2017 IEEE 37th International Conference on Distributed Computing Systems

An Event-Level Abstraction for Achieving Efficiency and Fairness in Network Update

Ting Qu*, Deke Guo*, Xiaomin Zhu*, Jie Wu†, Xiaolei Zhou* and Zhong Liu*
*National University of Defence Technology
†Temple University

***Abstract*—Changes of network state are a common source of instability in networks. An update event typically involves multiple flows that compete for network resources at the cost of rescheduling and migrating some existing flows. Previous network updating schemes tackle such flows independently, rather than as the entity of an update event. They only optimize the flow-level metrics for the flows involved in an update event. In this paper, we present an event-level abstraction of network update which groups flows of an update event and schedules them together to minimize the event completion time (ECT). We then study the scheduling problem of multiple update events for achieving high scheduling efficiency and preserving fairness. The designed least migration traffic first (LMTF) method schedules all update events in the FIFO order, but avoids head-of-line blocking by randomly fine-tuning the queue order of some events. It can considerably reduce the update cost, the average, and tail ECTs of all update events. In addition, we design a general parallel-LMTF (P-LMTF) method to guarantee fairness and further improve scheduling efficiency among update events. It improves the LMTF method by opportunistically updating multiple events simultaneously. The comprehensive evaluation results indicate that the average ECT of our approach is up to $10\times$ faster than the flow-level scheduling method for network update events, and its tail ECT is up to $6\times$ faster. Our P-LMTF method incurs 75% reduction in the average ECT compared with FIFO when the network utilization exceeds 70%, and it achieves a 42% reduction in tail ECT.**

I. Introduction

Due to updating issues triggered by operators, applications, and network devices etc., network condition consistently undergoes changes. Issues include the upgrades of switches, network failures and VM migrations [1]. When upgrading a switch, all flows initially passing through it should be rerouted along other parts of the network to ensure the normal execution of network applications. For the VM migration, a set of new flows would be generated for migrating involved VMs to other servers in the network. There are two general consequences of such update issues: change of network topology and change of traffic matrix. The aforementioned updating issues are common sources of instability in networks. Therefore, each network update should be planned well in advance and should be tackled by designing effective and efficient update schemes.

For a network update event under the initial network configuration, an update plan would usually derive a desired final network state in advance, including the final network topology and traffic distribution. The update process, however, usually includes many intermediate network states and may exhibit serious traffic congestion and other issues. For this reason, prior updating schemes focus on realizing the correct transition from an initial network state to a final network state and are roughly divided into two categories. One is the consistent update [2], which means that a packet/flow traverses the network obeying either old or new network configuration. The two-phase update method and its variants [3], [4], [5] fall into such a category. The other is the congestion-free update, which makes an update plan for any update event in advance. This plan contains a series of intermediate states, but the transition across adjacent states is lossless. SWAN [6], zUpdate [1] and their variants [7], [8] fall into this category.

An update event typically involves a collection of new or existing flows, and it cannot finish until such flows have been completed. However, existing updating schemes treat each flow in such a collection in isolation rather than organizing involved flows of an update event as an entity. Those updating schemes to optimize flow-level metrics do not perform well in optimizing event-level metrics, including event completion time (ECT) of an update event, the average and tail ECTs of multiple update events. Actually, the abstraction of per-flow update cannot capture the event-level or inter-event requirements in a collection of update events. Prior updating schemes fail to provide a frame to represent those event-level update semantics. For example, some flows of an update event may be blocked because they lack sufficient network resources, which are occupied by heavy flows of other update events. This would lead to high average and tail ECTs.

In this paper, we define an event-level abstraction of network update to group a collection of flows for an update event and schedule them together. If the network cannot serve a flow in the collection, a few existing flows are locally migrated to other appropriate paths if they exist to satisfy the bandwidth requirement of the flow in the collection [8]. We define the update cost of an update event as the amount of migrated traffic of existing flows for all of its flows. However, it is NP-complete to identify which existing flows should be migrated to satisfy the requirement of any flow in an update event and guarantee a minimal update cost. Accordingly, we propose an efficient method to calculate the set of migrated flows to approximate the optimal solution. The event-level abstraction can decrease the ECT by cooperatively allocating network resource for an update event. This considerably speed network update process and is very important for network management [9].

However, operators, applications, and network devices create multiple update events in a shared network. Such update

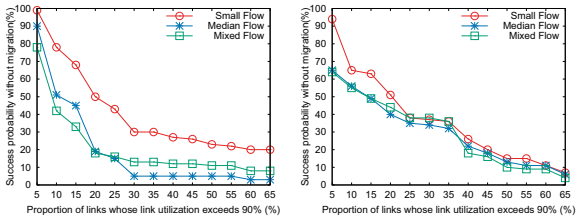
1063-6927/17 $31.00 © 2017 IEEE
DOI 10.1109/ICDCS.2017.124

events exhibit wide variations in the number of flows, the size of individual flows, and the total size. Simple scheduling mechanisms like FIFO [10] remains inapplicable to this inter-event scheduling problem. It usually incurs the serious head-of-line blocking problem when we use scheduling method like FIFO. That is, the head-event may be heavy and occupies more network resources for a longer period of time. Thus, many smaller update events that arrive later would be slowed down a long wait time in the update queue. This would increase the average ECT and tail ECT of a set of update events.

In this paper, we investigate the scheduling problem of multiple update events and focus on two different objectives: 1) speeding up the network update process by decreasing the average and tail ECTs and 2) preserving the update fairness. We propose the least migration traffic first (LMTF) method which schedules update events based on their arrival order, but dynamically fine-tunes the execution order when heavy update events are encountered. An intrinsic method is to dynamically compute the costs for all update events in the queue and execute the update event with the lowest cost first. This would cause non-trivial computation and time overhead. In contrast, LMTF compares the head-event with a few update events randomly selected from the queue and executes the smallest one first. This policy ensures that not all smaller update events are blocked behind heavy update events as they also have a chance to be executed earlier in each round. Additionally, LMTF considerably simplifies and accelerates the decision-making process.

Although LMTF can effectively decrease the average and tail ECTs, it impacts the FIFO fairness of queued events. To improve fairness for a queue of update events, we design a more general method called P-LMTF, which introduces the policy of opportunistic updating on the basis of our LMTF method. After inferring the new head-event from LMTF, it tries to find several potential events in arrival order that can be executed at the same time as the head-event. A heavy update event, which arrives earlier but is delayed by LMTF, still has priority to be executed in a timely manner during the process of opportunistic updating. This opportunistic updating policy further increases scheduling efficiency. In summary, as a general inter-event scheduling method, P-LMTF relaxes fairness slightly to achieve high efficiency, effectively decreasing the average and tail ECTs. The extensive trace-driven evaluations indicate that the average ECT of our approaches is up to 10× faster than the flow-level scheduling method for network update events, and the tail ECT is up to 6× faster. Our P-LMTF incurs 75% reduction in the average ECT against FIFO when the network utilization exceeds 70%, and it achieves a 42% reduction in tail ECT.

The remainder of this paper is organized as follows. Section II gives an overview of network update. In Section III, we present an event-level abstraction of network update and discuss inter-event scheduling models. In Section IV, we design inter-event scheduling methods for achieving both fairness and efficiency. We report the evaluation methodology and results in Section V. We discuss related work and conclude this paper in Sections VI and VII, respectively.



(a) Yahoo! Trace (b) Random Trace

Fig. 1. The success probability of accommodating a flow for an update event, without migrating other flows.

II. Network Update Overview

An update event typically involves a collection of new flows or existing flows. All flows in this collection need to compete for network resources along desired routing paths. However, the desired path for such a flow may not offer sufficient residual bandwidth for it. In this scenario, the desired path will exhibit congestion once the related flow is inserted into the network, especially when the link utilization is very high. Fig. 1 plots the success probability of inserting a flow of an update event into a Fat-Tree data center when k=8. The success probability decreases along with the increase of link utilization under the trace of Yahoo! data center [11] and the random trace with the distribution of traffic [12], irrespective of the flow size.

For this reason, it is necessary to first check if all links along the desired path offer sufficient link bandwidth when tackling each flow of an update event. If they do not, then the update event needs to be carefully addressed. An intrinsic method is to assign priorities to all flows in the network [13]. Existing flows with lower priorities will be removed if they block the new flows of higher priorities. This policy incurs a large volume of burst traffic, due to the retransmission of all removed flows. What is worse, determining which flows should be removed in an update event is an NP-hard problem, as proved in [8].

Another method is to reroute all existing flows to supply the flows of the update event with sufficient network resources. This policy aims to achieve a better network performance, in terms of load balance and link utilization, when the network topology or traffic changes. It is time-consuming to solve a series of linear programming (LP) problems. Moreover, globally rerouting all existing flows will lead to serious network-scale traffic migration.

Despite such considerations, the network update problem still lacks efficient solutions. In this paper, we present a novel strategy to locally adjust a few existing flows at congested links on the desired path of each flow. For such a flow, we look to see if a feasible path exists whose residual network resource is sufficient to meet the flow's requirement. If not, we locally migrate a few existing flows on congested links, so as to accommodate the new flow. Migrating more traffic will certainly take more time and negatively impact applications. We prefer to find a local re-routing solution to minimize the migration of existing flows, while releasing sufficient network resource to support a flow of the update event.

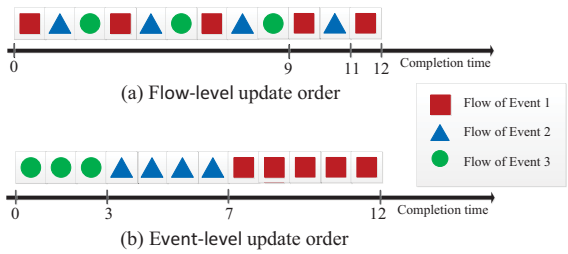


Fig. 2. The update orders of flows under flow-level and event-level methods.

The above approaches optimize flow-level metrics such as success probability and update speed. However, they are incapable of optimizing event-level metrics like average ECT and tail ECT, which are the primary goals for network update in many networks. Consider a series of flows caused by three update events. We may schedule such flows independently, as shown in Fig. 2(a). Alternatively, we may regard flows of an update event as a collection and schedule them in a certain order, as shown in Fig. 2(b). The average ECT of the three events is $(3+7+12)/3=22/3$ under the event-level scheduling manner, which is lower than $(9+11+12)/3=32/3$ under the flow-level scheduling manner. The tail ECTs of the three update events under the two scheduling manners are the same since we assume the durations of all flows are the same. In real networks, the durations of flows usually vary. So, the tail ECT of these three update events changes as well.

III. Event-level Abstraction of Network Update

In this section, we start with the event-level abstraction of network update and discuss the cost optimization of an update event. Accordingly, we characterize the inter-event scheduling problem of multiple update events.

A. Abstraction of event-level network update

The network is defined as a graph $G=(V,E)$, where V and E denote a set of switches and a set of links connecting those switches, respectively. Let $c_{i,j}$ be the residual bandwidth of link $e_{i,j}\in E$, while D denotes the network diameter. In addition, F refers to all flows in the network. For any flow $f\in F$, its bandwidth requirement is defined as d^f. Flow f is routed along a selected path p from set $P(f)$, denoting all feasible paths for that flow. For each link $e_{i,j}$ in the selected path p, $d^{f_{i,j}}$ denotes the consumed bandwidth by flow f on link $e_{i,j}$. The network is congestion-free if the following constraints are satisfied:

- $\forall f,\ \forall e_{i,j}\in p,\ d^f = d^{f_{i,j}}$,
- $\forall f,\ \forall e_{i,j}\notin p,\ d^{f_{i,j}} = 0$,
- $\forall e_{i,j}\in E,\ c_{i,j}\geq 0$.

The aforementioned constraints ensure that each flow $f\in F$ is unsplit and is forwarded along a certain path p. The last condition indicates that each link in the network is congestion-free after the network accommodates the entire flow set F.

Definition 1 (Migration of existing flows for a new flow): Consider a new flow f_a that is inserted into the network containing an existing flow set F. When flow f_a traverses a path $p\in P(f_a)$, it may cause congestion on some links. We define the set of congested links caused by flow f_a as $E^c_{f_a}$, i.e.: $\forall e_{i,j}\in p,$ if $d^{f_{i,j}} > c_{i,j},$ then $e_{i,j}\in E^c_{f_a}$.

We then define a set of all existing flows, each of which passes at least one congested link in $E^c_{f_a}$, as F_A. That is: $\forall f\in F, \exists e_{i,j}\in E^c_{f_a},$ if $d^{f_{i,j}}\neq 0,$ then $f\in F_A$.

So far, we try to find a subset F_a of F_A for a new flow f_a with the bandwidth requirement d^{f_a}. Thus, flow f_a could be accommodated by the network, if all flows in the set F_a are migrated to other parts of the network. That is,

$$\exists F_a\subset F_A, \forall e_{i,j}\in E^c_{f_a}, \sum_{f\in F_a} d^{f_{i,j}} + c_{i,j} \geq d^{f_a}. \tag{1}$$

Definition 2 (Event-level abstraction of network update): We abstract an update event U as a set of involved flows, denoted as $U=\{f_1, f_2, ..., f_w\}$. Any flow $f_a\in U$ would cause the migration of existing flows in set F_a, as shown in Definition 1. The sum of traffic migration of all flows in these sets is defined as the cost of an update event U. That is,

$$Cost(U) = \sum_{a=1}^{w} sum(F_a),$$

where $sum(F_a)$ denotes the sum of traffic migration of all flows in the set F_a for each flow $f_a\in U$. The reason is also explained in our previous work [14].

B. Cost optimization of an update event

Consider the negative impact of traffic migration on the network application. In this paper, we try to minimize the amount of migrated traffic caused by an update event U containing w flows. To tackle this problem, we need to find the minimum subset F_a in set F_A for each flow $f_a\in U$. At the same time, flow f_a can be served by the network if all flows in set F_a are migrated to other parts of the network. Thus, the sum of residual link bandwidth and the amount of migrated traffic on each congested link in $E^c_{f_a}$ should exceed the requirement of flow f_a, as shown in Formula (1).

The cost optimization problem of any update event U containing w flows can be formed as

$$\min \quad \sum_{a=1}^{w} \mathrm{sum}(F_a) \tag{2}$$

$$s.t.\ \ \forall e_{i,j}\in E^c_{f_a}, \sum_{f\in F_a} d^{f_{i,j}} + c_{i,j} \geq d^{f_a} \tag{3}$$

$$F_a \subset F_A \text{ for each flow } f_a, 1\leq a\leq w. \tag{4}$$

In addition, the migration of any flow in set F_a along another path p will not lead to congestion on other links, i.e.,

$$\forall f\in F_a,\ \forall e_{i,j}\in p,\ d^{f_{i,j}} \leq c_{i,j} \tag{5}$$

It is hard to calculate such a set F_a for any flow $f_a\in U$. Thus, we design a novel strategy to reduce the migration of existing flows, and any flow $f_a\in U$ will get enough bandwidth after migrating these existing flows.

C. Inter-event scheduling models among update events

A shared network usually needs to tackle a series update events that form a queue according to their arrival order. Simple scheduling mechanisms like FIFO [10] do not perform well in this environment. The head-event may be heavy and have a long execution time. It would block many smaller events that arrive later and increase the average and tail ECTs. In this paper, we study inter-event scheduling among multiple update events. Design goals focus on decreasing the average and tail ECTs and preserving update efficiency and fairness. To this end, we design two scheduling models.

Fine-tuning the order of update events. We utilize the cost of an update event in Definition 2 as the metric to schedule all update events in a queue. A simple way is to reorder all queued events based on their update costs and choose the smallest event to execute firstly. Note that the update queue is in flux due to the changed network traffic. Consequently, we have to reorder all queued events frequently. This causes non-trivial computation and time overhead, especially for large-scale networks and events. Moreover, the entire reordering fully breaks the order of the queue and destroys fairness among update events.

In this paper, we prefer to schedule update events based on their arrival order, but dynamically fine-tune the execution sequence to tackle the head-of-line blocking problem. Consider a set of n update events $U_1, U_2, ..., U_n$. We randomly choose two update events U_b and U_c, which contain v and w flows, respectively. Let F_b and F_c denote the set of existing flows, which need to be migrated for any flow f_b in U_b and any flow f_c in U_c, respectively. We then calculate the update cost of the two update events as follows:

$$Cost(U_b) = \sum_{b=1}^{v} sum(F_b) \text{ and } Cost(U_c) = \sum_{c=1}^{w} sum(F_c). \quad (6)$$

Finally, to guarantee fairness to some extent, we compare the head-event with the two selected update events and execute the one with the smallest update cost first.

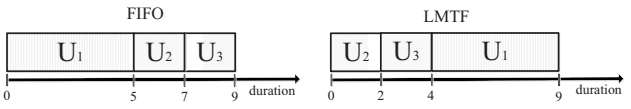
Opportunistic updating. The aforementioned scheduling model can decrease the average and tail ECTs at the cost of relaxing the fairness slightly. To improve the fairness level, we propose the opportunistic updating model based on the fine-tuning model. The basic idea is to find the first event that should be executed via the fine-tuning model and to perform other events which can be updated with the first event together. A heavy update event, which arrives earlier but is delayed by the fine-tuning model, thus has the chance to be quickly executed in the process of opportunistic updating. This model increases scheduling efficiency and improves fairness to some extent.

IV. Efficient methods for inter-event scheduling

In this section, we propose an approximation method to reduce the update cost of any single update event and reduce the ECT. Then, we design two inter-event scheduling methods, LMTF and P-LMTF, to guarantee the fairness and efficiency of a set of network update events.

A. Cost optimization method for any update event

For any update event, the following two issues dominate the event completion time. First, is it necessary to reroute some existing flows if desired paths lack sufficient bandwidth to transmit flows of an update event? If some existing flows need to be migrated, which paths should be reallocated to ensure sufficient bandwidth for migrated flows? Second, if needed, which existing flows should be migrated so that the network resource becomes just enough to accommodate the flow of an update event? This problem is NP-complete. Thus, we aim to design an approximation algorithm to determine the minimum set of migrated flows while providing sufficient bandwidth for flows in the update event and for those migrated existing flows.



(a) Initial update order (b) Fine-tunig update order

Fig. 3. Our LMTF can reduce the average ECT against the FIFO. U_1, U_2, U_3 stand for three update events with various event durations.

Any migrated flow for an update event competes for network resources, such as link bandwidth and switches with other traffic during the rerouting process. This behavior further affects the network ability to handle more update events. Migrating existing flows would take non-trivial time and decrease the completion process of the update event. Thus, it is crucial to reduce the update cost, i.e., the amount of migrated existing flows, so as to reduce the ECT for each update event. However, for a waited event in a queue of update events, its update cost usually changes due to the dynamics of network traffic. This brings more challenges to reduce the update cost.

B. Fine-tuning the order of update events

The aforementioned optimization method can effectively decrease the completion time of a single update event. It is unknown how to schedule multiple update events that form a queue according to their arrival order. The scheduling order determines not only the average and tail ECTs (two metrics about the efficiency), but also the fairness among such update events. In this environment, FIFO is attractive because it is simple to implement and guarantee strict fairness. If the durations of such update events are similar, FIFO is proven to be optimal for minimizing the tail ECT and achieving tight fairness [15]. If the duration exhibits heavy-tailed distribution, FIFO usually leads to the head-of-line blocking of heavy update events arriving earlier and increases the average and tail ECTs. In this scenario, FIFO guarantees the strict fairness, but fails to supply efficiency to update events. For a queue of update events, we first focus on decreasing the average and tail ECTs at the cost of slightly relaxing the fairness requirement.

An intrinsic method is to reorder all queued events based on their update costs and choose the lowest-cost event to execute firstly. Fig. 3 gives an example of scheduling three update events. The execution time of each update event is 1 second. The update cost is 4 seconds for event U_1 and 1 second for events U_2 and U_3, respectively. The average ECT of such update events is $(5+7+9)/3=7$ seconds, and the tail ECT is 9 seconds under the FIFO method, as shown in Fig. 3(a). If we order those update events according to their update costs, the ideal sequence is shown in Fig. 3(b). The average ECT is reduced to $(2+4+9)/3=5$ seconds and the tail ECT is the same. Theoretically, such a reordering of all update events could tackle the head-of-line blocking problem, and hence, reduce the wait time of the lower-cost events that arrive later. As discussed in Section IV, this method suffers huge computation and time overhead, the loss of fairness, and other issues.

In this paper, we propose the least migration traffic first (LMTF), a lightweight but effective scheduling method. It prefers to schedule update events based on their arrival order, but dynamically fine-tunes the execution sequence of a few selected events to tackle the head-of-line blocking problem. The basic idea is to randomly sample $\alpha \geq 1$ update events except for the head-event from the queue to find the one with the lowest update cost. At the same time, we compare the chosen event with the head-event to guarantee fairness in some degree. Finally, the update event with the lowest cost among the $\alpha+1$ candidates is identified to be performed first. If the initial head-event is selected, the FIFO fairness is guaranteed in this round. Otherwise, it is a heavy event and will initially block all of the α candidates. The head-of-line blocking problem, however, is well-tackled by selecting the update event with the lowest cost among $\alpha+1$ events.

Note that LMTF does not persist in sampling α update events when the queue contains less than $\alpha+1$ update events. The evaluation results indicate that our LMTF method effectively decreases the average and tail ECTs for any queue of update events even when the sampling number α is set as 2. This is a regular pattern explained in load-balance theory of the power of two random choices [16].

It is worth noting that our LMTF method improves scheduling efficiency by relaxing fairness. That is, the fine-tuning of a few update events may delay the execution of heavy update events which arrive earlier. There is room for further improvement of the fairness level, while decreasing the average and tail ECTs. To this end, we propose the opportunistic updating method based on our fine-tuning method.

C. Opportunistic updating

A common feature of the simple FIFO method, the intrinsic recording method, and our fine-tuning method is the sequential update. That is, the network only executes one update event in each round, which could be either the current head-event or a selected event from the queue. We concentrate on identifying multiple update events from the queue that can be executed in the same round if they exist.

To this end, we present the opportunistic updating method, a general inter-event scheduling policy to improve fairness. The basic idea is to find the first event that should be executed via the fine-tuning policy and to perform other update events together with the first event if possible. A heavy update event that arrives earlier would be scheduled later by the fine-tuning method. In the opportunistic updating process, it, however, would be checked whether it can be updated with the selected event together. If yes, it has a chance to be quickly executed. Obviously, the level of fairness among update events can be improved to some extent. Moreover, the updating efficiency can also be improved due to the chance of parallel update.

A heuristic schedule method, parallel-LMTF (P-LMTF), is proposed to realize the design. In the first step, we form a candidate set, consisting of the initial head-event and other α update events randomly sampled from the queue. The one with the lowest update cost among the $\alpha+1$ candidates is selected as the new head-event that will be executed first, just like the LMTF does. In the second step, the other α candidates would be checked if it is feasible to be performed with the new head-event together, according to their arrival orders. That is, the second step offers priority to update events that have arrived earlier, and hence, effectively improves fairness.

Note that P-LMTF does not check the entire queue to search for events that can be executed at the same time as the new head-event. The reason is that this behavior causes huge computation and time overhead, especially for large-scale networks and update events.

V. Experimental Evaluation

We start with the settings of our trace-driven evaluations in an 8-pod Fat-Tree datacenter network. We then compare the performance of our event-level and flow-level abstractions about network update. Finally, we evaluate our LMTF and P-LMTF scheduling methods over a real data-set from Yahoo!'s data center [11], against the FIFO method.

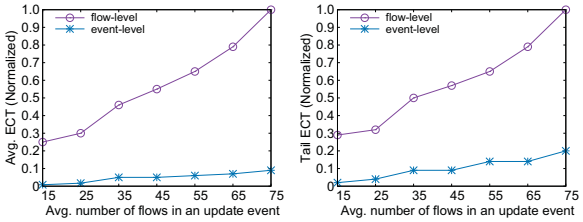
A. Evaluation settings

Topology. We consider an 8-pod Fat-Tree [17] datacenter network where the bandwidth of each link is fixed as 1 Gbps. In a Fat-Tree data center, the number of servers and switches is determined by the setting of parameter k. A Fat-Tree topology accommodates $5k^2/4$ switches and $k^3/4$ servers, which form k pods. The parameter k is set to 8 in our experiments.

Workloads. To enable the trace-driven evaluation of our methods and related work, we inject a large amount traffic into the Fat-tree datacenter as background traffic, so that the network utilization grows up to 70%. All injected flows are generated from a real traffic data-set from Yahoo!'s data center [11]. This trace records the basic information of each flow, including IP addresses of both source and destination servers, the size and duration of each flow, etc. Note that the real IP addresses in the trace are anonymous. We use a hash function to map the IP addresses of the source and destination of each flow into our datacenter network.

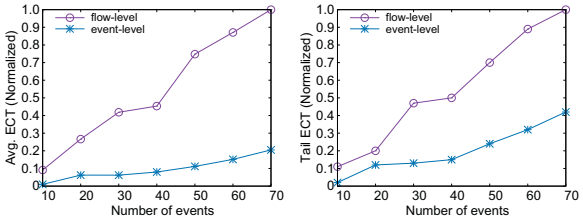
We further generate a set of heterogeneous network update events which differ in the number of flows, flow sizes, and flow durations. We set the average number of flows caused by each update event as a random integer ranging from 10 to 100. We then generate new flows for each update event according to the characteristics of network traffic mentioned in [12]. For a generated flow, the IP addresses of its source and destination are selected randomly over the entire data center.

Metrics. For a queue of network update events, we evaluate the benefits of event-level abstraction by comparing five metrics of the flow-level scheduling method, our LMTF method, P-LMTF method. They are the total update cost of all update events, the average ECT, the tail ECT, the total plan time, and the event queuing delay. The update cost of an update event means the amount of migrated traffic of existing flows. The average and tail ECTs indicate average and tail completion times of all events in the queue. The total plan time indicates the duration of making the update plan for all queued events.



(a) Average ECT (b) Tail ECT

Fig. 4. The changing trends of two metrics under 10 update events when network utilization approaches 70%. The average number of flows in each event increases from 15 to 75.



(a) Average ECT (b) Tail ECT

Fig. 5. The changing trends of two metrics for different numbers of update events when network utilization is 70%. Note that the number of flows in each event ranges from 10 to 100 randomly.

The event queuing delay is the time from the moment an event gets placed in an update queue until its execution starts.

We first compare the flow-level and event-level scheduling methods for a queue of update events. We then regulate several essential factors to evaluate their impacts on the performance metrics. They are the number of queued events, the type of queued events, and the event queueing delay.

B. Flow-level vs. Event-level scheduling methods

To evaluate the effectiveness of our event-level abstraction, we compare it with the flow-level method for any queue of update events. We construct a set of update events, each of which has 10 to 100 flows. The duration of each flow is set according to the characteristics of trace data. We report only the normalized results of each metric, which are achieved by dividing the maximum value of the flow-level method.

Fig. 4 plots the average and tail ECTs of 10 update events, when the average number of flows in each event varies from 15 to 75. The average and tail ECTs of our event-level method are, respectively, up to $10\times$ and $6\times$ faster than the flow-level method. Additionally, the growth of the average ECT is relatively smooth at the beginning of the two methods. In this stage, each update event contains only a few flows and consequently, has less probability meeting heavy flows. Thus, the last flow of an update event will not be delayed too long by other events. After the average number of flows in an event exceeds 35, the two metrics of the flow-level method increase notably. In this scenario, there must exist several heavy flows that block the later events in the queue. The metrics of our event-level scheduling maintain a relative and stable increase.

We can infer from Fig. 5 that more events in the queue would lead to larger average and tail ECTs for both methods. More precisely, the event-level scheduling elevates the average and tail ECTs by $5\times$ and $2\times$, respectively, over the flow-level method on average. The two metrics of the flow-level method exhibit sudden increases when the number of events reaches 30. Thus, some heavy flows that block the latter flows in update events must exist. Our method can effectively solve this problem because later events with lower update costs may achieve high priority and be executed first.

In summary, the flow-level scheduling method processes flows according to their arrival order, no matter which event they belong to. However, all flows in an update event are grouped and scheduled together. Thus, our event-level scheduling method can significantly reduce the average and tail ECTs.

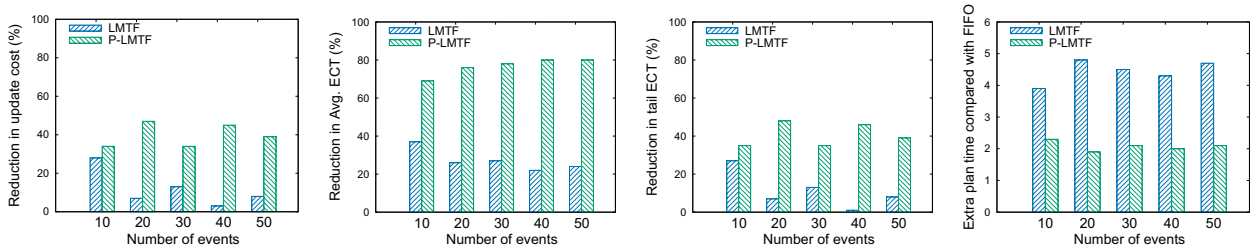
C. Impact of the number of update events

We evaluate our LMTF and P-LMTF scheduling methods against the fairness scheduling method FIFO as we vary the number of update events in the queue. The parameter α is set as 4, which means we chose 4 events randomly from the update queue to compare update cost with the head-event chosen by our two methods. The network utilization varies from 50% and 70% while each update event has 10 to 100 flows. Fig. 6 plots the evaluation results.

Fig. 6(a) reports reduction in the total update cost of our method against the FIFO method. P-LMTF achieves a stable reduction by $34\%-45\%$ as the number of queued events varies from 10 to 50. In this setting, LMTF also reduces the total update cost, but its achievement is always smaller than P-LMTF. As expected, the changing trends of average and tail ECTs are similar to the total update cost. For example, in the setting of 20 update events, the reduction in the update cost by LMTF descends a lot. At the same time, the average and tail ECTs also decrease to some extent, especially the tail ECT. That is to say, there must exist multiple heavy update events in the update queue that perform before light events. Such heavy events arrive earlier, and therefore, block later events; hence, they increase the ECT of some latter events. This scenario, however, will not affect the performance of P-LMTF. As the analysis in Section IV-C shows, P-LMTF improves LMTF by appending an opportunistic updating process and has a better performance.

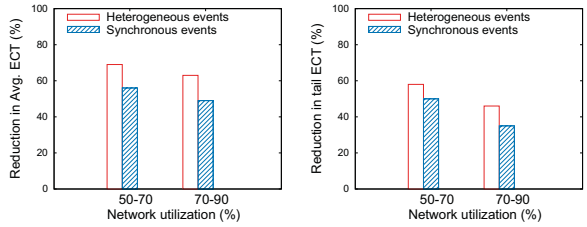
Fig. 6(b) indicates that P-LMTF achieves $69\%-80\%$ reduction in the average ECT compared with FIFO and LMTF achieves $22\%-36\%$ reduction. This significant improvement of P-LMTF comes from the introduction of opportunistic updating. It permits multiple events to be updated simultaneously if possible, i.e, queued events that arrive earlier have a chance to be executed at the same time as the head-event. Thus, P-LMTF further decreases the average ECT and simultaneously guarantees fairness among update events. On the other hand, P-LMTF reduces the tail ECT by $35\%-48\%$ and LMTF reduces $5\%-26\%$ against with FIFO, as shown in Fig. 6(c).

Finally, we measure the total plan time for all queued update events, as shown in Fig. 6(d). As expected, FIFO takes the least time since it does not conduct other actions during the decision process. Our LMTF and P-LMTF methods cause longer plan time because LMTF calculates the update costs for $\alpha+1$ update events to find the new head-event with lowest update cost. However, LMTF and P-LMTF take about 4.5



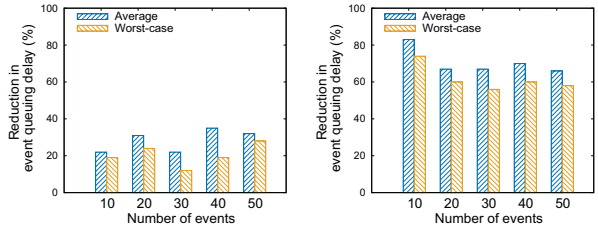
(a) Total update cost (b) Average ECT (c) Tail ECT (d) Total plan time

Fig. 6. Reductions of three metrics against FIFO and the changing trend of plan time of our scheduling methods under different numbers of events where the network utilization fluctuates between 50% and 70% and α=4. Note that the number of flows in each update event is a random integer from 10 to 100.



(a) Average ECT (b) Tail ECT

Fig. 7. Reduction in the two metrics with our P-LMTF method against FIFO for various type of events under different network utilization. The number of update events is 30, and α=4.



(a) LMTF (b) P-LMTF

Fig. 8. Reduction in average event queuing delay with our methods against FIFO under different number of events where the network utilization fluctuates from 50% to 70% and α=4.

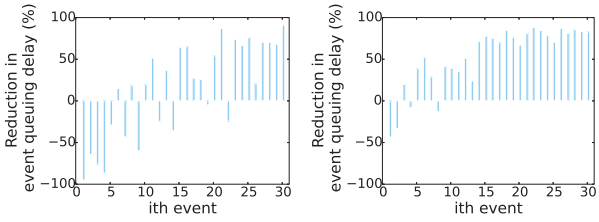
times and 2 times more plan time than FIFO, respectively, regardless of the number of update events. P-LMTF takes less plan time than LMTF since it has a chance to make an update plan for multiple events simultaneously in one round. It is an acceptable tradeoff to achieve significant reductions in the other three metrics at the cost of taking some extra plan time.

D. Impact of the type of events

We now observe the performance of our methods under different type of events. In particular, we examine two events: (1) heterogeneous events, each of which contains 10 to 100 flows; (2) synchronous events, each of which consists of 50 to 60 flows. For this set of experiments, our focus is on the different events, so we keep the background traffic static. As shown in Fig. 7, P-LMTF provides 60% – 70% reduction in average ECT and 40% – 60% reduction in tail ECT for heterogeneous events and 40% – 50% and 30% – 50%, respectively, for synchronous events when the network utilization fluctuates from 50% to 90%. As the figure shows, P-LMTF supports both heterogeneous event and isomorphic event update and is almost not affected by the network utilization.

E. Event queuing delay

We first study the average event queuing delay during a network update with multiple events. Then, we observe the



(a) LMTF (b) P-LMTF

Fig. 9. Reduction in event queuing delay with our methods against FIFO where the network utilization fluctuates from 50% to 70% and α=4. Note that the number of update events is 30 and the number of flows in each update event ranges from 10 to 100.

queuing delay of 30 events in the update queue. Our results are depicted in Fig. 8 and Fig. 9.

Fig. 8 plots the reduction in the event queuing delay with LMTF and P-LMTF compared with FIFO for the heterogeneous events. As expected, with the increase of event number, the benefits have a stable fluctuation. LMTF provides 20% – 40% and 10% – 30% reduction in average and worst-case event queuing delay. With P-LMTF, the worst case event queuing delay reduces by 60% – 74% and the average by 67% – 83%. We can infer that the event queuing delay in P-LMTF mainly depends on the executed events, network utilization, etc., not on the number of queued events. Further, in Fig. 9, we can clearly see a reduction in the queuing delay of each event with LMTF and P-LMTF against FIFO. Because of the fine-tuning of the execution sequence, LMTF leads to the delay of several update events. P-LMTF, based on the LMTF, provides more opportunities for events with a high update cost to be executed earlier, and thus, reduces the queuing delay of an update event and guarantees fairness during network update.

VI. Related Work

Consistent update. Reitblatt et al. present the concept of per-packet/per-flow consistent network update [2]. This means that a packet/flow traverses the network according to either the old network configuration or the new configuration. In addition, they propose a two-phase method to guarantee a consistent update with a version tag at each packet or flow. Katta et al. try to reduce the overhead of keeping new and old configurations at related switches at the cost of increasing the overall update duration [3]. Dionysus finds a consistent migration sequence by searching through a dependency graph of possible migrated steps [9]. Moreover, the timed-based update methods [4], [5] aim to achieve consistent update using the accurate time to trigger a network update at each phase.

Such methods effectively reduce the duration of flow rules on the switches and the update duration. In addition, the authors in [8] propose an effective method to decide if a consistent update is possible. Foerster et al. study the power of two random choices in consistent network update for updating forwarding rules in a loop-free manner and migrating flows without congestion [18]. They propose an effective algorithm to migrate two-splittable flows and an alternative when no consistent migration exists.

Congestion-free update. zUpdate [1], SWAN [6], and Caesar [19] try to make a congestion-free update plan in advance for any update event. This plan contains a sequence of intermediate states from an initial network state to the final network state, such that the transition across any pair of adjacent states is lossless. However, this strategy has several drawbacks. First, achieving such an update plan means solving a series of LPs. The time complexity is very high for large-scale network and update events. Second, to guarantee a series of congestion-free transitions, a portion $(10\% - 15\%)$ of link capacity has to be left in advance, which decreases the network utilization [6]. Cupid locally constructs a dependency graph among key-nodes for congested links to avoid high overhead among updates guarantee congestion-free data plane update. This work also proposes a heuristic algorithm to update flow table consistently and effectively [20].

Accelerate the update process. B4 speeds up the update process, using the custom hardware [21]. The work in [7] speeds the update process at the cost of incurring a given level of congestion. The basic idea is to minimize the transient congestion during the network update and achieve a better tradeoff between the update speed and the transient congestion.

For any update event, the aforementioned updating schemes focus on optimizing flow-level metrics and do not perform well in optimizing event-level metrics. Moreover, they cannot capture the inter-event requirements in a queue of update events. In summary, prior updating schemes do not provide an abstraction to represent event-level update semantics. Event-level abstract, expressing the semantics of a collection of flows caused by an update event, is different from the network abstract Coflow [22], each of which is a collection of flows between two groups of machines with associated semantics and a collective objective.

VII. Conclusion

The network condition is constantly in flux, due to updating issues. Prior updating schemes tackle all flows caused by such updating issues individually, but ignore the event-level update requirements. In this paper, we propose an event-level abstraction of network update which groups flows of an update event and schedules them together. It can considerably reduce the update cost and completion time of any update event. We further propose two efficient approaches, LMTF and P-LMTF, to schedule multiple update events while improving update efficiency and preserving fairness. The trace-driven evaluations indicate that our event-level LMTF method achieves $10\times$ and $6\times$ speed-up in average and tail ECTs, respectively, than flow-level method. Our P-LMTF method reduces average ECT by 75% and tail ECT by 42% compared with FIFO, when the network utilization exceeds 70%.

Acknowledgment

This work is partly supported by the National Natural Science Foundation for Outstanding Excellent young scholars of China under Grant No. 61422214, National Basic Research Program (973 program) under Grant No. 2014CB347800, National Nature Science Foundation of China under Grant Nos. 71471175 and 61572511.

References

[1] H. H. Liu, X. Wu, M. Zhang, L. Yuan, R. Wattenhofer, and D. A. Maltz, "zUpdate: updating data center networks with zero loss," in *Proc. ACM SIGCOMM*, Hong Kong, 2013.

[2] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," in *Proc. ACM SIGCOMM*, Helsinki, Finland, 2012.

[3] N. P. Katta, J. Rexford, and D. Walker, "Incremental consistent updates," in *Proc. ACM SIGCOMM*, Hong Kong, 2013.

[4] T. Mizrahi, E. Saat, and Y. Moses, "Timed consistent network updates," in *Proc. SOSR*, Santa Clara, CA, 2015.

[5] T. Mizrahi, O. Rottenstreich, and Y. Moses, "Timeflip: Scheduling network updates with timestamp-based TCAM ranges," in *Proc. INFOCOM*, Hong Kong, 2015.

[6] C. Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven WAN," in *Proc. ACM SIGCOMM*, Hong Kong, 2013.

[7] J. Zheng, H. Xu, G. Chen, and H. Dai, "Minimizing transient congestion during network update in data centers," in *Proc. IEEE ICNP*, San Francisco, CA, 2015.

[8] S. Brandt, K.-T. Foerster, and R. Wattenhofer, "On consistent migration of flows in SDNs," in *Proc. INFOCOM*, San Francisco, CA, 2016.

[9] X. Jin, H. H. Liu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford, and R. Wattenhofer, "Dynamic scheduling of network updates," in *Proc. ACM SIGCOMM*, Chicago, Illinois, CA, 2014.

[10] A. Endres and H. Stork, "Fifo-Optimal placement on pages of independently referenced sectors," *Inf. Process. Lett.*, vol. 6, pp. 46–49, 1977.

[11] Y. Chen, S. Jain, V. K. Adhikari, Z.-L. Zhang, and K. Xu, "A first look at inter-data center traffic characteristics via yahoo! datasets," in *Proc. INFOCOM*, Orlando, FL, 2011.

[12] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. ACM SIGCOMM*, New Delhi, 2010.

[13] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow, "RSVP-TE: extensions to RSVP for LSP tunnels," *RFC Editor*, 2001.

[14] T. Qu, D. Guo, Y. Shen, X. Zhu, L. Luo, and Z. Liu, "Minimizing Traffic Migration During Network Update in IaaS Datacenters," *Accepted to appear at IEEE Transaction on Service Computing*, 2016.

[15] A. Wierman and B. Zwart, "Is Tail-Optimal scheduling possible?" *Operations Research*, vol. 60, pp. 1249–1257, 2012.

[16] M. Mitzenmacher, "The power of two choices in randomized load balancing," *IEEE Transaction Parallel Distribute System*, vol. 12, pp. 1094–1104, 2001.

[17] C. E. Leiserson, "Fat-trees: universal networks for hardware-efficient supercomputing," *IEEE transactions on Computers*, vol. 100, pp. 892–901, 1985.

[18] K. T. Foerster and R. Wattenhofer, "The power of two in consistent network updates: Hard loop freedom, easy flow migration," in *Proc. ICCCN*, Waikoloa, Hawaii, 2016.

[19] S. Ghorbani and M. Caesar, "Walk the line: consistent network updates with bandwidth guarantees," in *Proc. ACM HotSDN*, Helsinki, 2012.

[20] W. Wang, W. He, J. Su, and Y. Chen, "Cupid: Congestion-free consistent data plane update in software defined networks," in *Proc. 35th IEEE INFOCOM*, San Francisco, CA, 2016.

[21] 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," in *Proc. ACM SIGCOMM*, Hong Kong, 2013.

[22] M. Chowdhury and I. Stoica, "Coflow: A networking abstraction for cluster applications," in *Proc. ACM Workshop on Hot Topics in Networks*, Redmond, WA, USA, 2012.