Balancing Traffic Load for Devolved Controllers in Data Center Networks

Wanchao Liang†, Xiaofeng Gao†, Fan Wu†, Guihai Chen†, Wei Wei‡
†Department of Computer Science and Engineering, Shanghai Jiao Tong University
‡Department of Computer Science, Stanford University
lwcallenhome@gmail.com,{gao-xf, fwu, gchen}@cs.sjtu.edu.cn, wwei2@stanford.edu

***Abstract*—Using a centralized controller for resource management and coordination is a common practice in cloud services. For scalability concern, in recent literature a novel approach, namely devolved controllers, was proposed. Such approach splits the network into regions, while each controller only monitors a portion of the traffic. This technique alleviates scalability issue, but brings other critical problems, such as unbalanced work load among controllers and reconfiguration complexities. In this paper, we investigate the usage of devolved controllers for large-scale data centers, and design a new scheme to overcome shortcomings, and to improve system performance. We first define *Load Balancing problem for Devolved Controllers* (LBDC), and prove its NP-completeness. For LBDC, we design an f-approximation, where f is the largest number of potential controllers for a switch in the network. We also propose both centralized and distributed approaches to solve LBDC time effectively. The numerical results validate our designs, which become a solution to manage and coordinate large-scale data centers.**

I. Introduction

In recent years, data center has emerged as an infrastructure that holds thousands of servers and supports many cloud services like computing, group collaboration, storage and financial applications, etc. The fast proliferation of cloud computing has promoted a rapid growth of large-scale commercial data centers. Companies such as Amazon, Cisco, Google and Microsoft have made huge investments in Data Center Networks (DCNs) for improvement.

Typically, a DCN has a centralized controller to monitor, manage network resources, and update routing information [1] [2] [3]. For instance, Hedera [4] and SPAIN [5] use a controller to collect the traffic statistics and reroute flows. Controller also provides address look-up services for VM migrations [6].

However, for large-scale DCN with thousands of racks, centralized controller suffers from many problems, such as scalability and availability. Driven by unprecedent scale and control objectives, researchers tried to deploy multiple controllers in DCNs [7]–[11]. The concept of *devolved controllers* is thereby introduced in [7], where they used dynamic flow [8] to illustrate the detailed configuration. Devolved controllers are a set of controllers that function as an omniscient controller. However, none of them have the entire information. Instead, every controller only maintain partial information beforehand, thus reduces the workload significantly.

Multi-controller technique alleviates the scalability problem, but still has several issues to explore. Firstly, pre-computed multipaths must be recalculated and network must be reconfigured if we expand the current network, which brings updating difficulties and heavy load for computation. Secondly, the flow queries will send to every controller, which makes controllers relatively busy and ignores the distance between senders and receivers. Thirdly, the pre-computation is actually centralized when configuring the network, which does not fit the distributed and MapReduce related applications. With the advent of Software Defined Network (SDN) [13] as proposed by OpenFlow [14], data center networking become revolutionizing in the industry, and several papers in designing distributed controller [9]–[12] appear. In [9], the authors overcome the limitation of statically configured mapping between switches and controllers, and propose migration protocol. Thus it is desirable to design an efficient strategy for devolved controllers to better manage the network traffic, and reduce routing cost.

Motivated by these challenges, in this paper, we propose a novel scheme to manage the traffic within OpenFlow framework. In our scheme, each controller monitors the traffic of switches locally. When traffic load imbalance occurs, some controllers will migrate part of their work to other controllers to keep the workload dynamically balanced. We define this problem as *Load Balancing problem for Devolved Controllers* (LBDC). We prove that LBDC is NP-complete, and then design one linear programming with deterministic rounding approximation, one centralized and one distributed algorithm, to dynamically balance the traffic load among controllers. Such methods can avoid the emergence of traffic hot spot, which will degrade network performance. Also, these schemes can significantly improve the availability and throughput of DCN. To the best of our knowledge, we are the first to discuss workload balancing problem among multi-controllers in DCNs, which has both theoretical and practical significance.

The rest of the paper is organized as follows. Section II presents the scenario and problem statement. Section III and IV presents LBDC solutions. Section V exhibits our performance evaluation. Finally, Section VI concludes the paper.

II. Problem Statement

Traffic in DCN can be considered as Virtual Machine (VM) communication. VMs in different servers collaborate together to complete designated tasks. In order to communicate between VMs, communication flow will go through several switches.

Base on OpenFlow [14] concept, each switch has a flow

This work was supported by the State Key Development Program for Basic Research of China (973 project 2014CB340303 and 2012CB316201), China NSF grant (61202024, 61272443 and 61133006), Shanghai Science and Technology fund (13PJ1403900, 12PJ1404900, 12ZR1445000, and 12ZR1414900), and Shanghai Educational Development Foundation (Chenguang Grant No.12CG09). X. Gao is the corresponding author.

table, and one responsibility of a controller is to modify these flow tables when communication occurs. Every controller composes of several hierarchical switches. Moreover, every rack has a server called *designated server* [15], which is responsible for aggregating, processing and sending the traffic statistics to the controller. By receiving these data, the controller assigns a routing component to compute flow reroute. Then the controller installs the new route to all associated switches by modifying their flow tables.

Now we define our problem formally. In a typical DCN, denote s_i as the ith switch, with the corresponding traffic weight $w(s_i)$, which is defined as the number of out-going flows. Next, given n switches $S = \{s_1, \cdots, s_n\}$ and m controllers $C = \{c_1, \cdots, c_m\}$, we make a weighted *m-partition* for switches such that each controller will monitor a subset of switches. The weight of a controller $w(c_i)$ is the weight sum of its monitored switches. Due to physical limitations, assume every s_i can only be monitored by its potential controller set $PC(s_i)$. Every c_i can only control switches in its potential switch set $PS(c_i)$. After the partition, the real controller and switch subset is denoted by $rc(s_i)$ and $RS(c_i)$ respectively.

The symbols used in this paper are listed in Table I:

TABLE I. DEFINITION OF TERMS

Term	Definition
S, s_i	switch set consists of n switches: S=$\{s_1, \cdots, s_n\}$
$w(s_i)$	weight of s_i, defined as the number of out-going flows.
$PC(s_i)$	Potential Controllers set of ith switch.
$rc(s_i)$	the real controller of ith switch.
C, c_i	controller set consists of m controllers: C=$\{c_1, \cdots, c_m\}$
$w(c_i)$	the weight of ith controller, sum of $RS(c_i)$'s weight.
$PS(c_i)$	Potential Switches set of ith controller.
$RS(c_i)$	Real Switches set of ith controller.
$AN(c_i)$	Adjacent Nodes (1-hop neighborhood) of ith controller.

To keep the quality of network management, each controller should have nearly the same workload. Otherwise, if all switches always communicate with the same controller, it will bring bottleneck congestion, hence downgrade performance. To precisely quantify the balancing performance among controllers, we define *Standard Deviation* of the partitions' weights as the metric, denoted by $\sigma = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(w(c_i) - \overline{w(c)})^2}$, where $\overline{w(c)}$ is the average weight of controllers. If the traffic flows vary as system running and the weight of c_i grows explosively, then we must regionally migrate some switches in $RS(c_i)$ to other available controllers to reduce its workload and keep the traffic balanced.

Then our problem becomes balancing the weight among m partitions. We define this problem as *Load Balancing problem for Devolved Controllers* (LBDC). In our scheme, each controller can dynamically migrate or receive switches to keep load balanced. Figure 1 illustrates the migration pattern.

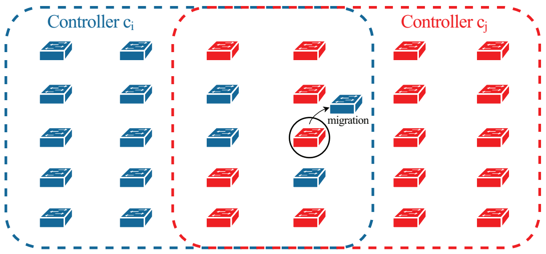

Fig. 1. An example of regional balancing migration. Controller c_j dominates 17 switches and Controller c_i dominates 13 switches. The traffic between c_i and c_j is unbalanced, and c_j is migrating one of its switch to c_i.

Define $x_{ij} = \begin{cases} 1 & \text{If } c_i \text{ monitors } s_j \\ 0 & \text{otherwise} \end{cases}$, Then LBDC can be further formulated as an programming:

$$\min \quad \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(\sum_{j=1}^{n} w(s_i)\cdot x_{ij} - \overline{w(c)}\right)^2} \tag{1}$$

$$s.t. \quad \overline{w(c)} = \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n} w(s_j)\cdot x_{ij} \tag{2}$$

$$\sum_{i=1}^{m} x_{ij} = 1, \quad \forall 1 \le j \le n \tag{3}$$

$$x_{ij} = 0, \quad \text{if } s_j \notin PS(c_i) \text{ or } c_i \notin PC(s_j), \forall i, j \tag{4}$$

$$x_{ij} \in \{0,1\} \quad \forall i, j \tag{5}$$

Eqn.(1) is the standard deviation, Eqn.(2) calculates the average weight among controllers, Eqn.(3) means each switch should be monitored by exactly one controller, Eqn.(4) is the regional constraints, while Eqn.(5) is the integer constraints.

Theorem 1. *LBDC is NP complete.*

Proof. We will prove the NP completeness of LBDC by considering a decision version of the problem, and show a reduction from PARTITION problem [16]. An instance of PARTITION is: given a finite set A and a $size(a) \in \mathbb{Z}^+$ for each $a \in A$, is there a subset $A' \subseteq A$ such that $\sum_{a\in A'} size(a) = \sum_{a \in A\backslash A'} size(a)$? Now we construct an instance of LBDC. In this instance there are 2 controllers c_1, c_2 and $|A|$ switches. Each switch s_a represents an element $a \in A$, with weight $w(s_a) = size(a)$. Both controller can control every switch ($PS(c_1) = PS(c_2) = \{s_a | a \in A\}$). Then, given a YES solution A' for PARTITION, we have a solution $RS(c_1) = \{s_a | a \in A'\}$, $RS(c_2) = \{s_a | a \in A\backslash A'\}$ with $\sigma = 0$. The reverse part is trivial. The reductions can be done within polynomial time, which completes the proof. □

Next we present our solutions for LBDC. We implement the scheme within OpenFlow, which changes the devolved controllers from a mathematical model into an implementable prototype. Also, our scheme is topology free, which is scalable for any DCN topology like Fat-Tree, BCube, Portland, etc.

III. Linear Programming and Rounding

Given the traffic status of the current DCN with devolved controllers, we can solve LBDC using programming (1)-(5). To simplify this programming, we can transfer it into a similar integer programming. Firstly, we can convert the standard deviation (1) to the sum of absolute values:

$$\min \quad \frac{1}{m}\sum_{i=1}^{m}\left|\sum_{j=1}^{n} w(s_i)\cdot x_{ij} - \overline{w(c)}\right| \tag{6}$$

Then we rewrite Eqn.(6), and obtain the integer programming:

$$\min \quad \frac{1}{m}\sum_{i=1}^{m} y_i \tag{7}$$

$$s.t. \quad y_i \ge \sum_{j=1}^{n} w(s_i)\cdot x_{ij} - \overline{w(c)} \tag{8}$$

$$y_i \ge \overline{w(c)} - \sum_{j=1}^{n} w(s_i)\cdot x_{ij} \tag{9}$$

$$\overline{w(c)} = \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n} w(s_j)\cdot x_{ij} \tag{10}$$

$$\sum_{i=1}^{m} x_{ij} = 1, \quad \forall 1 \le j \le n \tag{11}$$

$$x_{ij} = 0, \quad \text{if } s_j \notin PS(c_i) \text{ or } c_i \notin PC(s_j), \forall i, j \tag{12}$$

$$x_{ij} \in \{0,1\} \quad \forall i, j \tag{13}$$

In general, integer programs may not be easily solved in polynomial time, so we adopt *relaxation* to transfer our integer programming into a linear programming (LP). Then we can get a fractional solution and round it to a feasible solution of the original integer programming. To obtain the linear programming, we replace Eqn.(13) with $x_{ij} \geq 0$ $(\forall i, j)$.

After solving this LP, we recover a feasible solution to LBDC by a deterministic rounding [17] stated as follows:

Algorithm 1: Deterministic Rounding (LBDC-R)

```
1 for Each switch $s_j$ do
2     Search the solution space of LP:
3     Let $l = \arg\max_i \{x_{ij} \mid 1 \leq i \leq m\}$;
4     if $\exists$ several maximal $x_{ij}$ then
5         Let $l = \arg\min_i \{\sum_{j=1}^{n} w(s_j) \mid$ each max $x_{ij}\}$
6     Round $x_{lj} = 1$;
7     for $c_i \neq c_l$ do
8         Round $x_{ij} = 0$;
```

For instance, if the switch j has $x_{1j} = 0.2, x_{2j} = 0.7, x_{3j} = 0.1$ in the solution space of LP, then according to Alg. 1, we round $x_{2j} = x_{lj} = 1$, $x_{1j} = x_{3j} = 0$. We claim that the solution is feasible for LBDC.

Theorem 2. *LBDC-R results in a feasible solution for LBDC.*

Proof. According to LBDC-R, for each s_j, we only round the maximum $x_{ij} = 1, \forall 1 \leq i \leq m$, all the other $x_{ij} = 0$. Each switch is dominated by only one controller and no switches are idle. Thus we can get a feasible solution for LBDC. □

Next we analyze the performance of LBDC-R. We define Z^*,Z^{LP} and Z^R as the integer programming solution, linear programming solution and the solution after the rounding process respectively. f is defined as the maximum number of controllers in which any switch potentially appears. More formally, $f = \max_{i=1,\dots,n} |PC(s_i)|$. We claim that LBDC-R is an f-approximation. To prove it, we first prove lemma 1 and 2.

Lemma 1. $\overline{w(c)^{LP}} = \overline{w(c)^*} = \overline{w(c)^R}$

Proof. From the definition of the original $\overline{w(c)}$, the ideal weight of each controller is the sum of the weight of all switches divided by the number of controllers. This definition is suited for all the solution space, thus we can conclude that $\overline{w(c)^{LP}} = \overline{w(c)^*} = \overline{w(c)^R} = \frac{1}{m}\sum_{i=1}^{n} w(s_i)$. □

Lemma 2. $x_{ij}^R \leq x_{ij}^{LP} \cdot f$

Proof. We have the constraint $\sum_{i=1}^{m} x_{ij}^{LP} = 1$ $(\forall 1 \leq j \leq n)$. Also x_{lj}^{LP} is the largest of all x_{ij}^{LP} $(\forall 1 \leq i \leq m)$, then by Pigeonhole principle, we must have $x_{lj}^{LP} \cdot f \geq 1$. Since for each s_j, x_{lj}^R equals to 1 and others equal to zero, which is less than or equal to the corresponding LP solution times the f factor. Then for any c_i, we have $x_{ij}^R \leq x_{ij}^{LP} \cdot f$. □

Theorem 3. *LBDC-R is an f-approximation algorithm.*

Proof. Since the LP is a relaxation, we have $Z^{LP} \leq Z^*$. Also we have $Z^* \leq Z^R$ because the solution of LBDC-R is feasible by Theorem 2, while Z^* denotes the optimal solution.

Because $\overline{w(c)}$ means the ideal weight of each controller, it must be the same in all the solutions according to Lemma 1, thus we let $\overline{w} = \overline{w(c)}$. From $Z^{LP} \leq Z^*$ we can derive:

$$\frac{1}{m}\sum_{i=1}^{m}\Big|\sum_{j=1}^{n} w(s_i)\cdot x_{ij}^{LP} - \overline{w}\Big| \leq \frac{1}{m}\sum_{i=1}^{m}\Big|\sum_{j=1}^{n} w(s_i)\cdot x_{ij}^{*} - \overline{w}\Big|$$

Since we already know $|x| - |y| \leq |x| + |y|$, we can get:

$$\frac{1}{m}\sum_{i=1}^{m}\Big|\sum_{j=1}^{n} w(s_i)\cdot x_{ij}^{LP}\Big| \leq \frac{1}{m}\sum_{i=1}^{m}\Big|\sum_{j=1}^{n} w(s_i)\cdot x_{ij}^{*}\Big| + 2\overline{w}$$

The approximation ratio can be obtained by the following:

$$\begin{aligned}
\frac{1}{m}\sum_{i=1}^{m}\Big|\sum_{j=1}^{n} w(s_i)\cdot x_{ij}^{R} - \overline{w}\Big| &\leq \frac{1}{m}\sum_{i=1}^{m}\Big(\Big|\sum_{j=1}^{n} w(s_i)\cdot x_{ij}^{R}\Big| + \overline{w}\Big) \\
&\leq \frac{1}{m}\sum_{i=1}^{m}\Big|\sum_{j=1}^{n} w(s_i)\cdot x_{ij}^{LP}\cdot f\Big| + \overline{w} \\
&\leq f\cdot\frac{1}{m}\sum_{i=1}^{m}\Big|\sum_{j=1}^{n} w(s_i)\cdot x_{ij}^{*}\Big| + (1+2f)\overline{w} \\
&= f\cdot OPT + (1+2f)\overline{w}
\end{aligned}$$

Therefore LBDC-R is an f-approximation. □

IV. Algorithm Design

Linear programming and rounding can solve LBDC theoretically. But solving an LP is time consuming and not practical for real-world applications. Therefore it is essential to design efficient and applicable algorithms. In this section, we propose centralized and distributed greedy algorithms for LBDC. Centralized scheme is suitable for relatively small scale DCNs, while distributed is natural for huge scale DCNs.

A. Centralized Migration

Centralized Migration splits into two phases. First we need to configure and initialize the DCN. As the traffic changes dynamically, we come to the dynamical migration phase.

Centralized Initialization: First we need to initialize DCN and assign switches to controllers, satisfying the load balance requirement. So we design centralized initialization algorithm (LBDC-CI). In order to get rid of dilemmas when selecting conflicted switches/controllers, we first present *Break Tie Law*.

Break Tie Law: 1) When choosing s_i from S, we select the largest weight one. If there are several switches, the one with the smallest $|PC(s_i)|$ is preferred. If there are still several candidates, pick randomly. 2) When choosing c_i from C, we select the minimum weight one. If there are several controllers, the one with the smallest $|RS(c_i)|$ is preferred. If there are still several candidates, we choose by physical distance. Finally, if we still cannot make decision, pick randomly.

Then we design LBDC-CI as shown in Alg. 2.

Algorithm 2: Centralized Initialization (LBDC-CI)

```
Input  : $S$ with $w(s_i)$; $C$ with $w(c_i)$;
Output: An $m$-Partition of $S$ to $C$
1 $RemList$={$s_1$,$s_2$,$\cdots$,$s_n$};
3 while $RemList \neq \emptyset$ do
4     Pick $s_i$ from $RemList$;
5     if $|PC(s_i)| = 1$ then
6         Assign $s_i$ to its unique controller in $PC(s_i)$;
7     else
8         Assign $s_i$ to the $c_j$ with min $w(c_j)$ in $PC(s_i)$;
9     Remove $s_i$ from $RemList$;
```

LBDC-CI needs $O(n)$ to assign the switches in $RemList$. In *while* loop, it takes $O(f)$ to select a c_j. Hence the worst case running time is $O(n^2)$. If we store the $RemList$ in priority heap, we can reduce the overall running time as $O(n)$.

As system runs, traffic may vary frequently and affect the balanced status. Correspondingly, we design the centralized migration algorithm (LBDC-CM) to alleviate the situation.

Centralized Regional Migration: Since we must assess when the controller should execute migration, we set a threshold to judge the traffic status. When the controller's traffic degree exceeds the threshold, we regard this controller as unbalanced that needs migration. Some measurement studies [18] [19] of data center traffic have shown that data center traffic is expected to be linear. We set the threshold upon the current traffic sample and the history record to mimic RTT and Timeout of TCP [20]. This linear expectation use two factors α and β depending on the traffic features of DCN, where $0 \leq \alpha \leq 1$ and $\beta > 1$. We divide the time into several rounds and run LBDC-CM periodically. Then we use Thd and Efn to denote the parameters of threshold and effluence, Avg_{last} and Avg_{now} to represent the average workload of the last and the current sample round. In each round, we sample the current weight of each node and calculate $Avg_{now} = \Sigma w(c_i)/m$. The Linear Expectation can be computed as follows:

$$Thd = Avg_{now}\alpha + Avg_{last}(1-\alpha), \;\; Efn = \beta \times Thd$$

The core principle of LBDC-CM is migrating heavy switches to light controllers greedily. Figure 2 and Alg. 3 illustrates the workflow and procedure of centralized migration.

Algorithm 3: Centralized Migration (LBDC-CM)

```
Input: $S$ with $w(s_i)$; $C$ with $w(c_i)$;
       $PendList = OverList = \{\emptyset\}$;
1  Step 1: Add $c_i \rightarrow OverList$ if $w(c_i) > Efn$;
2  Step 2: Find $c_m$ of max weight in $OverList$;
3  if $\exists c_n \in AN(c_m) : w(c_n) < Thd$ then
4      repeat
5          Pick $s_m$ of max weight in $c_m$, refer $PC(s_m)$:
6          if $\exists c_f \in AN(c_m)$ && $w(c_f) < Thd$ then
7              Send $s_m \rightarrow c_f$;
8          else
9              Ignore the current $s_m$ in $c_m$;
10     until $w(c_m) \leq Thd$ or $w(c_f) \geq Thd$;
11     if still $w(c_m) > Efn$, move $c_m$ to $PendList$;
12 else
13     Move $c_m$ from $OverList$ to $PendList$;
14 Step 3: Repeat Step 2 until $OverList = \{\emptyset\}$;
15 Let $OverList = PendList$, Repeat Step 2 until
   $PendList$ become stable;
16 Step4: Now $PendList$ has several connected
   components $CC_i (1 \leq i \leq |CC|)$;
17 for each $CC_i \in CC$ do
18     Search the $\bigcup_{c_j \in CC_i} AN(c_j)$;
19     Compute $avg_{local} = \frac{w(CC_i \cup AN(CC_i))}{|CC_i|+|AN(CC_i)|}$;
20     if $w(c_j) > \gamma \cdot avg_{local}$, where $c_j \in CC_i$ then
21         Migrate the $s_{max} \in RS(c_j)$ to
           $c_{min} \in AN(CC_i)$ repeatedly until
           $w(c_j) \leq \gamma \cdot avg_{local}$;
22         remove $c_j \in CC_i$ from $PendList$;
23 Step5: Repeat Step 4 until $PendList$ become stable.
```

LBDC-CM searches $OverList$ to find c_m in Step 2, which takes $O(n)$. Next, it migrate switches from $OverList$, which takes $O(n^2)$. Step 3 invokes Step 2 several times until $OverList$ is empty and makes the $PendList$ become stable, which takes $O(n^3)$. Step 4 and Step 5 balance the $PendList$ locally as Step 2 and 3, so the worst case running time is $O(n^3)$. Also by storing the $OverList$ and $PendList$ in priority heap, we can reduce the complexity to $O(n^2)$.

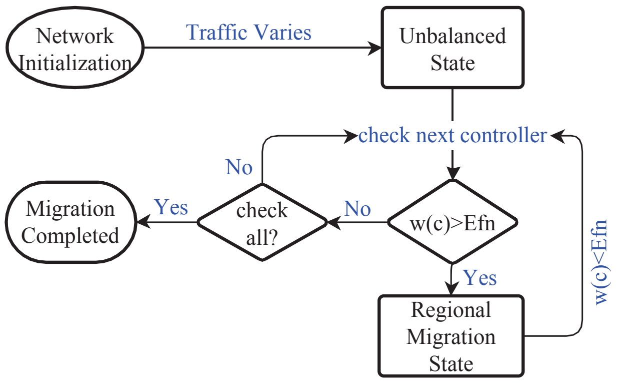

Fig. 2. Dynamic Load Balancing Workflow of LBDC

B. Distributed Migration

Centralized algorithm is suitable for relatively small-scale network because of its accuracy. But for large-scale network, we need to design a faster and practical distributed algorithm [21]. We assume a synchronous environment to perform our two phase algorithm.

Distributed Initialization: During this phase, we assign each switch a controller randomly by message communication. Alg. 4 illustrates the distributed initialization procedure.

Algorithm 4: Distributed Initialization (LBDC-DI)

```
1 Send "CONTROL" message to my own $PS(c_{my})$;
2 $s_i$ reply the first-come "CONTROL" message with
  "YES", all the other messages after that with "NO";
3 Move each $s_i$ with "YES" from $PS(c_{my})$ to $RS(c_{my})$;
4 Wait until all the switches in $PS(c_{my})$ reply, terminate;
```

After initialization, we design the distributed migration algorithm (LBDC-DM) to balance the workload dynamically.

Distributed Regional Migration: In this phase, the controller uses the threshold to decide whether it should start migration. Since it only access the neighborhood, the threshold is not a global one, but an independent value computed by each controller locally. The algorithm runs periodically in several rounds. In each round, each controller samples $AN(c_i)$ and applies Linear Expectation again:

$$\begin{aligned} Avg &= \frac{\sum_{c_k \in AN(c_i)+c_i} w(c_k)}{|AN(c_i)|+1} \\ Thd &= Avg_{now}\alpha + Avg_{last}(1-\alpha) \\ Efn &= \beta \times Thd \end{aligned}$$

A controller monitors its traffic status by local threshold. When the traffic degree is larger than Efn, it enters sending state and initiate a transaction to transfer heavy switches to neighbors. Alg. 5 illustrates the distributed migration procedure.

Algorithm 5: Distributed Migration (LBDC-DM)

```
1  if $\geq Efn \rightarrow$ *sending* then
2      if $\exists c_i \in AN(c_{my})$ *in receiving or idle* then
3          add $c_i \rightarrow RList$(receiving > idle);
4      repeat
5          Pick $s_{max}$ with max weight, refer $PC(s_{max})$,
           find $c_j$(in $RList$) with min weight, send
           "HELP[$c_i, s_{max}$]" to $c_j$, then check response:
6          if *response*="ACC" then
7              $c_{my}$ start migration with $c_j$ and $s_{max}$.
8          else if *response*="REJ" then
9              remove $c_j$ from $RList$, find next $c_j$, send
               "HELP" again, check response.
10     until $w'(c_i) \leq Efn$;
11 else if $\leq Thd \rightarrow$ *receiving* then
12     When receiving "HELP" messages:
13     repeat
14         receive switches for $c_j$ and send back "ACC";
15     until $w(c_j) + w(s_{max}) \geq Thd$;
16     Now all "HELP" messages will reply "REJ"
17 else if $(Thd, Efn) \rightarrow$ *idle* then
18     When receiving "HELP" message:
19     repeat receiving state until
       $w(c_j) + w(s_{max}) > Efn$;
20     Now facing other "HELP"s, controller will reply
       "REJ" and enter the sending state;
```

It is easy to prove the features of a distributed algorithm such as termination, agreement, and validity for Alg. 5, which indicates its correctness and efficiency.

V. Performance Evaluation

We evaluate the performance of our scheme by considering the case of traffic demand changes and examine whether the metric of balanced workload is minimized. We also take the number of migrated switches into consideration. Furthermore, we check how different parameters will impact the results.

A. Environment Setup

We place 10000 switches and 100 controllers in a 100×100 m^2 square. Switches are evenly distributed, that is, a switch is $1m$ away from its neighbors. The controller is also evenly distributed and each one is $10m$ away from its neighbor. Each controller can control the switches within $30m$, and can communicate with other controllers within the range of $40m$. We assume the weight of each switch follows Pareto distribution with its parameter $\alpha = 3$. Now we set $\alpha = 0.8, \beta = 1.2, \gamma = 1.5$ in default.

B. Controller Number

Figure 5 uses the default configuration described above, except that the number of controllers varies from 20 to 100. We first apply LBDC-CI and change the traffic demands dynamically to emulate unpredictable user requests. Then we apply LBDC-CM to ease the spot congestion. We use the metric described in Section II to evaluate the performance. In Fig. 5, we compare the standard deviation of the initial bursty traffic state and the state after migration. We find that after migration, the metric decreases. As the number of controllers is increasing, the improvement ratio is also increasing. It is quite intuitive that more controllers will share jobs to reach balanced load. This figure also shows that our algorithm has pretty good performance when the number of controllers grows, which indicates our scheme is suitable for huge DCNs.

Figure 6 shows that our distributed algorithm has the effect of minimizing our metric. The performance of the LBDC-DM is poor when the number of the controllers is relatively small. This phenomenon is attributed to the fact that devolved controllers can only cover switches within $30m$. When the number of controllers is small, more switches can only be controlled by one particular controller without much choices. As the number of the controller increases, LBDC-DM has better performance and larger improvement ratio.

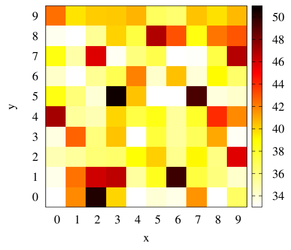

Fig. 3. Colormap before migration

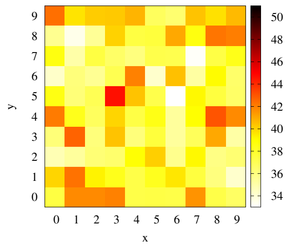

Fig. 4. Colormap after migration

C. Centralized Protocol vs. Distributed Protocol

Next, we compare our centralized and distributed protocols by changing the number of controllers from 30 to 210 with a step of 20. The results are shown in Fig. 7. It depicts that the performance of centralized version is much better than the distributed version, but the difference between them is decreasing as the number of the controllers increases.

We then assess the migration phase. Figure 3 and 4 shows the effectiveness of our migration algorithms. We consider a scenario: at the beginning of a time slot, the weights of switches are updated and we run the migration algorithm. The weight of switches follows Pareto distribution with its parameter $\alpha = 3$. Figure 8 shows the performance of LBDC-CM dynamically, which significantly reduce the metric. Figure 9 shows the performance of LBDC-DM. As expected, the performance metric is pretty large after initialization because LBDC-DI just assigns each switch a controller without load balancing consideration. As time goes by, the performance increases greatly.

Next we compare the number of migrate switches of centralized and distributed scheme, and information from Fig. 10 shows that the metric of centralized algorithm fluctuates at a certain value while the metric of distributed algorithm is decreasing and becomes stable as time goes by. This is because LBDC-CM can migrate switches from a global perspective. Therefore every time the number of migrated switches is almost the same. But LBDC-DM only migrates locally, so the number of migrated switches reduce significantly and become stable slowly.

D. Parameter Specification

Next we explore the impact of the threshold parameters α, β, γ. Here α is a parameter to balance conservativeness and radicalness. We examine the impact of changing α. Due to Step 4 in LBDC-CM, the impact is relatively small. β is a

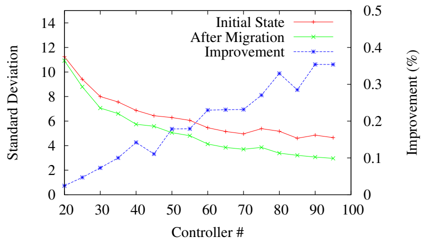

Fig. 5. Improvement of different number of controllers in centralized migration

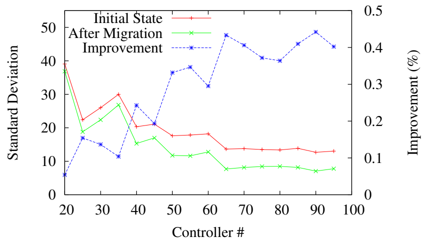

Fig. 6. Improvement of different number of controllers in distributed migration

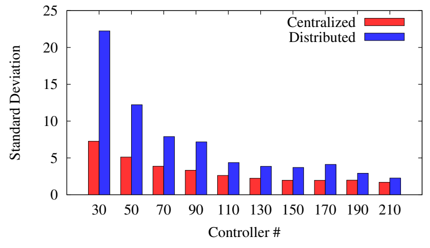

Fig. 7. Performance comparison between centralized and distributed algorithm

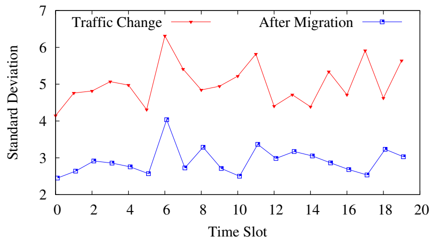

Fig. 8. Centralized migration traffic statistics at different time slot

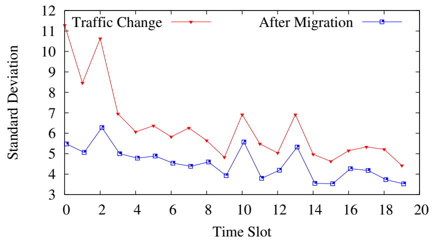

Fig. 9. Distributed migration traffic statistics at different time slot

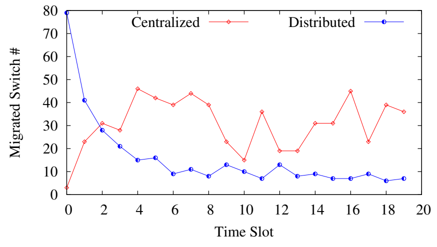

Fig. 10. Migrated switches of centralization and distributed migration at different time slot

crucial parameter which decide whether to migrate or not. We set different value for β and see the impact of changing β. TABLE II list the statistics for β ranging from 1.1 to 1.5. Clearly, the improvement rate and the number of migrated switches is decreasing as β increases, which is correct from the threshold definition. γ is used in Step 4 of LBDC-CM and the effect of γ is similar to β, so we omit the discussion.

TABLE II. INFLUENCE OF β FACTOR

β	Initial	After Migration	Rate	switch no.
1.1	150.279	96.165	0.541	376
1.2	157.080	107.749	0.414	356
1.3	166.194	123.509	0.365	316
1.4	166.904	130.265	0.327	259
1.5	151.475	119.928	0.287	196

VI. CONCLUSION

As the evolution of DCNs, the usage of a centralized controller is the performance bottleneck of the DCN and the traffic management problem becomes severer. In this paper, we have explored the usage of *devolved controllers* to manage the data center effectively as well as alleviate the scalability issue. In order to monitor and manage the traffic of data centers, we have developed a new implementable scheme to overcome the shortcomings such as workload congestions and reconfiguration complexities. We have further defined *Load Balancing problem for Devolved Controllers* (LBDC) and given its NP-completeness. We have also provided an f-approximation solution, designed applicable centralized and distributed algorithms to balance the workload among controllers in dynamic situation. The feature of traffic load balancing ensures scaling efficiently, enhances responsiveness of client's requests as well as improves the throughput and the availability of DCNs. Our performance evaluation validates our design, which becomes a solution to monitor, manage, and coordinate large-scale data centers.

REFERENCES

[1] M. Al-Fares, A. Loukissas, and A. Vahdat. "A scalable, commodity data center network architecture," *ACM SIGCOMM*, 63-74, 2008.

[2] B. Heller, S. Seetharaman, et al. "ElasticTree: Saving Energy in Data Center Networks" *USENIX NSDI*, 249-264, 2010.

[3] S. Kandula, J. Padhye, and P. Bahl. "Flyways to de-congest data center networks." *ACM Hotnets-VIII*, 2009.

[4] M. Al-Fares, S. Radhakrishnan, et al. "Hedera: Dynamic flow scheduling for data center networks," *USENIX NSDI*,19-19, 2010.

[5] J. Mudigonda, et al. "SPAIN: COTS Data-Center Ethernet for Multipathing over Arbitrary Topologies." *USENIX NSDI*, 265-280, 2010.

[6] A. Greenberg, J. Hamilton, N. Jain, et al. "VL2: a scalable and flexible data center network," *ACM SIGCOMM*, 51-62, 2009.

[7] AS-W. Tam, K. Xi, and H. Chao. "Use of devolved controllers in data center networks." *IEEE INFOCOM*, 596-601, 2011.

[8] AS-W. Tam, K. Xi, and H. Chao. "Scalability and Resilience in Data Center Networks: Dynamic Flow Reroute as an Example," *IEEE GLOBECOM*, 1-6, 2011.

[9] A. Dixit, F. Hao, et al. "Towards an elastic distributed sdn controller." *ACM SIGCOMM*, 7-12, 2013.

[10] A. Tootoonchian, Y. Ganjali. "HyperFlow: A distributed control plane for OpenFlow." *USENIX INM/WREN*, 3-3, 2010.

[11] C. Macapuna, C. Rothenberg, M. Magalhaes. "In-packet Bloom filter based data center networking with distributed OpenFlow controllers." *IEEE GLOBECOM*, 584-588, 2010.

[12] S. Yeganeh and Y. Ganjali. "Kandoo: a framework for efficient and scalable offloading of control applications." *ACM HotSDN*, 19-24, 2012.

[13] T. Koponen, et al. "Onix: A Distributed Control Platform for Large-scale Production Networks." *USENIX NSDI*, 1-6, 2010.

[14] N. McKeown, T. Anderson, H. Balakrishnan, et al. "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM*, 69-74, 2008.

[15] T. Benson, A. Anand, A. Akella, and M. Zhang. "Microte: fine grained traffic engineering for data centers," *ACM CONEXT*, 8, 2011.

[16] R. Karp, "Reducibility among Combinatorial Problems", *Springer US*, 85-103, 1972.

[17] D. Williamson and D. Shmoys. "The design of approximation algorithms." *Cambridge University Press*, 2011.

[18] S. Kandula, S. Sengupta, et al. "The nature of data center traffic: measurements & analysis," *ACM SIGCOMM*, 202-208, 2009.

[19] T. Benson, A. Akella, and D. Maltz. "Network traffic characteristics of data centers in the wild," *ACM SIGCOMM*, 267-280, 2010.

[20] D. Corner, "Internetworking with TCP/IP." Vol 1, Page 226, 2000.

[21] N. Lynch. "Distributed algorithms." *Morgan Kaufmann*, 1996.