ability to change a running data processing graph, improve flexibility and elasticity, we elaborate on the idea of using SDN-inspired control of both network communication and inter-worker communication on a single host. Even though we have fine grained control over the communication between nodes, typically only local and low-level configuration options are available for all communication that stays within the boundaries of a host. We propose extending the software-defined network boundary into the operating system and even applications. The SDN controller can then be used to create network abstractions for particular applications and change properties of the communication graph to react to application's needs.

We believe that a central controller may handle both network communication and communication local to a host. Thereby, we provide a general abstraction for communication patterns independent of whether communication partners are placed on the same host or not. In fact, our system may provide a very similar API to the one used by the most prominent SDN wire protocol OpenFlow. That is a flow abstraction to specify data streams across the network. With this abstraction (which would also seamlessly work with existing SDN technology), it does not matter if a flow between nodes or across the topology is local to a single host or spreads across an entire datacenter.

Going one step further, this abstraction also allows tenants in a datacenter to specify their data-flow graph and the controller may allocate bandwidth, place, and migrate virtual machines to provide the requested topology and characteristics.

Through central control, optimized placement across the entire administrative domain (potentially saving resources and money by giving their placement logic to the datacenter operator) is possible.

2.3 Prototype

Instead of directly integrating this idea into existing stream processing frameworks, we started by implementing a very limited custom prototype of a stream processing framework incorporating central control of the topology and its properties. Our prototype consists of skeleton code for workers, a so-called application switch and the central controller. The controller controls both OpenFlow switches deployed throughout the network and application switches, our messaging mechanism leverages Linux' D-Bus message bus system to pass data between workers. If the next processing worker of a given tuple is not local to the application switch, the data is sent to the appropriate host where it is again handled by an application switch instance.

Based on our early evaluations, our system is able to dynamically (at runtime) migrate workers between hosts, change network properties based on application demands, and insert and remove workers and classes of workers. We believe that this approach may significantly improve flexibility and elasticity of such and other systems that highly rely on intra- and inter-host communication. Furthermore, given that applications are able to dynamically dictate their needs to the network, and more quickly react to increased demand, we expect overall processing performance to increase in many cases.

3. FUTURE DIRECTIONS

In the future, we propose investigating three major areas along the lines of this project. First, we believe it is imperative to extend an existing stream-processing framework such as Storm or Samza with our technology. This would allow us to perform case studies and performance evaluations using typical workloads for these systems. Also, consequences of dynamic, feedback-based reconfiguration and placement such as optimized resource utilization can be explored. Second, we would like to explore which other types of applications may benefit from our idea. We believe that operating systems could directly expose an API to control messaging and communication flows between processes. Third, leveraging the flexibility that our system provides, it is possible to centrally control scheduling, shuffling and partitioning behavior of stream processing applications.

4. REFERENCES

[1] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. Towards Predictable Datacenter Networks. In *Procesdings of SIGCOMM 2011*.

[2] A. D. Ferguson, A. Guha, C. Liang, R. Fonseca, and S. Krishnamurthi. Participatory Networking: An API for Application Control of SDNs. In *Proceedings of SIGCOMM 2013*.

[3] K. Goodhope, J. Koshy, and J. Kreps. Building LinkedIn's Real-time Activity Data Pipeline. *IEEE Data Eng. ...*, 2012.

[4] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz. Runtime measurements in the cloud: Observing, analyzing, and reducing variance. *Proc. VLDB Endow.*, Sept. 2010.