2016 IEEE 24th International Conference on Network Protocols (ICNP)
Poster Paper

Macroflow: A Fine-grained Networking Abstraction for Job Completion Time Oriented Scheduling in Datacenters

Chen Tian† Junhua Yan† Alex X. Liu†‡ Yizhou Tang† Yuankun Zhong† Zi Li†
†*State Key Laboratory for Novel Software Technology, Nanjing University, China*
‡*Department of Computer Science and Engineering, Michigan State University, USA*

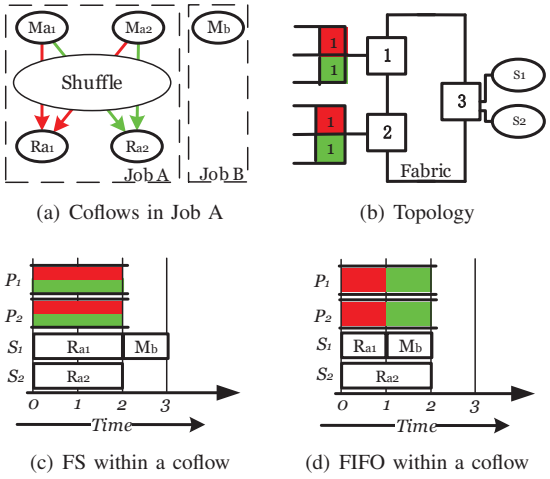
***Abstract*—For a datacenter running a data-parallel analytic framework, minimizing *job completion time* (JCT) is crucial for application performance. The key observation is that JCT could be improved, if network scheduling can exploit the opportunity of decreasing the amount of occupied machine slot-time spend on communication. We propose *Macroflow*, a networking abstraction that captures the primitive resource granularity of data-parallel frameworks. We study the inter-macroflow scheduling problem for decreasing application JCT. We propose the *Smallest-Macroflow-First* (SMF) and *Smallest-Average-Macroflow-First* (SAMF) heuristics that greedily schedule macroflows based on their network footprint. Trace-driven simulations demonstrate that our algorithms can reduce the average and tail JCT of network-intensive jobs by up to 20% and 25%, respectively; at the same time, the throughput of computation-intensive jobs is increased by up to 2.2×.**

1. Introduction

For a datacenter running a data parallel analytic framework such as Hadoop, minimizing *job completion time* (JCT) is crucial for application performance. A job's JCT is dominated by both computation and network durations. Depending on the footprint of communication, a job can be either network-intensive or computation-intensive. In Facebook, nearly 95% Hadoop jobs have no shuffle stages at all, and over 96% of such jobs are used for small-scale interactive and exploratory analyses: the average input data of each job is only 21 KB.

Network researchers currently optimize several network-level metrics to help applications. Many work focus on minimizing *flow completion time* (FCT) [**?**], [**?**]. Recently, the *coflow* abstraction bridges the gap between network and application metrics based on application semantics: Minimizing *coflow completion time* (CCT) might lead to reduced JCT [**?**]. However, there could be a large performance penalty when using state-of-the-art network metrics. Current network metrics such as CCT are semantically different from JCT, unless every job has and only has a single coflow phase and no computation phase at all.

Consider the example in Figure **??**: job A has two mappers M_{a1}/M_{a2}, and two reducers R_{a1}/R_{a2} with negligible computation requirements; totally there would be four flows in its coflow; there is an additional job B, which only has a single mapper task M_b with 1 slot-time computation waiting to be scheduled (Figure **??**). There exist ingress ports $P1/P2$ in this datacenter fabric, and each port can send/receive 1 unit of data in one time unit; egress port $P3$ can receive 2 units of data per time unit (Figure **??**); M_{a1}/M_{a2} are already finished, and each flow has one unit of shuffle data to be collected; two machine slots $S1/S2$ behind port 3 hold R_{a1}/R_{a2} running independently. Apparently, the average CCT is fixed (2 time units for job A to complete) regardless of flow scheduling. However, the average JCT can be quite different regards different flow scheduling policies. A widely-accepted mantra about coflow is to finish all flows simultaneously within a coflow. Suppose we use fair sharing among flows, then all four flows finished in time 2. In this case, M_b can be allocated only when job A is finished: the resulted average JCT is 2.5 time units (Figure **??**). As a comparison, if we let flows $M_{a1} \rightarrow R_{a1}$ and $M_{a2} \rightarrow R_{a1}$ have priority, then they can be finished in 1 time unit without impacting the coflow finish time (Figure **??**). In this case, R_{a1} has received all data and can be completed in time 1; then M_b can be scheduled to occupy the computing slot released by R_{a1}; as a result, the average JCT is only 2 time units.



(a) Coflows in Job A (b) Topology

(c) FS within a coflow (d) FIFO within a coflow

Figure 1. A Motivation Example.

The key observation is that JCT could be improved, if network scheduling can decrease the total occupied machine slot-time spend on network transfers by data receiver processes. For an individual reducer in a shuffle phase, its computation stage can start as soon as all input data are ready, which is independent of all other reducers. As shown

978-1-5090-3281-5/16/$31.00 ©2016 IEEE

by the motivation example, prioritize some reducers' flows over others, and finish them earlier, can complete existing tasks earlier; in turn, new tasks can be started earlier and the total JCT can be reduced.

In this paper, we propose *Macroflow*, a networking abstraction that can capture the primitive machine slot-time granularity of data-parallel frameworks. Each macroflow is a collection of flows between a single reducer and all mappers in the shuffle. Given this definition, a coflow is a collection of macroflows, each with a distinct reducer. For example, we can use reducer R_{a1}/R_{a2} to denote the two macroflows in Job A's coflow. The abstraction allows the network to take scheduling decisions on the collection to achieve an optimized goal.

2. Design

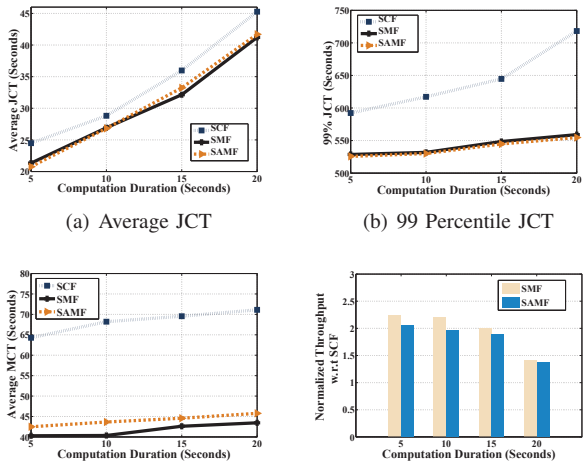
We study the inter-macroflow scheduling problem for decreasing application JCT. We prove the minimizing JCT problem to be strongly NP-hard and focus on developing effective heuristics. We demonstrate that the problem of *minimizing the total machine time of reducers* is equivalent to minimizing average *macroflow completion time* (MCT). We propose the *Smallest-Macroflow-First* (SMF) heuristic that greedily schedules a macroflow based on its network footprint. The SMF approach might interleave macroflows from different network-intensive jobs, since the correlations among macroflows in the same coflow are ignored. Correspondingly, we propose the *Smallest-Average-Macroflow-First* (SAMF) heuristic that greedily schedules all macroflows of a coflow together, based on the average footprint of all its contained macroflows.

We are currently in the process of developing a running system. The API framework is implemented in the application layer, by extending the existing coflow framework.

3. Evaluation

Methodology We evaluate our algorithms with a flow-level simulator by performing a replay of the collected Facebook logs. Since there are only flow information of network-intensive jobs in the logs, we choose to emulate each job's computation phase. For all jobs, the average computation duration is a parameter, and each computation job has a single mapper. A major decision is to compare the job throughput instead of job completion time: when and only when there is a free machine slot and there are no waiting reducers, such a computation-intensive job is inserted; the metric for performance comparison is the average throughput of computation jobs per second. Although an indirect metric, average job throughput is more accurate in capturing the saved machine slot-time than computation job JCT.

Figure **??** shows the experiment results. We make several observations. First, SMF/SAMF outperform SCF in both average/tail JCT. For average JCT, SMF/SAMF reduce JCT by up to 20% (when average computation duration is 5 seconds in Figure **??**). The largest improvement is for 99% tail latency: compared with SCF, the tail latency can be reduced by up to 25% (Figure **??**). The reason is that the



(a) Average JCT (b) 99 Percentile JCT

(c) Average MCT (d) Computation-intensive jobs

Figure 2. The Experiment Results.

larger the computation duration, the larger the impact of saved machine slot-time for algorithms.

Second, the average MCT is reduced by 35-37% for all scenarios (Figure **??**). As a result, SMF/SAMF can save a lot of machine slots. As expected, SAMF is slightly worse than SCF. A seemly counter-intuitive phenomenon is that: average MCT is even higher than average CCT. The reason is that a large number of small coflows dominate the average calculation of CCT; while the overwhelming number of macroflows in large coflows dominate the average calculation of MCT.

Third, the throughput of computation-intensive jobs are significantly higher with SMF/SAMF (Figure **??**). When computation duration is 5 seconds, SMF is 2.2× higher than SCF. Since SAMF is inferior in minimize MCT, the throughput of SAMF is lower, but still achieves 2× gain. With the increase of computation duration, the throughput gain of computation jobs decreases.

Acknowledgement

This work is partially supported by the National Science Foundation under Grant Numbers CNS-1318563, CNS-1524698, and CNS-1421407, and the National Natural Science Foundation of China under Grant Nos. 61472184, 61321491 and 61602194, and the Jiangsu High-level Innovation and Entrepreneurship (Shuangchuang) Program.

References

[1] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker. pfabric: Minimal near-optimal datacenter transport. In *ACM SIGCOMM*, volume 43, pages 435–446. ACM, 2013.

[2] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang. Information-agnostic flow scheduling for commodity data centers. In *NSDI*. USENIX, 2015.

[3] M. Chowdhury, Y. Zhong, and I. Stoica. Efficient coflow scheduling with varys. In *ACM SIGCOMM*, pages 443–454. ACM, 2014.