Using Random Neural Network for Load Balancing in Data Centers

Peixiang Liu
Graduate School of Computer and Information Sciences
Nova Southeastern University
3301 College Avenue, Fort Lauderdale FL 33314, USA
lpei@nova.edu

Abstract—A data center which consists of thousands of connected computer servers can be considered as a shared resource of processing capacity (CPU), memory, and disk space etc. The jobs arriving at the cloud data center are distributed to different servers via different paths. In addition, the internal traffic between servers inside the data center needs to be load balanced to multiple paths between them as well. How to select the underutilized or idle paths for the traffic so as to achieve load balancing and throughput optimality is a big challenge. The Random Neural Network (RNN) is a recurrent neural network in which neurons interact with each other by exchanging excitatory and inhibitory spiking signals. The stochastic excitatory and inhibitory interactions in the network makes the RNN an excellent modeling tool for various interacting entities. It has been applied in a number of applications such as optimization, communication systems, simulation pattern recognition and classification. In this paper, we propose to use Random Neural Network (RNN) to solve the load balancing problem in data centers. RNN is able to achieve adaptive load balancing based on the online measurements of path congestion gathered from the network.

Index Terms—Random Neural Network, Reinforcement Learning, Load Balancing, Data Center

I. INTRODUCTION

In the era of cloud computing, the cloud provides services including full software applications and development platforms, infrastructures such as servers, storage, and virtual desktops to corporate and government organizations, and individual users. A data center which consists of thousands of connected computer servers can be considered as a shared resource of processing capacity (CPU), memory, and disk space etc. Traditionally data centers have been built using hierarchical topologies: edge hosts are organized in racks; each edge host is connected to the Top-of-Rack (ToR) switch; these ToR switches then connect to End-of-Row (EoR) switches, which are interconnected with each other via core switches. Such topologies work well if most of the traffic flows into or out of the data center. However, if most of traffic is internal to the data center, the higher levels of the topology can be a bottleneck due to the uneven distribution of bandwidth.

Recently, other topologies such as FatTree [1] which employs commodity network switches using Clos network, Portland [2], VL2 [3], and BCube [4] were proposed to address the oversubscription and cross section bandwidth problem faced by the legacy three-tier hierarchical topologies. Depending on the traffic pattern, paths can be congested even if the topology offers 1:1 oversubscription ratio. How to select the underutilized or idle paths to carry the network traffic in order to avoid network congestion and improve the data center throughput is still a big challenge.

Different approaches have been used to spread traffic across different paths in data centers. For example, Equal-Cost Multi-Path (ECMP) [5] splits the flows roughly equally across a set of equal length paths based on the hash of some packet header fields that identify a flow. However, for some topologies such as BCube in which paths vary in length, ECMP is not able to access many paths available in the network since it spreads the traffic across the shortest paths only.

Multipath TCP (MPTCP) [6] was used to improve the data center performance and robustness. MPTCP stripes a single TCP connect across multiple network paths. Rather than sending the traffic on one single network path, additional subflows can be opened between the client and the server either using different ports or any additional IP addresses the client or server may have. MPTCP achieves load balancing by moving traffic off more congested paths and placing it on less congested ones based on the congestion control dynamics on those multiple subflows. In other words, the load balancing is implemented at the transport layer using the TCP congestion control mechanism. MPTCP still relies on the routing algorithm of the data center to select the path for each subflow. Further, MPTCP adds more complexity to transport layer which is already burdened by requirements such as low latency and burst tolerance. Data center fabrics, like the internal fabric within large modular switches, behave like a giant switch. The load balancing function in data center should not be bind to the transport layer. For some data center applications such as high performance storage systems, the kernel is bypassed so MPTCP cannot be used at all.

Alizadeh et al proposed a network based distributed congestion aware load balancing mechanism for data centers, which is called CONGA [7]. In CONGA, TCP flows are split into flowlets, which are assigned to different fabric paths based on the estimated real-time congestion on fabric paths. CONGA operates in an overlay network consisting of “tunnels” between the fabric’s leaf switches. When an endpoint (server or VM)
Each neuron in a RNN is represented by testing features: i) A neuron is considered to be in its "firing state" if it has a positive potential; and ii) The signal transmitted between any two neurons are in the form of spikes of a certain rate. Since the RNN was introduced by Gelenbe [8], it motivated a lot of research which generated various RNN extension models. The RNN has been applied in a number of applications such as optimization, image processing, communication systems, simulation pattern recognition and classification [9].

In this paper, we briefly introduce the RNN model and how the RNN with reinforcement learning was successfully used to design the Cognitive Packet Network (CPN) architecture, which offers adaptive QoS driven routing based on on-line measurement and monitoring to address the users’ Quality of Service (QoS) requirements [10]. Then we propose to use RNN with reinforcement learning to select the paths based on the path congestion metric gathered from the network to address the load balancing issue in data centers. The rest of the paper is organized as follows. In section II we present related work on load balancing in data center. In Section III, we briefly describe the mathematical model of RNN and its learning capability in Cognitive Packet Network (CPN). Then we discuss the approach of using RNN with reinforcement learning to solve the load balancing problem in data center in section IV. Finally section V concludes this paper.

II. RELATED WORK

Maguluri et al [11] considered a stochastic mode of jobs arriving at a cloud data center and proposed a load balancing and scheduling algorithm that is throughput-optimal without assuming that job sizes are known or upper-bounded. Paiva et al [12] studied how to assign data items to nodes in a distributed system to optimize one or several of a number of performance criteria such as reducing network congestion, improving load balancing, among others. Grandl et al presented Tetris [13], a multi-resource cluster scheduler that packs tasks to machines based on their requirements of all resource types to avoid resource fragmentation as well as over-allocation of the resources.

In Equal-Cost Multi-Path (ECMP) routing [5], the packets are routed along multiple paths of equal cost. Various methods were proposed for the router to decide which next-hop (path) to used when forwarding a packet. In Hash-threshold, the router selects a key by performing a hash (e.g., CRC16) over the packet header fields that identify a flow. The \( N \) next-hops are assigned unique regions in the key space and the router uses the key to determine which region (next-hop) to use. In Modulo-\( N \) algorithm, the packet header fields which describe the flow are run through a hash function. A final modulo-\( N \) is applied to the output of the hash which directly maps to one of the \( N \) next-hops. Another method is Highest Random Weight (HRW). The router generates a weight using a pseudo-random number generator with packet header fields which describe the flow and the next-hop as seeds. The next-hop which receives the highest weight is selected as the routing option. Basically,
ECMP distributes traffic to multiple paths without considering path quality or congestion.

Raiciu et al discussed how to use Multipath TCP (MPTCP) to improve data center performance and robustness in [6]. Dense interconnection data center topologies provides many parallel paths between pair of hosts. MPTCP establishes multiple subflows on different paths between the same pair of endpoints for a single TCP connection. The intuition of MPTCP is that by exploring multiple paths simultaneously and using the congestion response of subflows on different paths to direct traffic away from congested paths and place it on less congested ones, MPTCP is able to achieve higher network utilization and fairer allocation of capacity of flows.

Alizadeh et al [7] presented the design, implementation, and evaluation of CONGA, a network-based distributed congestion-aware load balancing mechanism for data centers. The majority of the functionality of CONGA resides at the leaf switches. The source leaf switch makes load balancing decisions based on the congestion metrics gathered from the network. CONGA leverages the Virtual eXtensible Local Area Network (VXLAN) encapsulation format used for the overlay network. CONGA leverages the Virtual eXtensible Local Area Network (VXLAN) encapsulation format used for the overlay network. CONGA leverages the Virtual eXtensible Local Area Network (VXLAN) encapsulation format used for the overlay network. CONGA leverages the Virtual eXtensible Local Area Network (VXLAN) encapsulation format used for the overlay network.

The external inputs to neuron $i$ are modeled with Poisson processes of rate $\Lambda(i)$, and $\lambda(i)$. When neuron $i$ is excited, it fires signals at a rate of $r(i)$, which follows the exponential distribution. Therefore, we have the rates at which positive and negative signals are sent out from neuron $i$ to $j$, $w_{ij}^+,$ and $w_{ij}^-$, where $w_{ij}^+ = r(i)p_{ij}^+$, and $w_{ij}^- = r(i)p_{ij}^-$, the total firing rate from the neuron $i$, can then be expressed as follows,

$$r(i) = \frac{\sum_{j=1}^{N} (w_{ij}^+ + w_{ij}^-)}{1 - d(i)}$$

Fig. 1 shows an example random neuron network with three neurons $i$, $j$, and $k$.

![RNN with 3 neurons](image)

Gelenbe [8] showed that the network’s stationary probability distribution can be written as the product of the marginal probabilities of the state of each neuron in the network. A vector of neuron potentials at time $t$, $k(t) = [k_1(t), k_2(t), ..., k_N(t)]$, is used to describe the network state. Let $k = [k_1, k_2, ..., k_N]$ be a particular value of the vector. The stationary probability distribution can be expressed as $p(k) = \lim_{t \rightarrow \infty} \text{Prob}[k(t) = k]$ if it exists. Each neuron $i$ in the $N$-neuron RNN has a state $q_i$, which is the probability that $i$ is excited. The $q_i$, with $1 \leq i \leq N$, satisfies the following nonlinear equations:

$$q_i = \frac{\lambda^+(i)}{r(i) + \lambda^-(i)}$$

with

$$\lambda^+(i) = \sum_{j=1}^{N} q_j w_{ij}^+ + \Lambda_i,$$

$$\lambda^-(i) = \sum_{j=1}^{N} q_j w_{ji}^- + \lambda_i.$$
A. RNN in Cognitive Packet Network (CPN)

RNN with reinforcement learning has been successfully applied in the Cognitive Packet Network (CPN) which provides adaptive QoS driven routing based on online measurements of the network [10], [16]. There are three different types of packets in CPN: Smart Packets (SPs), which explore the network for paths; Dumb Packets (DPs), which carry the payload using the path discovered by SPs; and Acknowledgment Packets (Acks), which bring back the paths discovered by SPs and feed the collected network measurement to RNNs. In CPN, intelligence is introduced to routers in the packet switching network so that the routers are able to learn from their interactions with the network. A RNN is created at each router to make routing decisions. The number of neurons in that RNN is equal to the number of neighbors of the router, with each neighbor represented by a neuron in the RNN. Note that the number of neighbors is actually the number of routing options at the current router. As we discussed earlier, each neuron in a RNN is represented by its signal potential, which is a non-negative integer. The probability that any neuron \( i \) is excited, \( q_i \), can be calculated using equation \( (3) \). When the RNN is queried for routing suggestion, the output link corresponding to the most excited neuron is returned as the routing decision.

B. Learning process of RNN in CPN

The objective of the learning algorithm is to output a distribution of neuron potential with the neuron corresponding to the desired routing option being most excited so that it can be selected. At each CPN router, we keep a threshold value which denotes the average QoS measurement of the paths from that router to the destination. When an Ack arrives, the instant QoS measurement it carries is compared with the threshold to decide whether the RNN should “reward” or “punish” the routing option the Ack brings back, which is accomplished through adjusting the weight matrices \( W^+ = \{ w^+_{ij} \} \) and \( W^- = \{ w^-_{ij} \} \). The routing decisions resulting in higher QoS than the average are rewarded, while the others are punished. This learning process continues until there is no traffic in the network. The new \( q_i \) values are calculated using equation \( (3) \) every time the weights matrices are modified so that the routing decisions for the following SPs are changed adaptively.

A lot of experiments have been conducted to study how CPN works well to satisfy users’ various QoS routing requirements, which include hopcount, delay, and the combination of hopcount and delay [17], [18], [19], [20]. Fig. 2 shows how the average length of paths used by packets changed over time when hopcount was used as the QoS goal. We started the experiments by sending out Constant Bit Rate (CBR) of 200pps from the source to the destination node in a 26-node testbed for two minutes assuming the source node knew nothing about the network. SPs were generated to explore the network and discover the paths which were then brought back by the Acks so DPs could use to carry the payload. The same experiments were conducted for 20 times and the average measurements with 95% confidence intervals were reported.

In Fig. 2, each point represents the average path length used by the packets (SPs or DPs) travelling from the source to the destination in every 10 seconds. We can see that the average length of the paths discovered in the first 10 seconds is about 9.60, and the curve keeps decreasing until it reaches to about 8.34. At the very beginning, the RNNs did not have any knowledge about the network so that the routing decisions they suggested to SPs were not the best options. With the time went on, more SPs were sent out to explore the network and more QoS measurements were fed to the RNNs, which resulted in the weights of RNNs being updated. The more RNNs interact with the network, the more network measurements they learn, and the more reliable were the decisions they made. The decreasing trend of the curves in Fig. 2 clearly shows the learning process of CPN.

In CPN, some SPs do not follow the direction to which the RNNs point, they are routed randomly instead which enables the SPs to explore the network thoroughly rather than stick only to those good paths suggested by RNNs. In our experiments, 5% of the SPs were randomly routed. The length of the shortest paths between the source and destination node in our testbed is 8. Considering the random routing of those 5% smart packets, we can safely draw the conclusion that RNNs have successfully learned the shortest paths when hopcount was used as the QoS goal. Fig. 2 also shows that the average length of the paths used by DPs is almost the same as that of SPs, which is reasonable since DPs use whatever paths the SPs discovers.

IV. RNN FOR LOAD BALANCING IN DATA CENTER

In modern data centers, multiple paths are available for switches to choose from for the same destination. For example, in Fig. 3, a data center with leaf-spine architecture, each leaf switch is fully connected to all four spine switches. In this data center network topology, each leaf switch has 4 switching options to any other leaf switches in the network.

We propose to use Random Neural Network (RNN) to help leaf switches make load balancing decisions. A RNN is created at each leaf switch for every destination leaf switch. The number of neurons in the RNN is equal to the number of uplinks the leaf switch has, with each neuron representing an
metric carried in \( FB\_Metric \) field represents the QoS of paths using outgoing uplink identified by \( FB\_LBTag \). The successive calculated values of \( R \) based on the QoS measurements piggybacked in different packets received by the source leaf switch are denoted by \( R_l, l = 1, 2, \cdots \), which are used to compute a decision threshold \( T_l \),

\[
T_l = \alpha T_{l-1} + (1 - \alpha) R_l
\]  

where \( \alpha \) is some constant \((0 < \alpha < 1)\), which is typically close to 1. \( R_l \) is the most recent value of the reward.

The reinforcement learning algorithm uses \( T_l \) to keep track of historical value of the reward. \( T_l \) can be considered as the average congestion metric (QoS) of paths from the source to the destination leaf switch using any uplinks. Suppose we have made the \( l \)th decision which corresponds to an outgoing uplink (neuron) \( j \) and that the \( l \)th reward calculated for the path congestion received is \( R_l \). We first determine whether \( R_l \) is larger than, or equal to, the threshold \( T_{l-1} \); if this is the case, it means the instant measured QoS for outgoing uplink \( j \) is better or not worse than the threshold QoS. In other words, it means the paths using outgoing uplink \( j \) are less congested or not worse than those paths using the other uplinks to reach destination leaf switch.

Once a neuron in RNN is excited, it sends out “excitation spikes” and “inhibition spikes” to all the other neurons at different firing rates, which are defined as the positive or negative weights: \( w^+_{ij} \) is the rate at which neuron \( i \) sends “excitation spikes” to neuron \( j \) when neuron \( i \) is excited and \( w^-_{ij} \) is the rate at which neuron \( i \) sends “inhibition spikes” to neuron \( j \) when neuron \( i \) is excited. Since the paths for the destination leaf switch via uplink \( j \) are less congested or not worse than paths using other uplinks, we increase very significantly the excitatory weights going into neuron \( j \) and make a small increase of the inhibitory weights leading to other neurons in order to reward it for its success; otherwise, if \( R_l \) is less than \( T_{l-1} \), the measured path congestion is worse than the threshold QoS, we simply increase moderately the excitatory weights leading to all neurons other than \( j \) and increase significantly the inhibitory weight leading to neuron \( j \) in order to punish it for its not being very successful this time.

- If \( T_{l-1} \leq R_l \)
  \[
  w^+_{ij} \leftarrow w^+_{ij} + R_l
  \]
  \[
  w^-_{ik} \leftarrow w^-_{ik} + R_l/(N-1), \text{ for } k \neq j
  \]
- Else
  \[
  w^+_{ik} \leftarrow w^+_{ik} + R_l/(N-1), \text{ for } k \neq j
  \]
  \[
  w^-_{ij} \leftarrow w^-_{ij} + R_l
  \]

Once the weights are updated, the probability that each neuron \( i \) is excited, \( q_i \), is computed using the nonlinear iterations (3) and (4) presented in section III. In the RNN model for load balancing, parameters \( \Lambda(i) \) and \( \lambda(i) \) can be set as constant values. The uplink associated with the most excited neuron is the best option when the path congestion is
network gets the opportunity to be measured therefore more valuable path congestion metrics are fed back to the RNNs located at every leaf switch in data center to make better load balancing decisions.

REFERENCES