CSE 222A Project Proposal
Coflow Scheduling

Group 6

Ruby Pai Amit Borase Sreejith Unnikrishnan Stas Mushits
Ritvik Jaiswal

October 22, 2015

1 Introduction

Data parallel cluster applications running on fast CPUs are limited by relatively slower network connections. The coflow abstraction for such applications was recently proposed in [1], which argued that identifying the individual parallel data flows associated with an application as a “coflow” and applying application-aware scheduling would improve application performance. Subsequently, Varys, a coflow scheduler was published [3], followed by Aalo, which improves on Varys [2]. Varys is based on heuristics that draw on insights about characteristics of an optimal coflow scheduler that would allow scheduling for minimal coflow completion time (CCT) or to meet a deadline.

The Varys scheduler was shown to improve CCTs by up to 3.16 times on average, and to increase the number of coflows meeting deadlines by a factor of 2 [3]. This is promising not only for application performance, but has implications for issues such as data center energy efficiency, as most of the power wasted in data centers is due to CPUs idling while waiting for the network. However, a significant barrier to usage of coflow schedulers like Varys or Aalo is that they require coflows to be explicitly identified by the programmer to the scheduler via an API. This poses problems in terms of legacy code, programmer burden, and finally, in cases in which the coflows that exist in an application may not be obvious or known.

Therefore, in this project, we propose to examine coflow scheduling in two parts. First, we will replicate the results of [3] using the same trace, which is available on [4] and extend the experiment to traces for different data parallel applications. Second, taking Varys as the established state-of-the-art in coflow scheduling (as the code for Aalo does not appear to be available yet) we will examine the impact of incomplete coflow identification on the scheduler. Examining the robustness of the current state-of-the-art scheduler to incomplete coflow labeling will inform whether automated algorithms for identifying coflows, using for example machine learning techniques, which cannot be expected to be 100% accurate, are viable with current coflow scheduling techniques.

2 Approach

As mentioned Section 1, the proposed project consists of two parts. The first part is to validate results from published literature, extending the scheduling results to new workloads. By examining how the scheduler performs with less than 100% coflow labeling, the second part takes the first step in considering how the published result (the Varys scheduler in particular and coflow schedulers in general) might work with automated coflow identification, which we argue is important for the reasons stated in Section 1.

3 Related Work

So far, we have focused on the seminal work of Chowdhury et al on coflows and coflow scheduling [1, 3, 2]. Though Aalo is the more recent result and the authors argue its overall merits over Varys, the code does not appear to be publically available (we have not contacted the authors). We thus focus on Varys, with a experimental setup that should be easily exendtible to Aalo, as both schedulers are implemented in Scala and expected to work with the coflow simulation code in [4]. Other work on coflow scheduling includes [6], which integrates routing for a performance improvement, as well as other works cited in Section 9 of the August 2015 Aalo paper [2].

Since our primary interest is ultimately in examining the performance of coflow scheduling with less than 100% labeling of coflows, we chose to use Varys, which is authored by the inventors of the coflow concept and for which the code is available. However, since Varys is based on heuristics, it is not currently clear to us what impact this has on the result for all possible coflow schedulers.

We are not currently aware of other work that performs validation of the Varys result and examines the performance of Varys with unidentified coflows, but plan further reading on coflow identification to continue searching for such.

4 Plan

First, we will set up a cluster simulation environment. We anticipate obtaining 10 virtual machines via class resources. We are using Mininet [5], a network emulator that can create arbitrary topologies of hosts, switches, controllers and links, to set up a virtual cluster environment of about 100 hosts, along the lines of Figure 1.

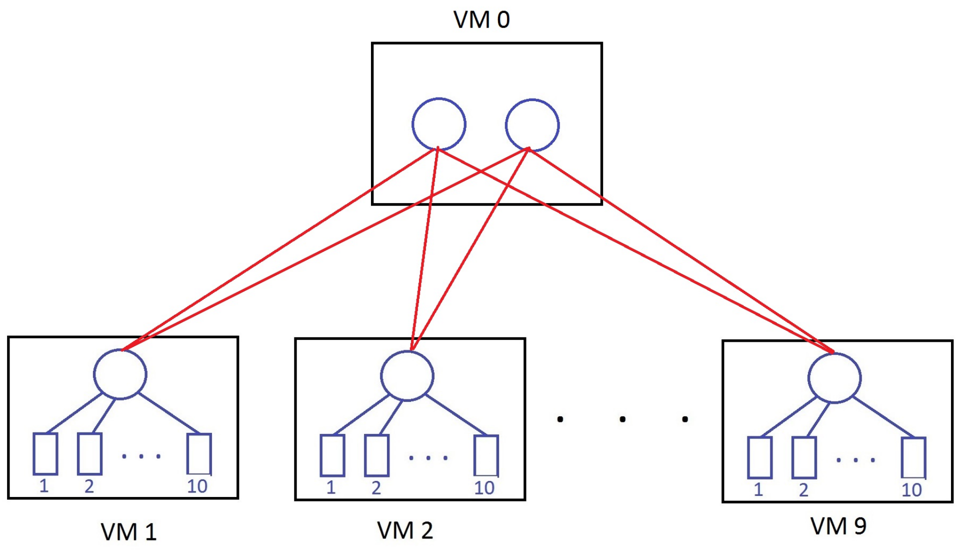

Figure 1: Proposed topology for simulated cluster. Virtual machines are in black; mininet elements are in blue. We are looking into how to implement the inter-VM links in red. The simulation is thus the blue and red elements of switches, hosts and links.

We are still trying to figure out how to implement the inter-VM links in red; GRE tunnels require distinct (source, destination)-tuples, so the two switches in VM 0 would be a problem. Alternatives using GRE would be either to just have a single high level switch, or to implement our entire simulation within one VM. A large number of network elements can be created on a single machine with Mininet, but while we have set up the indivdual VM building blocks to create Figure 1 on the one VM we currently have, we have not examined whether implementing the entire cluster within one VM would work for our purposes. Until we obtain the additional VMs, we will work with a smaller topology within a single VM, which we have set up.

Whatever the final setup selected, data rate must be selected appropriately to ensure that virtualization does not impact our experiment results, and testing performed to verify the correctness of our setup. Also, the scheduler performance metrics were coflow completion time (CCT) and number of coflows meeting their deadline over a per-flow scheduler. We need think carefully about the evaluation of these metrics in our experimental setup.

5 Schedule

By each of the given milestones, we plan to complete the following tasks.

1. Milestones 11/5 (2 weeks) Complete evaluation with the trace available on [4]. We anticipate that once this is done, significant hurdles will have been cleared.

2. Milestone 11/24 (2.5 weeks) Complete evaluation with other traces (11/5, one week). The significant work involved here is gathering other traces and adjusting them to our virtual environment. Also, we will be able to run the simulation without all coflows identified (11/24, one a half weeks).

3. Deliverable 12/8 Poster and Final Report (2 weeks)

We will complete all experiments with less than 100% coflow labelling and evaluate scheduler performance.

References

[1] Mosharaf Chowdhury and Ion Stoica. Coflow: A networking abstraction for cluster applications. In *ACM Hotnets*, 2012.

[2] Mosharaf Chowdhury and Ion Stoica. Efficient coflow scheduling without prior knowledge. In *ACM SIGCOMM*, 2015.

[3] Mosharaf Chowdhury, Yuan Zhong, and Ion Stoica. Efficient coflow scheduling with varys. In *ACM SIGCOMM*, 2014.

[4] `https://github.com/coflow`. Accessed: 2015-10-18.

[5] `mininet.org`. Accessed: 2015-10-18.

[6] Y. Zhao, K. Chen, W. Bai, C. Tian, Y. Geng, Y. Zhang, D. Li, and S. Wang. Rapier: Integrating routing and scheduling for coflow-aware data center networks. In *INFOCOMM*, 2015.