2015 44th International Conference on Parallel Processing

Sheriff: A Regional Pre-Alert Management Scheme in Data Center Networks

Xiaofeng Gao, Wen Xu, Fan Wu, Guihai Chen
Shanghai Key Laboratory of Scalable Computing and Systems,
Department of Computer Science and Engineering,
Shanghai Jiao Tong University, Shanghai, 200240, China
gao-xf@cs.sjtu.edu.cn, xuwen.sjtu@gmail.com, fwu@sjtu.edu.cn, gchen@cs.sjtu.edu.cn

***Abstract*—As the base infrastructure to support various cloud services, data center draws more and more attractions from both academia and industry. A stable, effective, and robust data center network (DCN) management system is urgently required from institutions and corporations. However, existing management schemes have several problems, including the difficulty to manage the entire network with heterogeneous network components by a centralized controller; and the short-sighted mechanism to deal with resource allocation, congestion control, and VM migration.**

In this paper, we design Sheriff: a distributed pre-alert and management scheme for DCN management. Sheriff is a regional self-automatic control scheme at end host side to balance network traffic and workload. It includes two phases: prediction and management. Each end-host predicts possible overload and congestion by prediction strategy based on ARIMA and Neural Network methodology, and perform an ALERT message. Delegated local controllers then monitor their dominating region and activate localized protocols VMMIGRATION to manage the network. We illustrate the predication accuracy by network traces from a local data center service provider; examine the management efficiency by simulations on both Fat-Tree topology and Bcube topology; and prove that VMMIGRATION is an approximation with ratio $3+\frac{2}{p}$ where p is a constant predefined in local search algorithm. Both numerical simulations and theoretical analysis validate the efficiency of our design. In all, Sheriff is a fast and effective scheme to better improve the performance of DCN.

I. INTRODUCTION

With the rapid development of cloud computing, data center networks (DCNs) become indispensable infrastructures for cloud services like web search, e-business, and social networking. IT companies have made tremendous investments in DCNs to support their online services and cloud platforms. The performance of DCNs impacts directly on the quality of cloud services, which implies the significance of efficient DCN management. However, existing DCN management system has several drawbacks discussed as follows.

This work has been supported in part by the State Key Development Program for Basic Research of China (973 project 2014CB340303), the National Natural Science Foundation of China (Grant number 61202024, 61422208, 61472252, 61272443 and 61133006), the Natural Science Foundation of Shanghai (Grant No.12ZR1445000), Shanghai Educational Development Foundation (Chenguang Grant No.12CG09), Shanghai Pujiang Program 13PJ1403900, and in part by Jiangsu Future Network Research Project No. BY2013095-1-10, CCF-Intel Young Faculty Researcher Program and CCF-Tencent Open Fund. We also would like to thank Wei Wei, Xuanzhong Wei, Tao Chen, and Zhangxuan Gu for their contributions on the early versions of this paper.

Centralization vs Distribution: Typically, DCN has a centralized manager to perform network management [1]–[5]. A centralized manager configures routes and distributes routing information to end-hosts. It must apply parallelism and fast route computation approaches to fit increasing scale of modern data centers and meet the demands of the traffic characteristics [6]. Towards the real-time web applications in production data centers, the response time of the centralized manager may increases sharply when the enormous parallel requests occur. Hotspots [3], [7] and elephant flows [1], [8] will also exacerbate the performance of DCNs by increasing considerable latency. Some request flows may have expired while the centralized manager still works with the out-of-date information. Additionally, with the rapid growth of cloud applications, the scope of DCN expands tremendously in recent years [9], possibly with upgraded or heterogenous components, bringing difficulty to centralized manager to monitor and control every running state of the system. To overcome the above shortcomings, *distributed managers*, also referred to as *devolved controllers*, are introduced to leverage the workload and provide quick responses for real-time web applications [10]–[15]. Currently ONOS V1.1.0 supports running multiple controllers in a clustered mode when the underlying OpenFlow switches are connected to more than one controller [16], which makes devolved controller management a practical strategy to manage a DCN system.

Contingency vs Pre-Control: Servers are always oversubscribed (especially when they host hundreds of virtual machines). Such scheme incurs unwanted problems like overload, low average resource efficiency, bandwidth shortage, and high packet loss rate. Switches in DCN may have congestions because of bursty links and unbalanced traffic, resulting packets lost and greatly increasing job completion time. Existing schemes apply congestion control like VM migration [17]–[20], QCN [21]–[23], when congestion appears or devices are overloaded. Such contingency-oriented mechanism only works after detecting errors, which is harmful to device prevention and system maintenance. On the contrary, if we can detect the unusual condition ahead of time, precisely predict the future tendency and pre-control the system, we would protect system from damage and extend the system lifetime.

Since a certain application such as telecommunication has an explicit diurnal traffic pattern [24], the traffic or CPU uti-

0190-3918/15 $31.00 © 2015 IEEE
DOI 10.1109/ICPP.2015.76

lization of a web application in the data centers is predictable. We can enable servers to forecast performance and report to distributed managers. The managers then take some early warnings and react in advance to avoid congestions. Some researchers also considered workload prediction as one way to multiplex resource demands, alleviate traffic congestion, and reduce hotspot effect [25], but they did not consider distributed management and balancing problem between each clusters.

Consequently, in this paper we design a novel distributed pre-alert and management scheme for DCN, named **Sheriff**. It is a regional self-automatic control scheme at end host side to balance traffic and workload according to reasonable predictions, such that the system will achieve a global optimization. Sheriff includes two phases: prediction and management. Each delegated controller predicts possible overload and congestion of the entities in its dominating region by prediction strategy based on ARIMA and Neural Network time series techniques, and perform ALERT messages. They then activate local protocols VMMIGRATION to migrate workload at server side and improve network performance. In all, by simply inserting a shim layer on each rack, Sheriff can automatically monitor its dominating region and provide quick response when it detects unusual situations. Instead of balancing workload after detecting congestion or device overloading, Sheriff predicts possible trend by time series forecasting. It solves potential problems before they actually happens, which protects devices and reduce maintenance costs tremendously.

We provide three ways to prove the efficiency of Sheriff: First, we illustrate a network trace prediction evaluation with real application data from ZopleCloud Corporation. Next, we prepare various simulations on typical switch-centric DCN and server-centric DCN such as Fat-Tree and Bcube to evaluate our management algorithm. Finally, we give a theoretical proof showing that VMMIGRATION is an approximation with ratio $3 + \frac{2}{p}$, where p is a constant in Local Search Algorithm. Both numerical experiments and theoretical proof validate the efficiency of Sheriff. To the best of our knowledge, we are the first to propose a pre-alert management scheme for DCN with distributed managers, which has both theoretical and practical significance in the related areas.

The rest of paper is organized as follows. In Sec. II, we provide the preliminaries and overview of Sheriff system design. We formulate the problem in Sec. III. Then, Sec. IV presents the pre-alert mechanism and distributed migration algorithms are shown in Sec. V-B. Finally, we provide an evaluation of our system including theoretical analysis in Sec. VI. Sec. VII and Sec. VIII are related works and conclusion.

II. PRELIMINARIES AND SYSTEM DESIGN

A. System Environment and Constraints

Facility and Environment Settings: We use standard rack with 0.6m wide, 2m tall, 1m deep and each rack is partitioned into 42 rack units (denoted by U). A typical server occupies 1-2U and switches occupy 1U for a top-of-rack (ToR) switch up to 21U for large aggregation and core switches [9]. Note that all equipments must be placed in a rack and racks are positioned alongside each other to form rows in data centers with approximately 2m space between two rows [26].

Network Settings: We take Fat-Tree [27] as an example to illustrate our design, which can be easily implemented in other DCN topologies. We suppose that a DC rack comprises 40 servers connected to an ToR switch which has 48 1Gbps and up to 4 10Gbps ports. The ToR is connected to aggregation switches (to network with other racks) with 10 Gbps links.

We adopt *virtual machines* (VMs) as the unit of resource allocation for multiple tenants in the cloud. In our model, we operate the systems by VMs on physical machines according to their dependency relations.

B. Distributed Controller and Shim Layer

To achieve distributed management, we need to partition the network into regions and deploy local managers for each region. According to the physical design of DCN, a natural choice is to append shim layer on each rack to make policy compliance mandatory by monitoring variations on each server and forcing all traffic into congestion-controlled tunnels. The shim layer should run by the management software in the virtualization or platform network stack, where it is well-isolated from tenant code. Shim layer should be responsible for the following supervisory tasks:

1) provisioning and monitoring local servers in the rack (e.g., Autopilot, Azure Fabric);
2) periodically checking flow samples coming into/out from the ToR switch of its local rack;
3) detecting the congestion feedback from aggregation switches, core switches and other ToR switches.

A *basic unit* of a DCN is the union of servers, ToR switch located at the same rack. A shim not only in charge of monitoring the basic unit, but also manages the local system according to difference situation. The detailed discussion of shim management will be shown in Sec III.

C. Symbols and Topology Construction

Firstly, define v_i as the ToR controller for each rack in DCN, which is responsible for managing and monitoring the servers inside each rack (usually it is combined together with the ToR switch). Easy to see, the Ethernet architecture within a rack in the network will not change during any working process for DCN. Therefore, using v_i as the smallest network unit in DCN is reasonable and precise.

Let $V = \{v_1, v_2, \ldots, v_n\}$ as the set of delegation nodes in DCN. Each v_i charges a set of servers (to avoid symbol confliction with switches, we denote server as *host*). Define $H_i = \{h_i^1, h_i^2, \ldots, h_i^{i_p}\}$ the set of hosts in rack v_i. Each rack contains a list of hosts/servers, thus let SR_i as the index set of hosts for each v_i. Set $\mathcal{SR} = \{\mathrm{SR}_1, \mathrm{SR}_2, \ldots, \mathrm{SR}_n\}$. Further, we can package multi-tier enterprise applications into *virtual machines* (VMs) and deploy VMs into physical servers. Correspondingly, set $M = \{m_1, m_2, \ldots, m_q\}$ the set of VMs in DCN. We map each VM to its host h_i by an index list VM_i. Let $\mathcal{VM} = \{\mathrm{VM}_1, \mathrm{VM}_2, \ldots, \mathrm{VM}_p\}$ as the collection

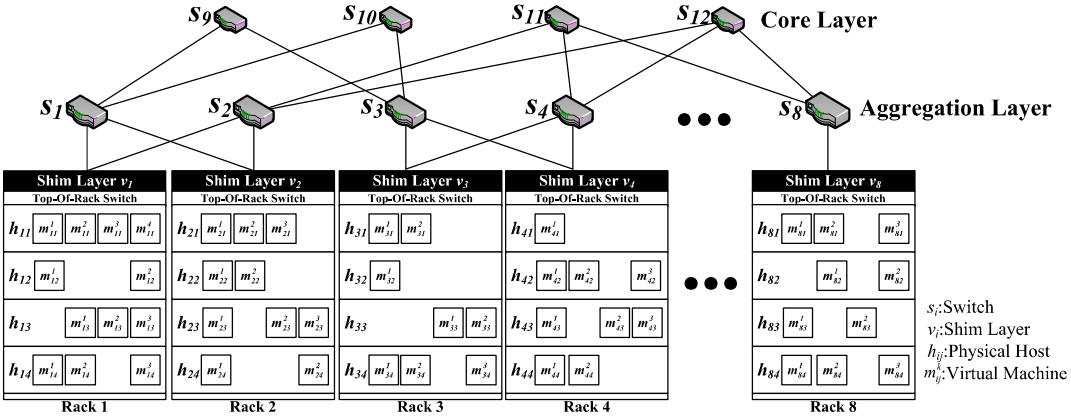

Fig. 1. The Architecture of Sheriff with Fat-Tree Topology

of such VM lists. Finally, define $S = \{s_1, s_2, \ldots, s_t\}$ as the set of aggregation switches and core switches in DCN.

Fig. 1 shows the architecture of Sheriff, using an eight pods Fat-Tree topology as an example. As it shows, every host h_{ij} could contains certain number of VMs (m_{ij}^k) and it connects to the shim layer v_i on the top of rack. Shim layer also connects to a certain number of switches. They all together construct the communication.

There are two kinds of graphs in a DCN, which are:

Wired Network Graph: Define $G_r = (V_r, E_r)$ as the underlying DCN. Intuitively, $V_r = V \cup S$, representing the combination of delegation nodes (shims) for each rack together with switches in the network. E_r contains two types of edges, one with notation (v_i, s_j), which means switch s_j connects to the ToR switch binding with v_i. Another type is (s_i, s_j), which means switch s_i is directly connected with s_j.

Dependency Graph: A dependency graph represents the relationship among pairs of VMs for their interdependencies and inter-communications patterns. Let $G_d = (V, E_d)$. Next, if $(v_i, v_j) \in E_d$, then there exist $h_{ia} \in v_i$, $h_{jb} \in v_j$, $m_{ia}^p \in h_{ia}$, $m_{jb}^q \in h_{jb}$, such that m_{ia}^p and m_{jb}^q are dependent with each other if any communication takes places between them. Due to inherent coupling between VMs and the resource shortages, two dependent VMs usually cannot reach an accommodation if they are hosted at the same physical server simultaneously [18]. Thus, the dependency graph can also be viewed as a conflict graph for VM migration.

In all, Table I summarizes main notations used in this paper.

TABLE I
MAIN NOTATIONS

Sym	Description	Sym	Description
$V = \{v_i\}$	Set of shim nodes	$C(e)$	Capacity of e
$S = \{s_i\}$	Set of switches	$D(e)$	Distance of e
$H_i = \{h_{ij}\}$	Set of hosts in v_i	$B(e)$	Available Bandwidth
$M_{ij} = \{m_{ij}^k\}$	Set of VMs in h_{ij}	$T(e)$	Transmission time
$G_r = (V \cup S, E_r)$	Regular wired graph	$P(e)$	Utility Rate
$G_d = (V, E_d)$	Dependency graph	ζ_{ij}^k	Location function
$N_d(v_i)$	v_i's neighbors in G_d	χ_{ij}^r	Edge function
$P(v_i, v_j)$	Path from v_i to v_j	B_l	Bandwidth threshold
$Cost(v_i, v_j)$	VM migration cost	C_r	Cost of initialization
p	Local change size	C_d	Unit cost per distance

III. PROBLEM FORMULATION

A. Potential Problems and Corresponding Solutions

As a server/switch may crash or overload under some specific circumstances, we do not take crash errors into consideration since we assume that they could be resolved by backup system. Next, we analyze the overload situation as follows:

1) Servers may be overloaded, for example, when its CPU utilization or memory utilization reaches up to 90%. Other conditions include outburst traffic, high power consumptions, etc. When facing these situations, we will migrate or reshuffle VMs to accommodate workload spikes and/or resource shortages.
2) Congestion may appear in switches and we can detect this by checking the content of the QCN feedback information [28] and modify the rate at end host to reach the goal of easing the congestion.

B. Pre-Alerts and Actions

Shim monitors servers and switches according to their feedbacks piggyback the value of target items.

1) The local computing device on each server will periodically collect information including CPU utilization rate, memory, disk I/O, uplink traffic through ToR to check the QoS performance and predict the future evolution of server's workload (as background service). Server will report an ALERT value to its dominating shim if ALERT exceeds the pre-defined THRESHOLD. The prediction phase applies classical time series ARIMA model and NN model, and then selects the best result by comparison, which will be illustrated in Sec. IV.
2) Similarly, each switch will detect the flow congestion and use the DSCP bits of the DS field in IP header (or VLAN Priority Code Point bit if DSCP is needed for other uses) for signaling congestion flows. Such bits can be modified through most existing congestion-control protocol (like OpenFlow). Alternatively, if DCN applies QCN-like protocols, it can return the sender a special feedback according to current queue length.

3) Specially, shim should monitor the uplink flow rate of its local ToR proactively and distinguish the possibility of uplink congestion. We also refer it as a kind of alert.

When a shim detects alerts from servers or switches, it will produce immediate reaction based on the following conditions:

Alert from Servers: If v_i receives an ALERT from its local host h_{ij}, it means h_{ij} cannot afford the working load from its VM's. v_i will then select a group of VM's in h_{ij} and try to migrate them into nearby neighbors to reduce overload of h_{ij}.

Alert from ToR Switch: If v_i receives alert from its ToR$_i$, it means shim detects a potential uplink congestion at ToR$_i$. At this moment shim should arrange a group of selected VM's and apply VM migration to release the workload of ToR$_i$ through wired links to neighbor racks.

Alert from Outer Switches: If v_i detects alerts from outer switch s_i, it will figure out the conflict flows from a set of local VM's. Then v_i should reroute portion of flows to their destinations without passing through hot switches.

Usually, live VM migration require additional memory, bandwidth, and short service downtime [17], which is more expensive and slower than flow rerouting. Thus shim will implement flow reroute first and then deal with VM migration.

C. Cost Functions

VM migration requires additional cost. We imply six-stage pre-copy live migration [17] from m_{ij}^k to m_{pq}^r:

1) **Initialization**: m_{ij}^k is selected for migration to host h_{pq}. Block devices mirrored and free resources maintained.
2) **Reservation**: Initialize m_{pq}^r container on target host.
3) **Iterative Pre-Copy**: RAM is sent in the first iteration. Enable shadow paging and copy dirty pages iteratively.
4) **Stop&Copy**: Suspend m_{ij}^k for a final transfer round.
5) **Commitment**: h_{pq} confirms successful transmission.
6) **Activation**: m_{pq}^r is active and resumes normal operation.

As shown in Fig. 2, we define t_1 as the initialization and reservation time, t_2 the pre-copy time; t_3 the final migration time, and t_4 the commitment and activation time. Set C_r as the cost of initialization, reservation, commitment, and activation process since it is hard to analyze these complicated stages involving CPU, memory, and storage. Without loss of generality, we assume C_r is the same for every VM, which is the *computing cost* of VM migration. Since the downtime in VM migration is a short period of 60ms [17], we ignore the influence of downtime and set the cost into zero.

Next, let us discuss *transmission cost*. Define $P(v_i, v_j)$ as the path from v_i to v_j (v_i can be h_{ij} or m_{ij}^k for simplicity), $C(e)$ as the maximum capacity of e. Next, set $D(e)$ as the physical distance of e, and $B(e)$ equals to the smaller one of current available bandwidth and bandwidth in request on e. Note that $B(e)$ must be greater than a threshold value B_l. Thus, the transmission cost is $\sum_{e\in P(v_i,v_p)}(\delta T(e) + \eta P(e))$, where $T(e) = \frac{m_{ij}^k.capacity}{B(e)}$ is the transmission time and $P(e) = \frac{B(e)}{C(e)}$ denotes the utilization rate of the bandwidth. δ and η are predefined parameters.

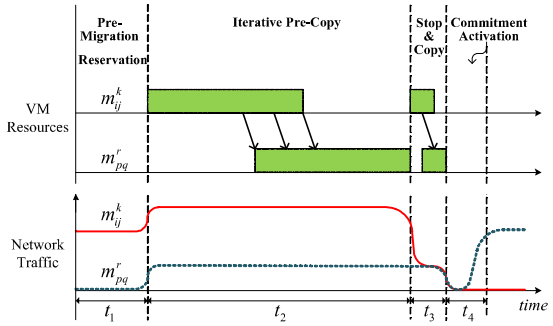

Fig. 2. Six Stage for Live VM Migration

Finally, we add *dependency cost* for VM migration. If G_d^i is the original dependency graph while G_d^p is the updated one if we complete VM migration from m_{ij}^k to m_{pq}^r. Next, the changes to communicate with m_{ij}^k to that with m_{pq}^r brings communication cost. Define $N_d(v_i)$ as the neighbor set of v_i in G_d (includes v_i), $G_r[N_d(v_i)]$ the induced graph from G_r with pathes to connect $N_d(v_i)$, and C_d the unit cost per distance in G_d, then we get the dependency cost is $\left(\sum_{e\in G_r[N_d(v_i)]} D(e) - \sum_{e\in G_r[N_d'(v_p)]} D(e)\right) C_d$. Let $G_z = G_r[N_d^i(v_i)] \setminus G_r[N_d'(v_j)]$ where N_d' denote the new state after m_{ij}^k is migrated to m_{pq}^r. To simplify this equation, we define characteristic function as

$$\chi_{ij}^r = \begin{cases} 1 & \text{if } \exists(v_i, v_j) \in G_r \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

All in all, the cost function of VM migration is

$$Cost(v_i, v_p) = C_r + C_d D(e)\chi_{ip}^z + \sum_{e\in P(v_i,v_p)} (\delta T(e) + \eta P(e)). \tag{2}$$

Define location function ζ as

$$\zeta_{ij}^k = \begin{cases} 1 & \text{if } m_{ij}^k \text{ is located on host } h_{ij} \text{ at } v_i, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

(similarly, $\zeta_{ij} = 1$ means h_{ij} is located on v_i.)

We formulate the global VM migration problem as below:

$$\min \quad \sum_{m_{ij}^k} \sum_{m_{pq}^r} Cost(m_{ij}^k, m_{pq}^r) \tag{4}$$

$$s.t. \quad \zeta_{ij}^k = 1, \quad \zeta_{pq}^r = 1 \tag{5}$$

$$v_p \in N_r(v_i) \tag{6}$$

$$\chi_{ij}^r = 0 \tag{7}$$

$$h_{pq}.capacity \geqslant m_{ij}^k.capacity \tag{8}$$

$$\sum_{\substack{m_{ij}^k \in h_{ij} \\ \text{ALERT}_{ij}^k \geqslant \text{THLD}}} k \geqslant 1 \tag{9}$$

$$\sum_{h_{ij}\in v_i} \sum_{m_{ij}^k \in h_{ij}} m_{ij}^k.capacity \leqslant \beta \cdot \text{ToR}_i.capacity \tag{10}$$

$$i, j, k, p, q, r \in \{1, 2, \ldots, n\}. \tag{11}$$

Eqn. (5) to Eqn. (11) describe the detailed requirements for VM migration explicitly. Obviously n is the largest number among all the indices mentioned above. Actually, this global VM migration problem is NP-Hard because it is reducible

from multiple knapsack problem.

IV. Pre-Alert Mechanism

Pre-alert scheme mainly contains three steps: (1). periodically collect information that may help for forecast. (2). respectively process each feature (e.g. network traffic) of collected information with prediction models that can best explain it. Here, we use the combination of *Autoregressive Integrated Moving Average* (ARIMA) model and *Nonlinear Autoregressive Neural Network* (NARNET) model to make predictions. (3). After predicting future value of all features, we propose a scheme to calculate an alert which indicates the seriousness of the condition of one VM. Finally, delegated controller collects alerts from all VMs in its dominating range every T seconds. Details will be discussed as follows.

A. Collecting Necessary Information

Each host m_{ij}^k will monitor information that may be useful to predict future state and in our scenario we mainly take CPU_{ij}^k, MEM_{ij}^k, IO_{ij}^k and TRF_{ij}^k into consideration. These four factors will be considered together as a *workload profile* $\mathbf{W}_{ij}^k = [\text{CPU}_{ij}^k,\ \text{MEM}_{ij}^k,\ \text{IO}_{ij}^k,\ \text{TRF}_{ij}^k]$ where CPU_{ij}^k represents the current CPU load of m_{ij}^k at time t and MEM_{ij}^k, IO_{ij}^k, TRF_{ij}^k present memory untility, disk IO rate and network traffic respectively. One thing to note is that each element of the workload profile should be normalized to $[0, 1]$.

Fig. 3-5 show the raw data of CPU, I/O, and Traffic records we collect from a local data center service provider. As we know, different applications may have different influence on hosts' performance. Prior works show hosts running MySQL tends to be CPU-bound, host running Apache/PHP tends to be memory-bound, whereas Map-Reduce applications may take up a lot of network bandwidth. When these applications use up resource it obtains, QoS may not be guaranteed. Thus, taking these factors into consideration is necessary.

Each v_i also monitors the queue length of the associated ToR switch periodically. Using the historic information about the queue length, we can predict future queue length. Since each machine is equipped with a pacer as in [8], great variance of the queue length is not likely to happen and this will make it easy for predicting the congestion in ToR switch.

B. Forecasting the Expected State

From evaluating the traces from a real-world data center, we find that traditional time series methods still work for DCN.

In the following part, we will mainly focus on the alert model and we will mainly discuss the time series of network traffic without loss of generality. The underlying techniques we use towards different features of $\mathbf{W}_{ij}^k$ are the same.

ARIMA: Given a time series of a feature of $\mathbf{W}_{ij}^k$, say, uplink traffic TRF_{ij}^k, denote it as $\{Y_t\}$ where t is an integer index and Y_t is a real number without any loss of generality. Define the lag operator L by $L^i = Y_{t-1}$ and the lag-1 difference operator ∇ are defined in the obvious way, i.e.,

$$\begin{cases} L^j Y_t = Y_{t-j}, \\ \nabla^j Y_t = \nabla(\nabla^{j-1} Y_t), \text{ for } j \geq 1, \text{ with } \nabla^0 Y_t = Y_t. \end{cases}$$

We can use Box-Jenkins method to specify the parameters of ARIMA model which can predict workload by learning trend, periodicity and autocorrelation in usage history.

For a particular series $\{Y_t\}$, we first difference the non-stationary series to remove periodicity and trends in $\{Y_t\}$ to obtain a stationary series $\{\widetilde{Y}_t\}$ by which we can apply an autoregressive moving-average (ARMA) process. Hence we have an $\text{ARIMA}(p, d, q)$ process that well explains the original time series $\{Y_t\}$. Now we have $\phi(L)\nabla^d Y_t = \theta(L) Z_t$ where $\{Z_t\} \sim WN(0, \sigma^2)$ denotes the uncorrelated white noise with zero mean, and $\phi(L) = 1 - \phi_1(L) - \ldots - \phi_p L^p$ and $\theta(L) = 1 + \theta_1 L + \ldots + \theta_q L^q$ are polynomial operators in L of degrees p and q.

Given $\{Y_1, \ldots, Y_t\}$, $P_t Y_{t+h}$ $(h > 0)$ is denoted as the h-step-ahead conditional mean prediction for Y_{t+h}, that is, the predicted value of Y_{t+h} given history data up to time t. Once the parameter for ARIMA is decided using Box-Jenkins method, we can derive the expected value of Y_{t+h} as follows:

1) One-Step-Ahead: In one-step-ahead prediction scenario, we can use the minimum mean square error (MMSE) forecast method to forecast future trend at one time unit. Since each time we should obtain a forecast range of the prediction result, we can use the method in to decide the predicted value, denoted as $\widetilde{Y}_{t+1}$.
2) K-Step-Ahead: The k-step-ahead value can be computed recursively using the one-step-ahead value as the historical data. Hence, we can effectively predict future workload for subsequent k time unit.

The result can be formulated as

$$P_t Y_{t+h} = (\nabla^d)^{-1} P_t \widetilde{Y}_{t+h}. \tag{12}$$

NARNET: ARIMA performs well when an initial differencing step can be applied to remove non-stationarity. However, ARIMA is a linear time series model and may not work otherwise. Applying neural network to work out a nonlinear model seems intuitive.

Here we choose nonlinear autoregressive neural network (NARNET) which can be trained to predict a time series from that series past values.The structure for an ordinary NARNET.

Let $\text{NARNET}(ni, nh)$ denotes a nonlinear autoregressive neural network with ni inputs and nh outputs. Such a model can be described algebraically as

$$Y_t = F(Y_{t-1}, Y_{t-2}, \ldots) + \varepsilon \tag{13}$$

where Y_t is the variable of interest, and ε is the error term. We can then use this model to predict the value of Y_{t+k}.

Dynamic Model Selection: Statistical methods and neural networks are commonly used for time series prediction. While ARIMA are good at modeling linear, dynamic signals, NARNET are reliable for modeling nonlinear, dynamic and chaotic. For a period of time, the time series $\{Y_{t-h}, Y_{t-h+1}, \ldots, Y_t\}$ may appear to conform to the assumption of one model. Therefore, instead of applying one model only, it would be better for us to make use of both models to better predict the workload profile. Another reason for use to select not a use

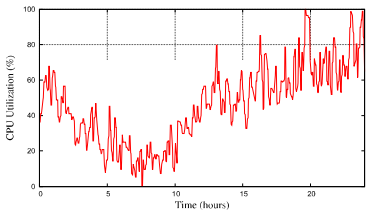

Fig. 3. Raw Data of CPU Utility

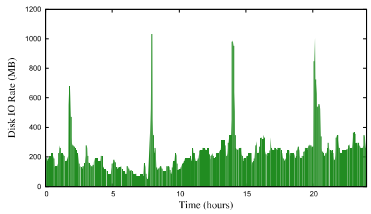

Fig. 4. Raw Data Of Disk I/O Rate

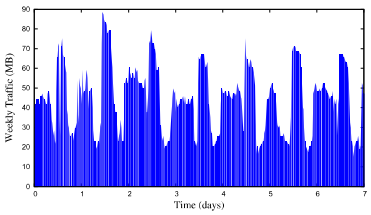

Fig. 5. Raw Data of Weekly Traffic

a single model is that it is pretty difficult to determine the parameter. For instance, different number of hidden layer nh can greatly influence the prediction accuracy.

We use the mean square predication error over a period T_p

$$\text{MSE}_f(t, T_p) = \frac{1}{T_p} \sum_{i=t-T_p+1}^{t} (\text{ERROR}_f(i))^2 \qquad (14)$$

with $\text{ERROR}_f(i)$ denote the error at time unit t as fitness metric for each method f at time t during period $[t - T_p, t]$.

Then we will choose the predicted value of model f among models of different parameters and methods with the minimum $\text{MSE}_f(t, T_p)$. For example, consider the case of four predictors, two of them are ARIMA model, $\text{ARIMA}(p_1, d_1, q_1)$ and $\text{ARIMA}(p_2, d_2, q_2)$ respectively, and two are NARNET model, $\text{NARNET}(ni_1, nh_1)$, $\text{NARNET}(ni_2, nh_2)$. To simplify the discussion, we use one-step-ahead prediction instead of k-step-ahead prediciton. We will take past T_p data as inputs to these four models and we will get T-seconds-ahead (one-step-ahead) predictions of each server's work portfolio. We will calculate the minimum value of $MSE_f(t, T_p)$ of each model f. Suppose $\text{ARIMA}(p_2, d_2, q_2)$ have the minimum $\text{MSE}_f(t, T_p)$, then we will use it as the predicted value.

C. Alert Scheme

After making the T-seconds-ahead predictions of each server's work load, we will describe how to use the workload profile of m_{ij}^k to give an alert to the delegation node indicating that it is in serious condition.

The seriousness of the condition can be evaluated using the following equation:

$$\text{ALERT}_{ij}^k = \begin{cases} \max(\mathbf{W}_{ij}^k) & \text{if } \exists x \in \mathbf{W}_{ij}^k (x > \text{THRESHOLD}), \\ 0 & \text{otherwise.} \end{cases}$$

After m_{ij}^k generated its ALERT_{ij}^k, m_{ij}^k can wait v_i for further commands. v_i will periodically collect alerts from its neighbor nodes and use these alerts to decide VMMIGRATION or FLOWREROUTE process. Details will be discussed in Sec. V-B.

V. ALERT-MIGRATION ALGORITHM

A. Centralized Alert-Migration Algorithm

Since the conditions and conclusions in VMMIGRATION problem are similar to the very famous problem the k-median to some extent, we can use some steps to transform the VMMIGRATION problem into a typical k-median problem. As a result, we not only prove that this problem is NP-hard, but give the newest k-median approximate algorithm and approximate ratio. Next we will transform VMMIGRATION problem into k-median problem.

1) Simplification: Consider the cost function we get before,

$$Cost(v_i, v_p) = C_r + C_d D(e) \chi_{ip}^z + \sum_{e \in P(v_i, v_p)} (\delta T(e) + \eta P(e)). \quad (15)$$

We can easily find that C_r is a constant so we just need to consider $C_d D(e) \chi_{ip}^z$ and $\sum_{e \in P(v_i, v_p)} (\delta T(e) + \eta P(e))$. Then we want to explain that dependent cost is only a function of v_i and v_p, transmission cost is a function of v_i, v_p and the edges between them. In other words, actually we can define functions $f(v_i, v_p) = C_d D(e) \chi_{ip}^z$ and $g(v_i, v_p, e_{ip}) = \sum_{e \in P(v_i, v_p)} (\delta T(e) + \eta P(e))$, in which e_{ip} are the edges between v_i and v_p. Then,

$$Cost(v_i, v_p) = C_r + f(v_i, v_p) + g(v_i, v_p, e_{ip}). \qquad (16)$$

Apparently, VMMIGRATION problem is all about the source and the destination of VM migration, involving only v_i, v_p and the edges between them (noted as e_{ip}). In the other words, transmission cost can be written as $g(v_i, v_p, e_{ip})$. Now we are trying to explain that the dependent cost is independent of the choice of path from the source to the destination. Since

$$\chi_{ip}^z = \begin{cases} 1 & \text{if } \exists (v_i, v_p) \in G_z \\ 0 & \text{otherwise.} \end{cases} \qquad (17)$$

and $G_z = G_r[N_d^i(v_i)] \setminus G_r[N_d^p(v_j)]$, for the fixed v_i and v_p, the $G_r[N_d^i(v_i)]$ and $G_r[N_d^p(v_p)]$ are fixed. Further more, G_z is fixed and finally $C_d D(e) \chi_{ip}^z$ is fixed. So we can use $f(v_i, v_p)$ to express the dependent cost. Till now we have proved the Eqn. (18)

2) Transformation: The next step is using efficient algorithm to transform $g(v_i, v_p, e_{ip})$ into $G(v_i, v_p)$. This means we can always choose a path that needs the smallest cost for two fixed v_i and v_p. First we construct a graph T whose vertices are all racks. All wired connections are the edges in T. Also every edge has a cost. Our aim is to find the best path between any two racks. This is a all pair shortest path problem. Since in T there are much more edges than vertices, we choose Floyd-Warshall algorithm. The time complexity is $O(n^3)$. So use this algorithm we finally get

$$Cost(v_i, v_p) = C_r + f(v_i, v_p) + G(v_i, v_p). \qquad (18)$$

To make it more clear, we can conclude that the cost between any two racks in T is just $Cost(v_i, v_p)$ and it do not dependent on the path between v_i and v_p.

The last step is to explain the VMMIGRATION problem as a k-median problem with what we have done above. Let C be the source ToRs and F be all the ToRs. The aim is to find a subset I of F, and connect every racks in C with I with the smallest cost. Apparently between every pair of racks there exists a cost, how to connect them is a k-median problem.

As for k-median problem, there is a constant ratio algorithm from [29] with a ratio $3+\varepsilon$. Further more, [30] find that the ratio can not be better than $1+2/\varepsilon$. All in all, we can solve the VMMIGRATION problem by this method.

B. Distributed Alert-Migration Algorithm

The distributed Alert-Migration algorithm has five subroutines. Note that each v_i runs a copy of algorithm locally.

Alg. 1 is the framework routine processing alerts and deciding which actions to take according to the type of alerts. We run it periodically every T time. At the beginning of each round, v_i collects alerts from its dominating region, use priority function in Alg. 2 to select a group of candidate VMs, then migrate some VMs in $\mathcal{M}_g^i$ set by VMMIGRATION (Alg. 3 and Alg. 4). We present α, β as different portion of capacity for migration since it is not necessary to migrate all VM's.

Algorithm 1: Pre-Alert Management Procedure

Input: Alert set $\mathcal{A}$.
Output: Flow rerouting set $\mathcal{M}_f^i$; VM migration set $\mathcal{M}_g^i$

$\mathcal{M}_f^i = \varnothing$, $\mathcal{M}_g^i = \varnothing$, ALERT_TOR $= \varnothing$
while $\mathcal{A} \neq \varnothing$ **do**
  Pick ALERT$_x \in \mathcal{A}$;
  switch ALERT$_x$ **do**
    case ALERT$_x$ *from* s_j
      // Compute congestion alert from outer switch s_j
      $\mathcal{F} = \{m_{ij}^k \mid \exists$ flows out from $m_{ij}^k \in h_{ij} \in v_i$ passing through $s_j\}$
      $\mathcal{M}_f^i = \mathcal{M}_f^i \cup$ PRIORITY$(\mathcal{F}, \alpha)$
    case ALERT$_x$ *from local ToR*
      ALERT_TOR $=$ ALERT_TOR $\bigcup \{$ALERT$_x\}$
    case ALERT$_x$ *from* h_{ij}
      // Receive overload alert from host $h_{ij} \in v_i$
      $\mathcal{F} = \{m_{ij}^k \mid m_{ij}^k \in h_{ij}\}$;
      $\mathcal{M}_g^i = \mathcal{M}_g^i \cup$ PRIORITY$(\mathcal{F}, 1)$
  $\mathcal{A} = \mathcal{A} \backslash \{$ALERT$_x\}$
if ALERT_TOR $\neq \varnothing$ **then**
  // Detect congestion alert from v_i's local ToR
  $\mathcal{F} = \{m_{ij}^k \mid m_{ij}^k \in h_{ij}, \forall h_{ij} \in v_i\}$
  $\mathcal{M}_g^i = \mathcal{M}_g^i \cup$ PRIORITY$(\mathcal{F}, \beta)$
VMMIGRATION$(\mathcal{M}_g^i)$; // Apply VM migration

Alg. 2 is a selection subroutine. To relieve the overload and congestion problem, we need to move a portion of VM's from the candidate list with α and β as the satisfaction parameters. The standard of selection is: firstly remove delay-sensitive flows, and then select the VM's with lowest value but largest size. We mimic a dynamic Knapsack algorithm by taking allowed capacity as knapsack size and picking up as many VM's with lowest value as possible. For convenience, we set Mbps as the minimum capacity unit. Specifically, if the priority parameter is one, we only pick one VM with the highest ALERT to ensure load balancing at the end host side.

Algorithm 2: PRIORITY Function

Input: Set of VMs $\mathcal{F} = \{m_{ij}^k\}$, Priority factor ω
Output: Set of selected VMs $\mathcal{VM}$

Eliminate delay-sensitive VMs from $\mathcal{F}$
switch ω **do**
  case α, β
    if ω=α **then** $\mathcal{C} = \alpha \cdot s_j.capacity$
    if ω=β **then** $\mathcal{C} = \beta \cdot$ ToR$_i.capacity$
    $d[0 \ldots \mathcal{C}]$ =LARGE NUMBER
    $\mathcal{V}[0 \ldots \mathcal{C}] = \varnothing$
    foreach $m \in \mathcal{F}$ **do**
      $d[m.capacity] = m.value$
      $\mathcal{V}[m.capacity] = \{m\}$
    for $i = 1$ **to** $|\mathcal{F}|$ **do**
      for $j = \mathcal{F}_i.capacity$ **to** $\mathcal{C}$ **do**
        if $d[j - \mathcal{F}_i.capacity] + \mathcal{F}_i.value < d[j]$ **then**
          $d[j] = d[j - \mathcal{F}_i.capacity] + \mathcal{F}_i.value$
          $\mathcal{V}[j] = \mathcal{V}[j - \mathcal{F}_i.capacity] \bigcup \{\mathcal{F}_i\}$
    $\mathcal{VM} = \mathcal{V}[\mathcal{C}]$
  case 1
    $m =$ VM $\in \mathcal{F}$ with max ALERT, $\mathcal{VM} = \{m\}$

Alg. 3 schedules VM migration process. It generates all possible new location $\mathcal{T}$ for candidate VM set $\mathcal{F}$ and connect them with edges with cost as its weight. We apply Minimal Weighted Matching with time complexity $O(n^3)$ to get the optimal pairs, such as Kuhn-Munkres algorithm (KM) with relaxation [31]. To avoid conflictions, a node can be migrated to another place only when the destination's delegation node accepts the migration request. Otherwise it rejects the request and v_i should recalculate possible migration destinations.

Correspondingly, Alg. 4 deals with receiver conflictions. Once a server accepts a VM migration request by FCFS rule, it will reply an ACK message, otherwise it will reject the request and send a REJECT message.

All in all, Alg. 1 to Alg.4 form the whole management process for a delegated manager under different circumstance.

VI. EVALUATIONS

In this section, we present the evaluation of Sheriff in three parts: first we use the data from real-world data center to verify our pre-alert system; second, we simulate distributed VMs migration algorithms, and compare the performance with

Algorithm 3: VMMIRGRATION

Input: Set of VMs $\mathcal{F} = \{m_{ij}^k\}$
Output: VM migration solution

$\mathcal{T}$ = Available VMs of $v_u \in N_r \bigcup N_w$
while $\mathcal{F} \neq \varnothing$ **do**
  $E_m = \varnothing$
  foreach $m_{ij}^k \in \mathcal{F}$ **do**
    foreach $m_{pq}^r \in \mathcal{T}$ **do**
      E_m=$E_m \bigcup \{(m_{ij}^k, m_{pq}^r, Cost(v_i, v_p))\}$
  Construct $G_m = (\mathcal{F} \bigcup \mathcal{T}, E_m)$
  $\mathcal{L}$ = MinimalWeightedMatching(G_m);
  foreach $m_{ij}^k \in \mathcal{F}$ **do**
    Find $(m_{ij}^k, m_{pq}^r) \in \mathcal{L}$
    if REQUEST(m_{ij}^k, m_{pq}^r) =ACK **then**
      Migrate m_{ij}^k to m_{pq}^r
      $\mathcal{F}$ = $\mathcal{F} \backslash \{m_{ij}^k\}$

Algorithm 4: REQUEST Action

Input: m_{ab}^c, m_{pq}^r.
Output: ACK, REJECT

if $i = p$ **then**
  if $h_{pq}.capacity \geq m_{ab}^c.capacity$ **then**
    $h_{pq}.capacity = h_{pq}.capacity - m_{ab}^c.capacity$;
    reply ACK_{ab}^c;
  else // h_{pq} does not have enough capacity
    reply REJECT_{ab}^c;
else // v_i is not the candidate delegation
  ignore REQUEST message

global optimal performance; finally we will theoretically prove the accuracy and efficiency of Sheriff.

A. Network Traces Training

In this research, we use the data obtained from a local data center service provider ZopleCloud Corp. We collect weekly traffic of a switch, CPU utilization rate of a VM, and the disk I/O speed. Fig. 3-5 exhibits the original data. As shown in Fig. 5, the weekly traffic have its peaks and troughs regularly and by using Box-Jenkins method, we can see that classical time series model ARIMA can be a candidate solution.

We use half of the data for training to calculate the parameters of ARIMA$(1, 1, 1)$ model via MATLAB and use another half as the test set. The training and test results are shown in Fig. 6. Similarly, we find that the model performs well.

However, classical ARIMA method mainly works for linear data, when confronted with nonlinear data, NARNET with 20 hidden layers, outperforms ARIMA. Thus we will use NARNET to further explore the ability of our prediction. Fig. 7 shows its ability in predicting the future, we use 70% of the dataset to train the model and the other 30% to test it. The prediction error is also very small and we can hardly recognize the difference.

Because a dataset may contain both linear data and nonlinear data, we suggest to use this combined model to better predict the future trend. The result is shown in Fig. 8 with a smaller minimum square error.

B. Simulation For VMs Migration

We take Fat-Tree with different numbers of pods (from 8 to 48) as the topology for testing. And we assume that five percent of virtual machines in each pod raise alerts for migration. As mentioned in SECTION V, we solve the VMMIGRATION problem by turning it into a k-median problem and the k-median problem can be efficiently dealt with Alg. 5.

Some initial settings are applied to the simulation. C_r is set to 100 which is a constant. Both δ and η are set to 1. The available bandwidth between core switches and aggregation switches is set to 10 meanwhile the available bandwidth between aggregation switches and ToRs is set to 1 according to the fat tree structure. C_d is set to 1 which means the unit cost per distance in G_d. The $VMcapacity$ is set up to value 20.

Another network model we use to do the simulation is Bcube. Here we change the number of the switches each level of Bcube have from 8 to 48. Most initial settings are the same as they are in the simulation on Fat-Tree.

As shown in Fig. 9 and Fig. 10, we can find out that the standard deviation of the workload percentages of all the servers in the network keeps going down both for Fat-Tree and Bcube, which means that our VM migration algorithm makes sense, and the workload becomes more and more balancing. The algorithm helps increase the utilization rate of the network.

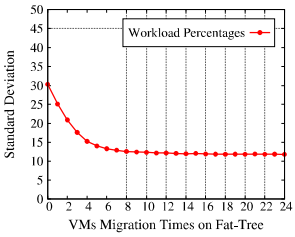

Fig. 9. Sheriff on Fat-Tree

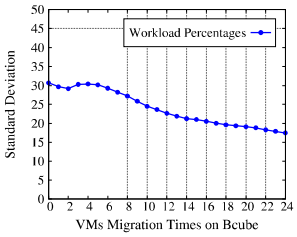

Fig. 10. Sheriff on Bcube

We simulate our Sheriff and a global (centralized) optimal manager. The result of Fat-Tree is shown in Fig. 11 and the result of Bcube is shown in Fig. 13. It shows that our regional distributed Sheriff performs quite well even compared to a centralized optimal manager in these two common models.

We also measure the searching space for matching possible candidate virtual machines and source virtual machines (which to be migrated). As shown in Fig. 12 and Fig. 14, the searching space of regional Sheriff is much smaller than a centralized manager, which takes all hosts into consideration. Thus, Sheriff performs much faster than the centralized manager both for Fat-Tree and Bcube.

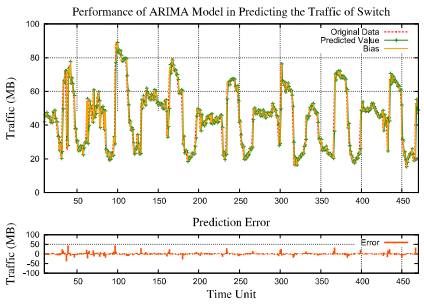

Fig. 6. Training Result for ARIMA

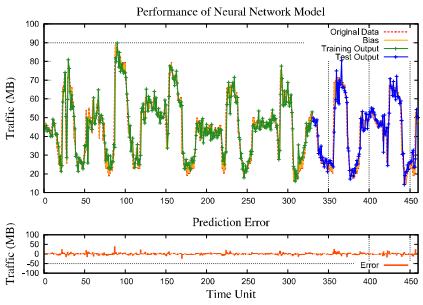

Fig. 7. Training Result for Neural Network

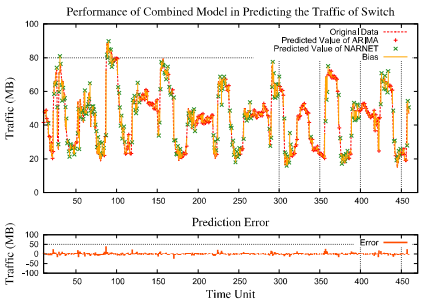

Fig. 8. Training Result for the Combination

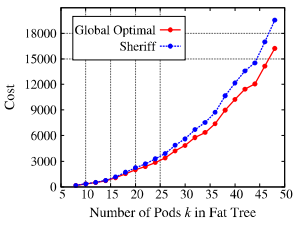

Fig. 11. Output: APP vs OPT

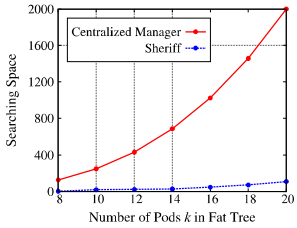

Fig. 12. Search Space Compare

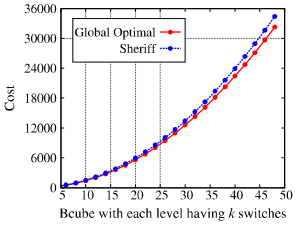

Fig. 13. Output: APP vs OPT

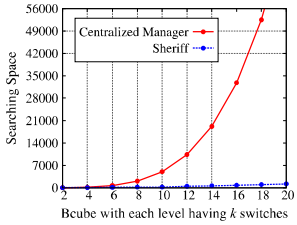

Fig. 14. Search Space Compare

C. Approximation Ratio

In this section, we will think about the approximation ratio of Alg. 3. At first we have a graph T with all ToRs as its vertices. And the edges are the cost of the connections between ToRs both in real wire way. These means T is a graph with multiply edges. By applying Floyd, we can get a new graph T' as a complete one and without multiply edges. According to [32], the precise algorithm has time complexity $O(n^3)$.

Then according to our proof before, it is a k-median problem now. The Local Search [29] algorithm (Alg. 5) performs best in solving k-median problem. It has an approximation ratio $3+\frac{2}{p}$ with time complexity $O(n^p)$.

Algorithm 5: LOCAL SEARCH ALGORITHM

Input: The number of destination ToRs m, edge costs, size of local change p

Output: Optimal solution of m ToRs

$S \longleftarrow$ an arbitrary feasible solution of m ToRs.
while $\exists S' \in B(S)$ *such that* $cost(S') < cost(S)$ *and* S' *can be got by change* p *ToRs in* S **do**
  $S \longleftarrow S'$
return S

Till now, we have find a destination ToRs set and a smallest cost. Therefore, the total approximation ratio of Alg. 3 is $3+\frac{2}{p}$ and the time complexity is $O(n^{3+p})$.

VII. RELATED WORK

The related researches on DCN can be categoried as follows:

Network performance prediction: Rich Wolki [33], [34] proposed Network Weather Service(NWS) to provide dynamic short-time resource performance forecasts in metacomputing environments. Bey-Beghdad's prediction model is based on the control of multiple Local Adaptive Network-based Fuzzy Inference Systems Predictors. [35].

VM migration: Clark et al. [17] presented the design, implementation and evaluation of high-performance OS migration built on top of the Xen virtual machine monitor. Nelson et al. [36] proposed a fast and transparent application migration system. Ye et al. [37] compared the live migration efficiency with different resource reservation approaches and proposed corresponding optimization methods.

Distributed controllers: Shieh et al. [38] proposed Seawall, a network bandwidth allocation scheme based distributed controller which gives a weight for each entity (VM, process, etc.) and guarantees the allocated bandwidth of the entity is proportional to its weight. In [10], [11], [15], researchers investigated the use of multiple independent devolved controllers that manages a part of the network only to solve the scalability problem of networking instead of a single centralized controller.

VIII. CONCLUSION

In this paper we design a fast distributed pre-alert management scheme Sheriff in data center network, which can dominate its local region by one hop wired neighbors. Sheriff will monitor the change of local VM's and predict possible ALERT situation. It then applies FLOWREROUTE or VMMIGRATION for portion of selected VM's to balance the traffic and workload. Since each local manager adjusts network traffic locally, they need to communicate between each other to avoid conflictions. We examine the prediction accuracy of Sheriff by network traces from a local data center service provider; illustrate the management efficiency by simulation on Fat-Tree and Bcube topologies; and prove that VMMIGRATION is a approximation with ratio $3+\frac{2}{p}$, where p is the size of local change in Local Search Algorithm. Both numerical experiments and theoretical analysis validate the efficiency of

Sheriff. It is the first attempt to manage DCN comprehensively by local managers.

REFERENCES

[1] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic Flow Scheduling for Data Center Networks," in *USENIX Symposium on Networked Systems Design and Implementation (USENIX NSDI)*, San Jose, USA, 2010, pp. 281–296.

[2] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowstron, "Better Never than Late: Meeting Deadlines in Datacenter Networks," in *ACM International Conference on the applications, technologies, architectures, and protocols for computer communication (ACM SIGCOMM)*, Toronto, Canada, 2011, pp. 50–61.

[3] A. G. 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 *ACM International Conference on the applications, technologies, architectures, and protocols for computer communication (ACM SIGCOMM)*, Barcelona, Spain, 2009.

[4] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric," in *ACM International Conference on the applications, technologies, architectures, and protocols for computer communication (ACM SIGCOMM)*, Barcelona, Spain, 2009, pp. 39–50.

[5] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "ElasticTree: Saving Energy in Data Center Networks," in *USENIX Symposium on Networked Systems Design and Implementation (USENIX NSDI)*, San Jose, USA, 2010.

[6] T. Benson, A. Akellaand, and D. A. Maltz, "Network Traffic Characteristics of Data Centers in the Wild," in *ACM Internet Measurement Conference (ACM IMC)*, Melbourne, Australia, 2010, pp. 267–280.

[7] D. Halperin, S. Kandula, J. Padhye, P. Bahl, and D. Wetherall, "Augmenting Data Center Networks with Multi-Gigabit Wireless Links," in *ACM International Conference on the applications, technologies, architectures, and protocols for computer communication (ACM SIGCOMM)*, Toronto, Canada, 2011, pp. 38–49.

[8] A. R. Curtis, W. Kim, and P. Yalagandula, "Mahout: Low-Overhead Datacenter Traffic Management using End-Host-Based Elephant Detection," in *IEEE International Conference on Computer Communications (IEEE INFOCOM)*, Shanghai, China, 2011, pp. 1629–1637.

[9] A. Curtis, T. Carpenter, M. Elsheikh, A. Lopez-Ortiz, and S. Keshav, "REWIRE: An Optimization-Based Framework for Unstructured Data Center Network Design," in *IEEE International Conference on Computer Communications (IEEE INFOCOM)*, Orlando, USA, 2012.

[10] A. S.-W. Tam, K. Xi, and H. J. Chao, "Use of Devolved Controllers in Data Center Networks," in *IEEE International Conference on Computer Communications (IEEE INFOCOM)*, Shanghai, China, pp. 596–601.

[11] A.-W. Tam, K. Xi, and H. J. Chao, "Scalability and resilience in data center networks: Dynamic flow reroute as an example," in *IEEE Global Telecommunications Conference (IEEE GLOBECOM)*, 2011, pp. 1–6.

[12] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *USENIX Proceedings of the 2010 internet network management conference on Research on enterprise networking (USENIX INM/WREN)*, 2010, pp. 3–3.

[13] C. A. Macapuna, C. E. Rothenberg, and M. F. Magalhaes, "In-packet bloom filter based data center networking with distributed openflow controllers," in *IEEE Global Telecommunications Conference (IEEE GLOBECOM)*, 2010, pp. 584–588.

[14] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an elastic distributed sdn controller," in *ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (ACM SIGCOMM HotSDN)*, vol. 43, no. 4, 2013, pp. 7–12.

[15] W. Liang, X. Gao, F. Wu, G. Chen, and W. Wei, "Balancing traffic load for devolved controllers in data center networks," in *IEEE Global Communications Conference (GLOBECOM)*, 2014, pp. 2258–2263.

[16] Open Networking Lab, "http://onosproject.org/," Feb.10th, 2015.

[17] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live Migration of Virtual Machines," in *USENIX Symposium on Networked Systems Design and Implementation (USENIX NSDI)*, Boston,USA, 2005, pp. 273–286.

[18] V. Shrivastava, P. Zerfos, K.-W. Lee, H. Jamjoom, Y.-H. Liu, and S. Banerjee, "Application-aware Virtual Machine Migration in Data Centers," in *IEEE International Conference on Computer Communications (IEEE INFOCOM)*, Shanghai, China, 2011, pp. 66–70.

[19] F. Tso, K. Oikonomou, E. Kavvadia, and D. Pezaros, "Scalable traffic aware virtual machine management for cloud data centers," *IEEE International Conference on Distributed Computing Systems (IEEE ICDCS)*, pp. 238–247, 2014.

[20] B. Cao, G. Xiaofeng, C. Guihai, and J. Yaohui, "Nice: Network-aware vm consolidation scheme for energy conservation in data centers," in *IEEE International Conference on Parallel and Distributed Systems (IEEE ICPADS)*, 2014.

[21] M. Alizadeh, B. Atikoglu, A. Kabbani, A. Lakshmikantha, B. Prabhakar, and M.Seaman, "Data Center Transport Mechanisms: Congestion Control Theory and IEEE Standardization," in *Allerton Conference on Communication, Control, and Computing*, Illinois, USA, 2008.

[22] A. Kabbani, M. Alizadeh, M. Yasuda, R. Pan, and B. Prabhakar, "AF-QCN: Approximate Fairness with Quantized Congestion Notification for Multi-tenanted Data Centers," in *IEEE Annual Symposium on High-Performance Interconnects (IEEE HOTI)*, Stanford, USA, 2010.

[23] Y. Hayashi, H. Itsumi, and M. Yamamoto, "Improving Fairness of Quantized Congestion Notification for Data Center Ethernet Networks," in *IEEE International Conference on Distributed Computing Systems (IEEE ICDCS)*, Minneapolis, USA, 2011, pp. 20–25.

[24] N. Laoutaris, M. Sirivianos, X. Yang, and P. Rodriguez, "Inter-Datacenter Bulk Transfers with NetStitcher," in *ACM International Conference on the applications, technologies, architectures, and protocols for computer communication (ACM SIGCOMM)*, Toronto, Canada, 2011, pp. 74–85.

[25] W. Wei, X. Wei, T. Chen, X. Gao, and G. Chen, "Dynamic correlative vm placement for quality-assured cloud service," in *IEEE International Conference on Communications (IEEE ICC)*, 2013, pp. 2573–2577.

[26] S. Kandula, J. Padhye, and P. Bahl, "Flyways To De-Congest Data Center Networks," in *ACM Workshop on Hot Topics in Networks (ACM HotNets)*, New York, USA, 2009, pp. 1–6.

[27] M. Al-Fares, A. Loukissas, and A. Vahdat, "A Scalable, Commodity Data Center Network Architecture," in *ACM International Conference on the applications, technologies, architectures, and protocols for computer communication (ACM SIGCOMM)*, Seattle, USA, 2008, pp. 63–74.

[28] Y. Zhang and N. Ansari, "On Mitigating TCP Incast in Data Center Networks," in *IEEE International Conference on Computer Communications (IEEE INFOCOM)*, Shanghai, China, 2011, pp. 51–55.

[29] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit, "Local search heuristics for k-median and facility location problems," *SIAM Journal on Computing (SIAM SICOMP)*, vol. 33, no. 3, pp. 544–562, 2004.

[30] K. Jain, M. Mahdian, and A. Saberi, "A new greedy approach for facility location problems," in *ACM Symposium on Theory of computing (ACM STOC)*, 2002, pp. 731–740.

[31] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *Journal of ACM*, vol. 19, no. 2, pp. 248–264, 1972.

[32] R. W. Floyd, "Algorithm 97: shortest path," *Communications of the ACM*, vol. 5, no. 6, p. 345.

[33] R. Wolski, "Dynamically forecasting network performance using the Network Weather Service," *Cluster Computing*, pp. 119–132, 1998.

[34] R. Wolski, N. T. Spring, and J. Hayes, "The network weather service: a distributed resource performance forecasting service for metacomputing," *Future Generation Computer Systems*, vol. 15, pp. 757–768, 1999.

[35] K. Bey-Beghdad, F. Benhammadi, Z. Guessoum, and A. Mokhtari, "CPU load prediction using neuro-fuzzy and bayesian inferences," *Neurocomputing*, vol. 74, pp. 1606–1616, 2011.

[36] M. Nelson, B. hong Lim, and G. Hutchins, "Fast Transparent Migration for Virtual Machines," in *USENIX Annual Technical Conference (USENIX ATC)*, Anaheim,USA, 2005, pp. 391–394.

[37] K. Ye, X. Jiang, D. Huang, J. Chen, and B. Wang, "Live Migration of Multiple Virtual Machines with Resource Reservation in Cloud Computing Environments," in *IEEE International Conference on Cloud Computing (IEEE CLOUD)*, Washington DC, USA, 2011, pp. 267–274.

[38] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha, "Sharing the Data Center Network," in *USENIX Symposium on Networked Systems Design and Implementation (USENIX NSDI)*, Boston,USA, 2011.