ParVarys

Etesam Ansari (ea2905), Yunlan Li (yl4387)

We plan to parallelize Varys [2] - a novel coflow scheduling algorithm for optimizing average coflow completion time[1] (CCT) in a datacenter environment. A task for datacenter applications could rarely be completed by sending a single request to another datacenter server. For example, web search workloads typically use a partition/aggregator model where a user query will trigger multiple subtasks to search for results in each shard stored on different servers, the results from each shard are then aggregated to compute the final answer to be sent back to the user. Because such tasks involve a collection of TCP flows, optimizing flow completion time (FCT) doesn't necessarily reduce task completion time, which is the latency of user requests and what determines the quality of user experience. This mismatch between the network optimization objective (FCT) and application-level objective (Task Completion Time) has motivated a new abstraction called *coflow*: a collection of flows that share a common performance goal. In the web search example, all flows created for answering the user query would belong to the same coflow. This coflow abstraction aligns the network-level and application-level objective and thus by optimizing CCT, we could observe better application and end-user experience. Therefore, there has been a large body of work in the network community to come up with a near-optimal coflow scheduling algorithm to minimize the average CCT.

The offline coflow scheduling problem is defined as follows:

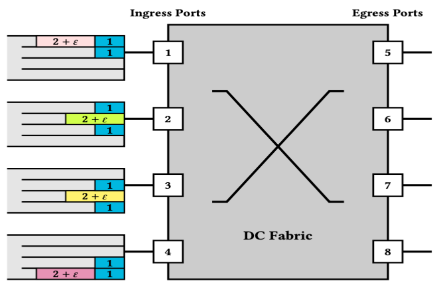

Figure 1: An instance of a coflow scheduling problem, used as a running example in the paper. The datacenter has 4 ingress and egress ports, with each ingress port having a "virtual output queue" for each egress port (see §2.2 for detailed model description). The example has 5 coflows. Coflow C1 has eight flows, with each ingress port sending unit amount of data to two egress ports; coflows C2, C3, C4 and C5 have one flow each sending $2 + \varepsilon$ amount of data to one of the egress ports (the ε amount is only for breaking ties consistently). Coflow C2, C3, C4 and C5 being single flow coflow is only for simplicity; the ε amount of flow could be sent to any egress port without changing the results in §3.

- the datacenter network fabric is abstracted into a single big switch consisting of m ingress ports (NICs) and n egress ports (ToR switches)

- assume all coflows arrive simultaneously at time $t = 0$, and the information about each coflow (number of flows, size, source and destination port of each flow) is all known

[1]

The goal is to find a schedule for the coflows (order, rate) to minimize the average CCT.

[1] CCT: the time it takes for the slowest flow of the coflows to complete. Average CCT is the average CTT of a collection of coflows.

This problem is NP-hard (via reduction from concurrent open-shop scheduling problem). Varys uses a Smallest-Effective-Bottleneck-First (SEBF) heuristic to produce an ordering of coflows, and then perform rate allocations to optimize average CCT. The entire algorithm is pasted below:

$$\Gamma = \max\left(\max_i \frac{\sum_j d_{ij}}{Rem(P_i^{\text{in}})}, \max_j \frac{\sum_i d_{ij}}{Rem(P_j^{\text{out}})}\right) \qquad (1)$$ [2]

Pseudocode 1 Coflow Scheduling to Minimize CCT

```
1:  procedure ALLOCBANDWIDTH(Coflows $\mathbb{C}$, $Rem(.)$, Bool $cct$)
2:      for all $C \in \mathbb{C}$ do
3:          $\tau = \Gamma^C$ (Calculated using Equation (1))
4:          if not $cct$ then
5:              $\tau = D^C$
6:          end if
7:          for all $d_{ij} \in C$ do                          ▷ MADD
8:              $r_{ij} = d_{ij}/\tau$
9:              Update $Rem(P_i^{\text{in}})$ and $Rem(P_j^{\text{out}})$
10:         end for
11:     end for
12: end procedure

13: procedure MINCCTOFFLINE(Coflows $\mathbb{C}$, $C$, $Rem(.)$)
14:     $\mathbb{C}'$ = SORT_ASC ($\mathbb{C} \cup C$) using SEBF
15:     allocBandwidth($\mathbb{C}'$, $Rem(.)$, true)
16:     Distribute unused bandwidth to $C \in \mathbb{C}'$   ▷ Work conserv. (§5.3.4)
17:     return $\mathbb{C}'$
18: end procedure
```

Line 14 calculates an ordering of the coflows by sorting them in order of their Γ value calculated using equation (1). This can be parallelized greatly and the computation of Γ^C of each coflow could be parallelized using chunking similar to Marlow's parallel KMeans example. However, it's not entirely obvious what data structure to represent the input data (coflows, ports, etc) as to allow for easy parallel computation of Γ, requiring some thought. We also think that there's an opportunity to use REPA to speed up Γ^C calculations. The second part of the algorithm (Line 15) is iterative and would be hard to parallelize since each iteration of the loop depends on the state *Rem(.)* updated by the previous iteration.

In datacenter environments, it's common to have thousands of coflows arrive per second, with hundreds of thousands of servers. It's common for NIC to support 8 virtual output queues. Thus, we plan to generate problems with 1000s of coflows of different sizes, 100,000s ingress ports and 8 egress ports for Varys to solve. The hope is that the algorithm will run long enough to allow parallelizing it to observe some speed up.

[2] for a coflow, d_{ij} represents the size of data that goes from ingress port i to egress port j. *Rem(.)* represents the remaining bandwidth of an ingress port or egress port. P_i^{in} represents ingress port i, P_j^{out} represents egress port j.

References

[1]: https://dl.acm.org/doi/10.1145/3230543.3230569
[2]: https://dl.acm.org/doi/10.1145/2619239.2626315