Correlation-Aware Flow Consolidation for Load Balancing and Beyond

Shiva Ketabi*, Matthew Buckley*, Parsa Pazhooheshy*, Faraz Farahvash†, Yashar Ganjali*
*University of Toronto, †Sharif University of Technology

ABSTRACT

Existing load balancing solutions rely on direct or indirect measurement of rates (or congestion) averaged over short periods of time. Sudden fluctuations in flow rates can lead to significant undershooting/overshooting of target link loads. In this paper, we make the case for taking variations and correlations of flows into account in load balancing. We propose *correlation-aware flow consolidation*, i.e. aggregating inversely correlated (or uncorrelated) flows into *superflows* and using them as building blocks for load balancing. Superflows are smoother than individual flows, and thus are easier to estimate with a higher confidence, and can reduce overshooting/undershooting of link capacities. We present heuristic methods combined with predictive models to consolidate flows and show they can lead to significant reductions in rate standard deviations compared to correlation-agnostic solutions (up to 33% and 12% improvements at the 50th and 99th percentiles respectively for 20 superflows based on real traffic traces).

1. INTRODUCTION

Modern network topologies offer several equal-cost paths between any pair of source/destination nodes [16, 1], accommodating a wide range of workloads from latency-sensitive short-lived flows (*e.g.*, web-search) to bandwidth-hungry flows (*e.g.*, backup traffic). To efficiently use these available paths and serve various workloads with different requirements, a critical component is a load balancing scheme that can dynamically assign traffic to the existing paths.

There are various load balancing schemes with varying levels of complexity and flexibility. ECMP is probably the simplest and most widely used scheme in practice that uses a simple hash to assign flows to outgoing paths of a given switch [19]. ECMP and other local schemes [18, 14] deal with hash collisions, packet disorders, and asymmetric topologies in the absence of global congestion state. Other schemes aim for global load balancing enforced in a centralized controller [2, 20, 34, 5], or in a distributed manner [3, 25].

A common challenge for these load balancing schemes is the volatile nature of flows. There is a constant barrage of new flows, and flow departures, and even during the life-time of a given flow its rate constantly changes. To deal with this volatility, today's load balancing solutions try to adapt themselves based on direct or indirect measurements of *average* utilizations (or congestion) along different paths. All these solutions, however, are oblivious to second degree variations in flow rates.

Correlation-Aware Flow Consolidation. In this paper, we introduce *"correlation-aware flow consolidation"* as a simple mechanism to enhance existing load balancing solutions. By aggregating a set of flows (or flowlets [37, 22, 38]) into a number of groups, *i.e. "superflows"*, we minimize the collective variations within each group. Aggregating N independent flows – based on the Central Limit Theorem [33] – reduces variations in the aggregate by $O(\sqrt{N})$. For inversely correlated flows, the aggregate might be even smoother.

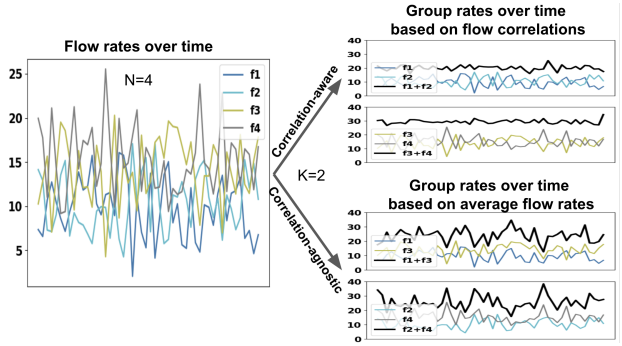


Figure 1: Flow consolidation example.

The output of this correlation-aware flow consolidation step (i.e. superflows), can be fed to existing load balancing schemes that are oblivious to flow variations over time. Using superflows here has two potential advantages: 1) reducing variations within each group leads to reducing peak requirements (*i.e.* eliminating overshooting of link/path capacities), and 2) reduced variations within each group helps in estimating future demands of these groups with higher confidence levels.

Example. Consider the simple example in Figure 1 with $N = 4$ flows consisting of two pairs of inversely correlated flows[1]. Here, f_1 and f_2 have an average rate of 10 and variance of 10, and $f3$ and $f4$ have an average rate of 15 and variance of 15. The consolidation on the top-right of the figure is correlation-aware in which we aggregate inversely correlated flows, *i.e.* flows f_1 and f_2 together and flows f_3 and f_4 together. We can see that variations in individual flow rates inside each group mostly cancel each other out, and we end up with relatively smooth aggregate rates. In the correlation-agnostic consolidation, shown in the bottom-right, we only consider the average rate of flows and group f_1 and f_3 together and f_2 and f_4 together. Here, both groups have an average rate of $10 + 15 = 25$ which is perfectly balanced on average. However, the high variance of flows causes much more overshooting/undershooting of the resources in the correlation-agnostic consolidation.

Challenges. Interdependency-aware scheduling of server workloads has been extensively studied in the literature [39, 28, 29, 27, 32]. In particular, Verma et al. [39] propose Correlation Based Placement (CBP), which assigns workloads to servers based on the pairwise correlation of different workloads. CBP tries to place workloads with lower pairwise correlation on the same server to avoid simultaneous workload peaks.

However, in the networking context, correlation-aware flow con-

Workshop on AI in Networks and Distributed Systems (WAIN) 2021 Milan,Italy

[1]For simplicity, these flows are synthetically generated from 4 Gaussian process distributions. In our evaluations, we use real network traces to show the point is still valid.

solidation is mostly overlooked due to the following challenges. First, measuring pairwise correlation of flows is more difficult compared to server workloads. The resource requirements of server workloads are assumed to be available upon arrival in most cases [28], but in networks, future flow demand rates are not given a priori. Second, the real-time nature of correlation-aware flow consolidation makes it very challenging and as a result, such a solution needs to be extremely fast.

Our Solution. To address the correlation-aware flow consolidation problem, our proposed solution has two main components: a rate prediction component and a correlation-aware heuristic to aggregate flows. The rate prediction component is responsible for providing estimates of the flow rates for the next few time slots, which are used to estimate flow correlations (that are fed to the aggregation algorithm). We note that our objective is not to accurately predict future rates; our ultimate goal is to get an estimation of future rate variations in order to measure flow correlations needed for aggregation. In our evaluations, we show this is a required step and we cannot simply rely on past correlations.

The second component is a statistical heuristic algorithm that calculates flow correlations based on the flow rate variations estimated by the prediction component, and iteratively groups flows with low correlation.While our goal is to keep the correlation in each group minimum, another requirement is to dynamically run the algorithm very fast. Therefore, to avoid quadratic run-time, instead of computing pair-wise correlation among the flows, we define a group representative, and calculate the correlation of each flow with each group representative and place flows in groups accordingly.

We use a correlation-aware solution for long-lived flows[2] to have the opportunity to measure flow correlations. Shorter flows are simply served in a round-robin (or randomized) manner which is very effective in terms of balancing the load. Since the majority of the bytes are usually carried by long-lived flows [23] (which is the case in the dataset used for evaluation with 90% of traffic carried by long-lived flows), the proposed solution seems to lead to considerable improvements overall (short-lived and long-lived flows).

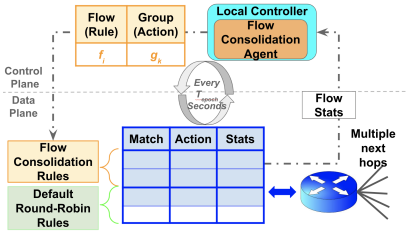
Results. Our experiments show that correlation-aware flow consolidation leads to significant reduction in the standard deviation of aggregate flows: up to 66% improvement on average in oracle-based scenarios and 33% improvement on average when using predictive models based on real traffic traces with more than 500 flows. We expect these reductions in rate variations would lead to improvement in any load balancing scheme that deals with these smooth superflows. Our experiments also show that the prediction component is essential for correlation-aware flow consolidation: *i.e.*, simply relying on past observations for estimating correlations gives little or no improvement in comparison with correlation-agnostic solutions on a consistent basis. However, adding predictions of future rate variations significantly improves the results.

We also evaluate the impact of ranging the number of groups between 2 and 30 and show that the performance of our solution improves as we increase the number of groups in this range (*e.g.* 14% improvement as the number of groups is increased from 10 to 20). This is an interesting observation as larger number of groups gives more flexibility to the load balancing scheme that will be fed by the output of the flow consolidation. Finally, our run-time evaluation of the proposed solution suggests extremely low overhead and run-time making the solution feasible in realistic scenarios.

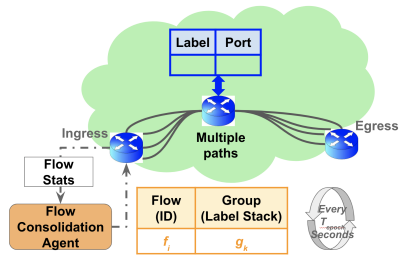
2. CONTEXT

The main premise of this paper is that correlation-aware flow consolidation leads to smoother aggregates that can enhance load balancing, especially when individual flows are volatile. In this section, we will set the stage and explain the context.

[2]Here, long-lived refers to flows that were active in at least two consecutive epochs, as defined in Section 5.



(a) Local setting.



(b) Global setting.

Figure 2: Local and global load balancing designs with correlation-aware flow consolidation.

Flow Correlations. Linear correlation among a pair of flows can be defined by Pearson correlation coefficient [4, 40]. A high positive correlation (closer to 1) represents two flows with similar behaviour (i.e. synced rates), a negative correlation (closer to -1) represents flows which are inversely synced, and correlation values close to zero show linearly independent flows.

Today's networks carry traffic with all types of correlations. For instance, distributed learning applications, *e.g.* federated learning [21], create a huge number of positively-correlated flows. Pipeline applications, *e.g.* MapReduce [10], lead to negatively correlated flows corresponding to data transfer of different processing elements. Furthermore, the network structure also causes positive/negative correlations in the traffic. As an example, flows passing through the same set of bottlenecks show inverse correlation as the sum of their rates is bounded by the bottleneck capacity.

Load Balancing System Design. Here we present our design proposal for using correlation-aware flow consolidation in both local and global load balancing schemes. Figure 2a shows a local scheme combined with correlation-aware flow consolidation for OpenFlow switches [31]. We install a flow consolidation agent (application) in a controller connected to each OpenFlow switch. This application receives flow statistics every T_{epoch} seconds from the switch and accordingly groups the flows. The output is a mapping from flows to groups (or superflows). The underlying load balancing scheme can decide the output port assigned to this superflow.

For global schemes, the goal is to balance the load over leaf-to-leaf switches or an overlay network illustrated in Figure 2b using segment routing protocols [11]. The flow consolidation agent can be installed close to the ingress (or source leaf) switch and periodically sends the flow groups to the switch. The ingress switch chooses a path for each superflow based on the grouping and the underlying load balancing scheme, and pushes an ordered list of labels to the packet header representing the end-to-end route of the packet which is used by the segment routing protocol in switches. We note that this paper focuses on evaluating the main idea to show its feasibility and potential benefits. Detailed systems level implementation and evaluation is left as a future work.

3. FLOW CONSOLIDATION PROBLEM

In this section, we formally define the flow consolidation problem. Let us consider a set of N flows $\mathcal{F} = \{f_1, \ldots, f_N\}$ that are going to be balanced over multiple paths (or links).We assume time is slotted and the demand rate of flow $f_i \in \mathcal{F}$ at time $j \in \{1, \ldots, T\}$ is represented by $\texttt{rate}_i(j)$. We also assume time is divided into fixed epochs of size T_{epoch} slots each. We focus on a given epoch t for the rest of this section ($t \in \{0, \ldots, \frac{T}{T_{epoch}} - 1\}$). For simplicity, we assume T is a multiple of T_{epoch}. Since we are focusing on the t-th epoch, let us represent the rate of flow f_i at the l-th ($l \in \{1, \ldots, T_{\text{epoch}}\}$) time slot of the epoch with $r_i(l) = \texttt{rate}_i(tT_{\text{epoch}} + l)$.

At the beginning of epoch t (i.e. at the beginning of time slot $t \times T_{\text{epoch}} + 1$), we consolidate flows into K groups or superflows. These superflows remain unchanged until the end of the epoch, and are fed to the underlying load balancing system. We assume all flow rates for the duration of the epoch are known at the start of the epoch. Later, we show how to relax this assumption in practice.

Our goal is to find superflows in a way that minimizes the maximum aggregate rate of superflows so as to reduce variations when these superflows are assigned to specific paths. More formally, we aim to partition the flows in $\mathcal{F}$ into K different groups denoted by $\mathcal{G} = \{g_1, \ldots, g_K\}$. We define a function
$\mathsf{Group} : \mathcal{F} \rightarrow \{1, \ldots, K\}$ such that $\mathsf{Group}(f_i) = k$ implies that $f_i \in g_k$ during the epoch. We are looking for the solution to the following mixed integer programming problem:

$$\begin{aligned} \min_{\mathsf{Group}, \{R_k(l)\}} \quad & \max_{k \in \{1,\ldots,K\}, l \in \{1,\ldots,T_{\text{epoch}}\}} R_k(l) \\ \text{subject to} \quad & \sum_{i \in g_k} r_i(l) \leq R_k(l), \\ & \forall k \in \{1, \ldots, K\}, \forall l \in \{1, \ldots, T_{\text{epoch}}\}. \end{aligned} \tag{1}$$

where $R_k(l)$ corresponds to a hard threshold on the aggregate rate of the flows assigned to group g_k (ie. the superflow rate). In other words, we are seeking a function that minimizes the maximum of the aggregate rates among all the possible grouping functions.

Computational Complexity. The flow consolidation problem, presented as a mixed integer program in (1), is closely related to a well-known NP-hard problem, namely multi-processor scheduling [13]. We prove the flow consolidation problem is NP-hard via a reduction from the multiprocessor scheduling problem. In the multiprocessor scheduling problem, we have a set of N jobs $\mathcal{J} = \{J_1, \ldots, J_N\}$ and a set of M processors $\mathcal{P} = \{\text{proc}_1, \ldots, \text{proc}_M\}$. Job J_i has a processing time $\text{p}_i \in \mathbb{N}_+$. The goal is to assign jobs to the processors so as to minimize the time required to finish all the jobs. Let us consider a specific version of our flow consolidation problem in which we are able to regroup the flows at every time slot (*i.e.*, $T_{\text{epoch}} = 1$). Let us also consider a given instance of the multiprocessor scheduling problem. We can map jobs to flows, job processing times to flow rates, and processors to groups. The aggregate rate of group k, i.e. $\sum_{i \in g_k} r_i(l)$ at time l is equivalent to the total processing time of the k-th processor. It is immediate to see that the solution to the flow consolidation problem is also a solution for the multiprocessor scheduling problem instance.

Practical Challenge. The silver-lining in the multiprocessor scheduling problem is that it has a $\frac{4}{3} - \frac{1}{3M}$ approximation algorithm known as *Longest Processing Time*, where M is the number of processors [15]. It sorts the jobs based on their descending processing times, starts with jobs with higher processing times, and assigns jobs to processors with the lowest end time so far. This solution requires full knowledge of job processing times a priori. Unfortunately, in networks, flow rates are not available a priori, so we should relax the assumption that we know flow rates beforehand.

4. CORRELATION-AWARE FLOW CONSOLIDATION

In this section, we present our solution for the flow consolidation problem. The main idea is to focus on the variance of the aggregated group rates instead of the maximum rates. By upper bounding this variance, we can provide high probability guarantees for the maximum rate of each group (using Chebyshev's inequality [33]). As also observed in the example of Figure 1, the aggregated group rates, shown in black on the top-right, have bounded variances over time which lead to a reduction in the maximum rate. Therefore, we convert the optimization problem in (1) into the following problem aiming to reduce the average of group rate variances:

$$\begin{aligned} & \min_{\mathsf{Group}} \frac{1}{K} \sum_{i=1}^{K} \mathbb{E}[\widehat{\text{var}}[\mathcal{S}_k]] \\ & \text{subject to } \mathcal{S}_k = \{\sum_{i \in g_k} r_i(1), \ldots, \sum_{i \in g_k} r_i(T_{\text{epoch}})\}. \end{aligned} \tag{2}$$

where $\mathcal{S}_k$ denotes the aggregated rate of Group k during the epoch, and $\widehat{\text{var}}[\mathcal{A}]$ represents the empirical variance of the elements in $\mathcal{A}$. In other words, the variance of each group is considered as a "surrogate" cost function for the maximum sum of the flow rates.

To control the variance of the aggregated group rates, we focus on flow correlations. We present a *correlation-aware* heuristic that estimates flow correlations based on the flow rates and accordingly predicts the optimized groups without accessing future flow rates. The intuition behind our correlation-aware solution is that the aggregation of independent (or inversely correlated) flows shows less variance over time and prevents flows peaking at the same time.

Correlation-Aware Heuristic. We present a statistical method for grouping flows called *Lowest Correlation Grouping (LCG)*. It sorts the flows into a fixed number of groups K, using the correlation between flows to decide on the best grouping. We also experimented with other heuristics using either correlation or variance. However, our experimental results showed LCG consistently performed best. To calculate the correlations, a subset of the flow rates in a fixed window are used. For the remainder of this section, the term correlation refers to correlation of flows measured in this window.

LCG directly sorts the flows into K groups by placing a flow into the group with which it has the lowest correlation. The algorithm starts by initializing K empty groups, $g_1, \ldots, g_K$. For each group g_k ($k \in \{1, \ldots K\}$), we maintain a total aggregate flow $\mathcal{S}_k$, which is the component-wise sum of the rates of all flows in the group. The first flow is assigned to g_1. Then at each iteration, we calculate the correlation between f, the next unassigned flow, and $\mathcal{S}_k$ for all non-empty groups and denote the minimum computed correlation as c^* and the corresponding group as g^*. If all the groups are non-empty, f is assigned to g^*. Also, if there are empty groups, but c^* is below a threshold (0.3 in our case), we still assign it to g^*. Otherwise, f is assigned to the next empty group. This procedure is repeated until all the flows are assigned to one of the K groups.

As discussed in Section 5, the algorithm was first run using the rates from the previous epoch to group flows in the current epoch. However, using this rule, we did not see satisfactory performance. Next, we assumed that we knew the rates in advance and grouped flows in the current epoch using rates from that epoch. With this rule, LCG shows significant improvements in performance. As it is unrealistic to assume we know the rates in advance, we relax this assumption by using predictors to estimate the rates for the current epoch based on past rates.

Prediction Component. To have a prediction about the future flow rates, we examined multiple machine learning models using the flow rate history. Our predictive models estimate the rates of a flow f in the current epoch, using the rates of f in the previous

epoch. Specifically, the models use the start time of f, the rates of f in each time slot of the previous epoch, as well as the maximum, mean, standard deviation, and skewness of the past rates [26]. We examined 4 classes of regression models: Random Forest Regression [6], XGBoost [8], Ridge Regression [30], and Stochastic Gradient Descent Regression (SGD) [12].

The first 30% of the epochs are used for training, the next 20% for validation, and the final 50% for testing. We use only the first 30% of the epochs for training to show that even a small amount of historical data can help us develop prediction models with an acceptable performance. The validation set is used to tune hyper-parameters for the models. For Ridge Regression, the regularization value is the only hyper-parameter. The regularization parameters and the learning rate are tuned for SGD. Finally, the tuned hyper-parameters for Random Forest Regression, are the maximum depth of the tree and number of trees, and for XGBoost regression, they are the learning rate, number of additive estimators, and maximum depth of the estimators. After finding the best hyper-parameters for each model, the models are trained on train and validation data again and then kept fixed during the test period. However, as SGD is an online algorithm, it continues to learn during the testing period.

We measure the quality of the predictions in the validation process using R-squared (R^2), a common metric in regression analysis as our prediction performance metric [9, 35]. As flow rates might be zero during many time slots, we use R^2 instead of relative error metrics such as root mean square relative error (rMSRE) [12].

5. EVALUATION

In this section, we evaluate our correlation-aware consolidation scheme LCG with a random grouping algorithm, and an oracle-based correlation-agnostic load balancing solution called *Highest Rate First (HRF)*. We use the standard deviation of group rates as our performance metric.

Traffic. We use real data traces collected at the border of an ISP [17]. We identify flows by the classic 5-tuples, and extract flow-level information every second from half an hour of the data.[3] We randomly select $N = 500$ flows that have non-zero rates for at least 70% of their sampled rates. We can easily expand our solution to a data center environment and evaluate it by reducing the time scales. In the interest of space, we keep this evaluation for future work.

In our experiments, we simulate a scenario in which shared network links cause correlation among flows passing through them. To this end, we randomly pass each of the 500 flows from one of 30 different paths simulated in ns3 [36]. We capture flow rates after passing through these shared paths, and use these captured flow rates to dynamically group flows in our correlation-aware solution.[4]

Comparables. We compare the LCG heuristic with a correlation-agnostic solution HRF, and a random grouping heuristic. Under the random grouping heuristic, we initialize K empty groups $g_1, ..., g_K$. Then, for each flow f, we choose a random number $k \in \{1, ..., K\}$ and assign flow f to group g_k. The *Highest Rate First (HRF)* scheme is derived from the *Longest Processing Time (LPT)* rule for the multiprocessor scheduling problem mentioned in Section 3. Similar to LPT, we assume the rates for the current epoch are all known in advance and HRF assigns the flows to groups in decreasing order of average flow rates. Knowing rates in advance is not realistic, but we are using this as a baseline to compare a solution which only focuses on average rates (even with perfect future knowledge) and ignores the rate variances with our correlation-aware solution LCG. Our goal is to show that LCG without having knowledge of future rates outperforms HRF with such knowledge.

[3]We have selected one second as our time slot unit since collecting information at sub-second granularity leads to many entries with zero packets in our dataset. Assuming the input dataset has more fine-grained granularity, there is no fundamental barrier going to lower timescales except for our run-time which can be very short.

[4]Please note that passing flows through shared links is a cause of correlation among flows. In principle, our solution is capable of identifying these correlations regardless of their source/cause.

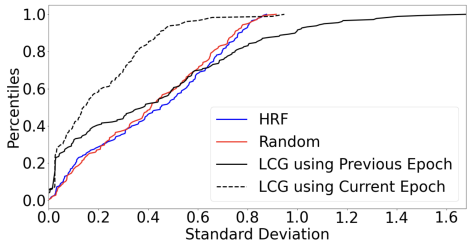


Figure 3: CDF of mean standard deviation without using predictive models.

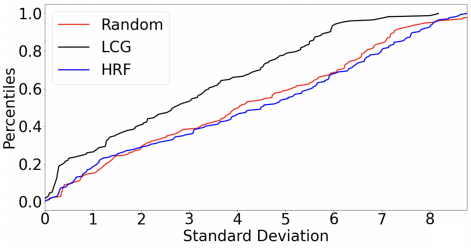


Figure 4: CDF of mean standard deviation with Ridge Regression.

For all experiments, $T_{\text{epoch}} = 5$ so that flows are regrouped every 5 time slots. There are a total of 1781 time slots, representing 356 epochs. When running our correlation-aware heuristic, we consider only flows that were active in at least 1 time slot in the previous epoch, and denote these as long-lived flows. Thus, short-lived flows correspond to flows that are active only in the current epoch. As IP addresses are anonymized and we have no previous rates for these flows, the best we can do for short-lived flows is to randomly assign them to groups. Thus, in all experiments, we run our correlation-aware heuristic on long-lived flows and randomly assign short-lived flows to groups.

Experiment Results. In our first experiments, we group flows in the absence of the prediction component explained in section 4. Figure 3 shows the results when sorting flows into 20 groups. Similar results hold for the other group sizes. The figure depicts the CDF of the mean group standard deviation for each epoch.

We have two variations of LCG here. In the first variation, LCG groups flows using their rates in the previous epoch. That is, starting from the second epoch, when considering the groups for the t-th epoch, LCG uses the rates from epoch $t - 1$. In this setting, LCG outperforms random more than 90% of time, providing a 26.23% reduction in average standard deviation compared to random at the 50^{th} percentile. However, it does not perform well at the tail of the distribution.These values get worse with more groups.This shows that by simply using rates from the previous epoch, we cannot hope to perform better than random in all scenarios.

In the second variation, we assume LCG knows the flow rates of the current epoch in advance. Here, LCG greatly outperforms the random algorithm. It leads to a reduction in average standard deviation when compared to random of 66.64% at the 50^{th} percentile and an increase of only 0.90% at the 99^{th} percentile. The results for the maximum group standard deviation show a similar story. Here, LCG with current epoch information far outperforms HRF. We note that HRF is assumed to be oracle-based and has access to all rate information in advance.

Using Predictive Models. The previous experiments clearly demonstrate that LCG has the ability to considerably smooth flows compared to the random algorithm. However, to compensate for the

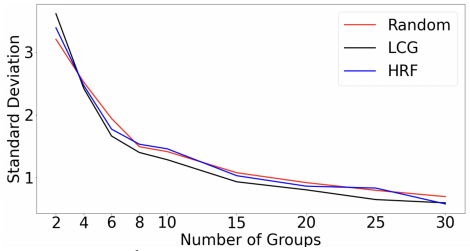


Figure 5: 99th percentile for standard deviation.

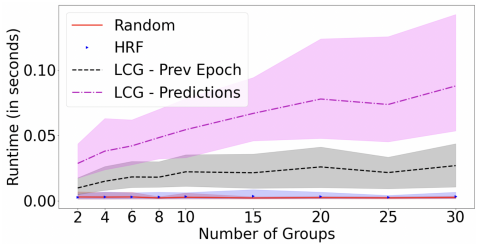


Figure 6: Run-time.

unrealistic assumption that LCG knows the rates in advance, we instead use predictive models to forecast the rates for the next epoch using rates from the previous epoch. Of all the predictive models introduced in Section 4, Ridge Regression has the best performance in terms of decreasing the standard deviation of groupings. Based on the coefficients of the final ridge regression model, the flow rate in the last slot and the average of the flow rate are the two most important features in predicting the future rates of the flow. Figure 4 shows the results of using LCG with Ridge Regression as a predictor. The figure again shows CDFs of the mean group standard deviation across epochs for 20 groups and similar results hold for the other group numbers examined. LCG outperforms both random and HRF here, leading to a reduction in average standard deviation of 33.36% and 12.58% at the 50th and 99th percentiles, respectively.

We also performed a set of experiments to see the impact of number of groups on our solution. Figure 5 shows 99th percentile values of mean standard deviation across different number of groups for LCG with predictive models, random, and HRF. Each point represents the percentile of mean standard deviation across all epochs for the particular algorithm versus the number of groups. These graphs clearly show that the LCG algorithm outperforms random, and this improvement becomes more apparent with a larger number of groups. LCG with a predictive model also outperforms the oracle-based HRF in most cases here as well.

Run-time Analysis. Let n be the number of flows in a given epoch t. We consider the run-times for one epoch. The random algorithm simply selects a group randomly for each flow and so its running time is $O(n)$. The LCG algorithm places flows into K groups by calculating a group dependent measure for each group. This requires $O(K)$ work for each flow and $O(Kn)$ work in total.

Figure 6 shows the running times when executing the introduced heuristics. The plot shows the average run-time across all epochs for each heuristic with various number of groups. The error bands represent the 5th and 95th percentile. These experiments were run on a personal laptop with an Apple M1 chip and 16GB of RAM. Moreover, the predictions were run on the same machine that was running the simulator. In a production setting, the prediction would likely take place in parallel on a separate machine and the remaining calculations would be performed in a larger compute cluster. Therefore, we expect the running time of the proposed solution be acceptable for large number of flows and groups.

Communication Overhead. Given n flows, T_{epoch} time slots per epoch, and d bits for representing a flow rate, the overhead for communicating the flow statistics is $n \times T_{epoch} \times d$ per epoch which is negligible and bounded by $500 \times 5 \times 32 = 80000$ bits/epoch in our experiments.

6. DISCUSSION

Comparison with state of the art load balancing. To the best of our knowledge, this is the first solution that proactively avoids undershooting/overshooting of link utilization in load balancing by taking into account flow variances and predicting flow correlations. Many load balancing schemes rely on random assignment of packets [19, 7] or flowlets [38, 18] to possible paths which are congestion-oblivious or suffer from topology asymmetry. Other schemes that take the network congestion state into account, either locally [14], or globally [3, 25, 24, 41], react based on how close average aggregated rates are to link capacities. These schemes are oblivious to rate variations and correlations. Our correlation-aware flow consolidation solution, however, can provide the load balancing schemes with smoother aggregate superflows which proactively eliminates congestion even in time slots less than control loop intervals. We note that many existing solutions balance at the granularity of flowlets, pinning an entire flowlet to one link to avoid packet reordering [3, 25]. This bounds the granularity of these solutions as the decision must be made based on the first packet of a flowlet. In contrast, our solution groups flows based on rates which can be sampled at any granularity desired. In addition, our solution can be used in conjunction with existing methods. Flows can still be balanced at the granularity of flowlets, so that different flowlets of the same flow can be assigned to different paths. Yet, our solution dictates when flowlets from different flows should be grouped together.

There are two main costs to this solution: 1) the overhead of collecting flow statistics and predicting correlations, and 2) the impact of reducing the granularity of load balancing as we need to deal with superflows rather than individual flows. Our preliminary results show that we can gain significant reductions in load variations of superflows, with reasonable overhead. The results are even better for larger number of superflows (up to an extent) which will ensure enough granularity for the load balancing step. Here, our goal is to show the potential benefits of the idea, and leave more extensive evaluations, and more detailed system analysis to future work.

Other Applications. In addition to load balancing, correlation-aware flow consolidation can improve other settings such as buffer sizing, power optimization, and even enhancing congestion control. In each of these scenarios we are dealing with a resource allocation problem (buffers and bandwidth). Using correlation-aware flow consolidation can help provide better estimates of needs, making it easier to allocate resources. There are many interesting problems here, and we leave them as future work.

7. CONCLUSION AND FUTURE WORK

A major challenge in traffic load balancing is the volatile nature of flows. To alleviate this problem, we propose correlation-aware flow consolidation in this paper. The goal is to reduce the fluctuations in aggregate flows by consolidating negatively correlated or independent flows. Our solution consists of a prediction component that generates flow rate estimates, and heuristics that use the estimates to compute flow correlations and group low-correlated flows. Our experiments show we can get significant reduction in the aggregated group standard deviation, and that using predictive models of flow variations and correlations are essential here.

This is a preliminary work that shows potential benefits of taking flow variations into account, and its feasibility under reasonable assumptions. We leave more detailed design and evaluations of various load balancing schemes using correlation-aware flow consolidation in different environments as well as other applications (e.g. buffer sizing and power optimization) as future work.

8. REFERENCES

[1] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. *ACM SIGCOMM computer communication review*, 38(4):63–74, 2008.

[2] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, A. Vahdat, et al. Hedera: dynamic flow scheduling for data center networks. In *Nsdi*, volume 10, pages 89–92. San Jose, USA, 2010.

[3] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, et al. Conga: Distributed congestion-aware load balancing for datacenters. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 503–514, 2014.

[4] J. Benesty, J. Chen, Y. Huang, and I. Cohen. Pearson correlation coefficient. In *Noise reduction in speech processing*, pages 1–4. Springer, 2009.

[5] T. Benson, A. Anand, A. Akella, and M. Zhang. Microte: Fine grained traffic engineering for data centers. In *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies*, pages 1–12, 2011.

[6] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[7] J. Cao, R. Xia, P. Yang, C. Guo, G. Lu, L. Yuan, Y. Zheng, H. Wu, Y. Xiong, and D. Maltz. Per-packet load-balanced, low-latency routing for clos-based data center networks. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, pages 49–60, 2013.

[8] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[9] D. Dancer and A. Tremayne. R-squared and prediction in regression with ordered quantitative response. *Journal of Applied Statistics*, 32(5):483–493, 2005.

[10] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[11] C. Filsfils, N. K. Nainar, C. Pignataro, J. C. Cardona, and P. Francois. The segment routing architecture. In *2015 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2015.

[12] J. Franklin. The elements of statistical learning: data mining, inference and prediction. *The Mathematical Intelligencer*, 27(2):83–85, 2005.

[13] M. R. Garey and D. S. Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

[14] S. Ghorbani, Z. Yang, P. B. Godfrey, Y. Ganjali, and A. Firoozshahian. Drill: Micro load balancing for low-latency data center networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 225–238, 2017.

[15] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM journal on Applied Mathematics*, 17(2):416–429, 1969.

[16] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. Vl2: a scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, pages 51–62, 2009.

[17] W. N. R. Group. WAND, ISPDSL II dataset. `https://wand.net.nz/wits/ispdsl/2/`.

[18] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella. Presto: Edge-based load balancing for fast datacenter networks. *ACM SIGCOMM Computer Communication Review*, 45(4):465–478, 2015.

[19] C. Hopps et al. Analysis of an equal-cost multi-path algorithm. Technical report, RFC 2992, November, 2000.

[20] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat. B4: experience with a globally-deployed software defined wan. *ACM SIGCOMM Computer Communication Review*, 43(4):3–14, Aug. 2013.

[21] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

[22] S. Kandula, D. Katabi, S. Sinha, and A. Berger. Dynamic load balancing without packet reordering. *ACM SIGCOMM Computer Communication Review*, 37(2):51–62, 2007.

[23] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken. The nature of data center traffic: measurements & analysis. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*, pages 202–208, 2009.

[24] N. Katta, A. Ghag, M. Hira, I. Keslassy, A. Bergman, C. Kim, and J. Rexford. Clove: Congestion-aware load balancing at the virtual edge. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, pages 323–335, 2017.

[25] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford. Hula: Scalable load balancing using programmable data planes. In *Proceedings of the Symposium on SDN Research*, pages 1–12, 2016.

[26] J. F. Kenney. *Mathematics of statistics*. D. Van Nostrand, 1939.

[27] P. Mahadevan, P. Sharma, S. Banerjee, and P. Ranganathan. Energy aware network operations. In *IEEE INFOCOM Workshops 2009*, pages 1–6. IEEE, 2009.

[28] H. Mao, M. Alizadeh, I. Menache, and S. Kandula. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pages 50–56, 2016.

[29] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM Special Interest Group on Data Communication*, pages 270–288. 2019.

[30] D. W. Marquardt and R. D. Snee. Ridge regression in practice. *The American Statistician*, 29(1):3–20, 1975.

[31] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[32] X. Meng, V. Pappas, and L. Zhang. Improving the scalability of data center networks with traffic-aware virtual machine placement. In *2010 Proceedings IEEE INFOCOM*, pages 1–9. IEEE, 2010.

[33] D. C. Montgomery and G. C. Runger. *Applied statistics and probability for engineers*. John Wiley & Sons, 2010.

[34] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal. Fastpass: A centralized" zero-queue" datacenter network. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 307–318, 2014.

[35] A. C. Rencher and F. C. Pun. Inflation of r2 in best subset regression. *Technometrics*, 22(1):49–53, 1980.

[36] G. F. Riley and T. R. Henderson. The ns-3 network simulator. In *Modeling and tools for network simulation*, pages 15–34. Springer, 2010.

[37] S. Sinha, S. Kandula, and D. Katabi. Harnessing tcp's burstiness with flowlet switching. In *Proc. 3rd ACM Workshop on Hot Topics in Networks (Hotnets-III)*. Citeseer, 2004.

[38] E. Vanini, R. Pan, M. Alizadeh, P. Taheri, and T. Edsall. Let it flow: Resilient asymmetric load balancing with flowlet switching. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, pages 407–420, 2017.

[39] A. Verma, G. Dasgupta, T. K. Nayak, P. De, and R. Kothari. Server workload analysis for power minimization using consolidation. In *Proceedings of the 2009 conference on USENIX Annual technical conference*, pages 28–28. USENIX Association, 2009.

[40] X. Wang, Y. Yao, X. Wang, K. Lu, and Q. Cao. Carpo: Correlation-aware power optimization in data center networks. In *2012 Proceedings IEEE INFOCOM*, pages 1125–1133. IEEE, 2012.

[41] H. Zhang, J. Zhang, W. Bai, K. Chen, and M. Chowdhury. Resilient datacenter load balancing in the wild. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 253–266, 2017.