Flow Scheduling in Datacenter Network

Lanqing Liu

May 15,2017

Abstract: Nowadays, the amount of data on the network and the population of wireless users increase rapidly. The application providers such as Facebook and Amazon demand the better performance of the datacenter network. The flow scheduling is the core problem of datacenter network. We come about two queuing model in the ingress side of the switch and make the related evaluation. And in order to describe the performance of the system, we introduce system welfare.

Keywords: Datacenter Network, flow scheduling, queue, system welfare.

1 Introduction

1.1 Background

In recent years, the development of the Internet is very rapid. The amount of data on the network surge and the number of wireless network users also explosively grows. As a result, the traditional data processing, storage technology are facing great challenges. Fortunately, cloud computing is produced. Cloud computing is a pay-per-use model that provides available, convenient, and on-demand network access to configurable computing resource sharing pools (resources including networks, servers, storage, applications, services). These resources can be quickly provided, with very little management effort, or little interaction with the service provider. Cloud computing can solve the problem of massive data processing.

With the emergence of cloud service providers (such as Amazon, Microsoft, Google and other companies) and the popularity of datacenter, large enterprises increasingly want to integrate into a datacenter hub and build a high-performance datacenter for computing and storage. So what is a Datacenter network?

Datacenter Network: A datacenter is always composed of some servers and some switches. The flows will be transmitted among the servers through the switches. The fabrics of Datacenter network is various. But the most used fabric is the tree structure showed in Figure 1. Each tier providing network transport to components below it: top-of-rack switches(ToR) connect servers in a rack, aggregation switches connect racks into clusters, core routers connect clusters.

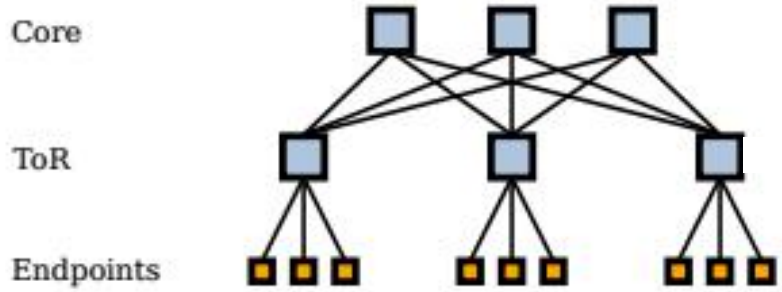

Figure 1

In reality, cloud services still face some problems when processing or storing data, such as the increasing traffic in the network, the complexity of the cloud service access environment, the diversity of user needs, and the heterogeneity of the service itself. They will lead to problems such as network congestion, reduced user experience, reduced computing resources and decreased utilization of network resources. Datacenter resource scheduling is the core of the whole cloud service, for different users of different needs, the datacenter will respond to different, but also as much as possible to improve the computing resources and network resources utilization.

1.2 Our Work

By reading many papers about flow scheduling in datacenter network. We find that there many scheduling and queuing method. Some system consider the problem in terms of flows, others in terms task. Considering the flows of one task have the same source, destination and demands, we make the queue in terms of task. We determine the priority according to the length and the number of the flows of the task. The flows in a task will have the same priority. In order to describe the performance of our system, we make a system welfare function. And the resource of the datacenter will be assigned to every switch according to the system welfare function.

In short, Our work is to find a most efficient queuing method in the ingress side of

the switch and make the resource assignment according to the system welfare. So we two-way priority sorting from the task level. And the observation is to get the most welfare of the whole system including the customs' profit and the provider's profit.

2 Related Work

Our system is build under the inspiration of many previous work of DCN. In order to help the readers better understand our system, the following are the description of some previous work most related to our system.

p-Fabric[1]: p-Fabric is the most used fabric for the slow scheduling in DCN. The p-Fabric assume that all the switches of the DCN is one giant switch. The fabric described in p-Fabric is showed in Figure 2. They will not consider the detailed transmission among the switches. They only consider the how to get the priority of each flow in the ingress and how to inqueue and outqueue for each flow. In our work, when we get the priority of each task, we assume the switches of the DCN is a giant switch just like the p-Fabric.

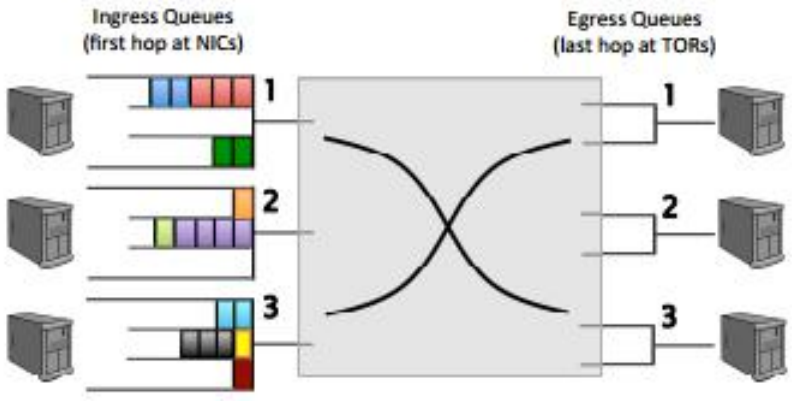

Figure 2

Dynamic pricing[2]: In this system, every edge in the datacenter network has a price. The tasks transmitted through this edge need to pay the price. These prices are

updated every time window. And the price of each edge is determined according to the performance of the system in the last time window. And they also used the system welfare to describe the performance of the system.

Paragon[3]: When one task coming, they will analysis the performance of the task to each server in heterogeneity and interference. And get 2 matrix of performance. This work is completed in the application classify part. After that, they will integrate the heterogeneity performance matrix and the interference performance matrix. And they will choose a best server for each task. This work is completed by server selection part.

3 Models and Evaluation

There are many observation of flow scheduling, such as shorter flow completing time, less packet retransmission and lower cost. At first, in order to find the best queuing method, we aim at cut down the flow completing time. Our queuing method is built based on the p-Fabric. This means that we considers all the switches of the network as one giant switch. We came about two queuing methods. The first is called two-dimensional queuing method. The priority of each task is determined according to two factor. The two factor is the length of all the flows of a task, and the number of the flows of a task. The second is called two-counters queuing method. There will be two counters in the ingress, and each counter is connected to a small switch. At last, in order to get the performance of our system, we introduce a system welfare. And we will make the resource allocation aiming at most system welfare.

3.1 Two-dimensional Queuing Model

On the offline step, there are a lot of tasks coming to the ingress of the switch. Every coming task will report some information of itself. These information include the source, destination, flow length and the flows' number.

To get the priority of a task, we will consider two factors. They are the length of all

the flows and the number of the flows. The length of the flows is the first priority factor. The size of the flows ranges from S_{min} to S_{max}. To get the priority more easily, we divide the range of size into N parts. The 1st part is [S_{min} , S_1], the 2nd part is [S_1, S_2]. As so on, the following parts are [S_2, S_3],...[S_N, S_{max}]. The priority of the task whose size is in the 1st range part is 1. The priority of the task whose size is in the k_{th} part is k. Then we can get the priority according to the first priority factor. The second priority is the number of flows. When the size of the flows is in the same part, we will consider the second priority part. The number of flows is larger, the higher priority it has. This is because the more flows are in the task, the more likely they may interleave with each other. [3]We use to task-id to describe the priority of the task.

On the online step, the task- ids will be updated. When a new task comes, we will use the algorithm came about above to update the task-ids of the remaining tasks in the ingress and the the new task. This queuing model is showed clearly at the Figure 3.

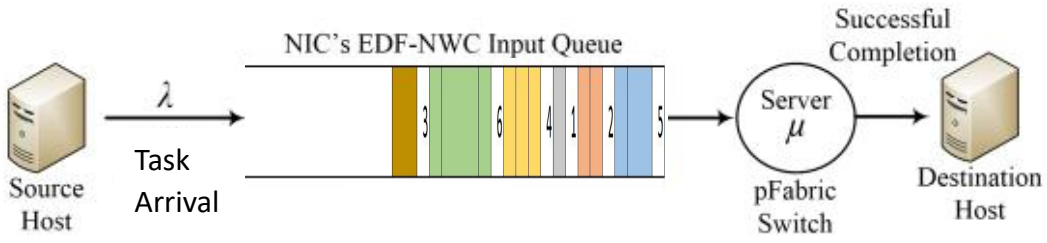

Figure 3

Evaluation: The code to get the task-id according to the size and the length of the flows is as flowing.

```
function [ id ] = priority(task )
%UNTITLED 此处显示有关此函数的摘要
%优先级队列的产生
%   此处显示详细说明
tasknumber=size(task,1);
maxflownumber=size(task,2);
id=ones(1,tasknumber);
prior=zeros(2,tasknumber);
```

```
tasksize=sum(task,2);
tasksize=tasksize';
flownumber=zeros(1,tasknumber);
for i=1:tasknumber
    for j=1:maxflownumber
        if(task(i,j)~=0)
            flownumber(i)=flownumber(i)+1;
        else
            break;
        end
    end
end

for i=1:tasknumber
    if(tasksize(i)<11)
        prior(1,i)=1;
    else if(tasksize(i)<21)
            prior(1,i)=2;
         else if(tasksize(i)<31)
                 prior(1,i)=3;
              else if(tasksize(i)<41)
                      prior(1,i)=4;
                   else
                     prior(1,i)=5;
                   end
              end
        end
    end
end

prior(2,:)=flownumber;
%prior(3,:)=tasksize;
for i=1:tasknumber
    for j=1:tasknumber
        if j==i
            continue;
        else
            if(prior(1,j)>prior(1,i))
               continue;
            else if(prior(1,j)<prior(1,i))
                    id(i)=id(i)+1;
                else
                    if(prior(2,j)<prior(2,i))
                        continue;
                    else if(prior(2,j)>prior(2,i))
```

```
                    id(i)=id(i)+1;
                else
                    if(i<j)
                        continue;
                    else
                        id(i)=id(i)+1;
                    end
                end
            end
        end
    end
end
```

3.2 Two-counters Queuing Model

In this model, there are 2 counters. One counter is connected one stack. Counter1 produce the priority of 1,3,5... and will drag the tasks whose priority is 1,3,5...to the stack1. Counter2 produce the priority of 2,4,6... and will drag the tasks whose priority is 2,4,6...to the stack2. The process to get the priority is online.

At first, we will assign the first coming task of the 1st priority and the second coming task of the 2nd priority. And drag them respectively to stack1 and stack2. For the following coming task,we will drag this task to which stack has the smaller size of flows. And the drag will happen when each task comes. And the tasks of the 2 stacks will be transmitted simultaneously.

The model is showed as Figure 4.

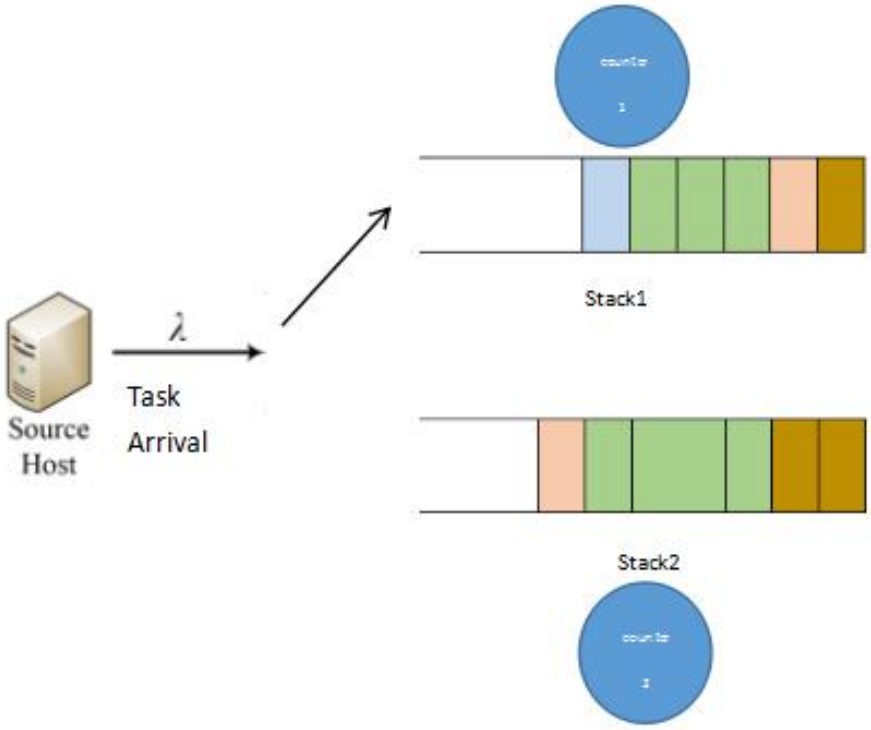

Figure 4

Evaluation: the flow completed time is the time of which stack complete its tasks later. The evaluation result is as figure 5 and figure 6.

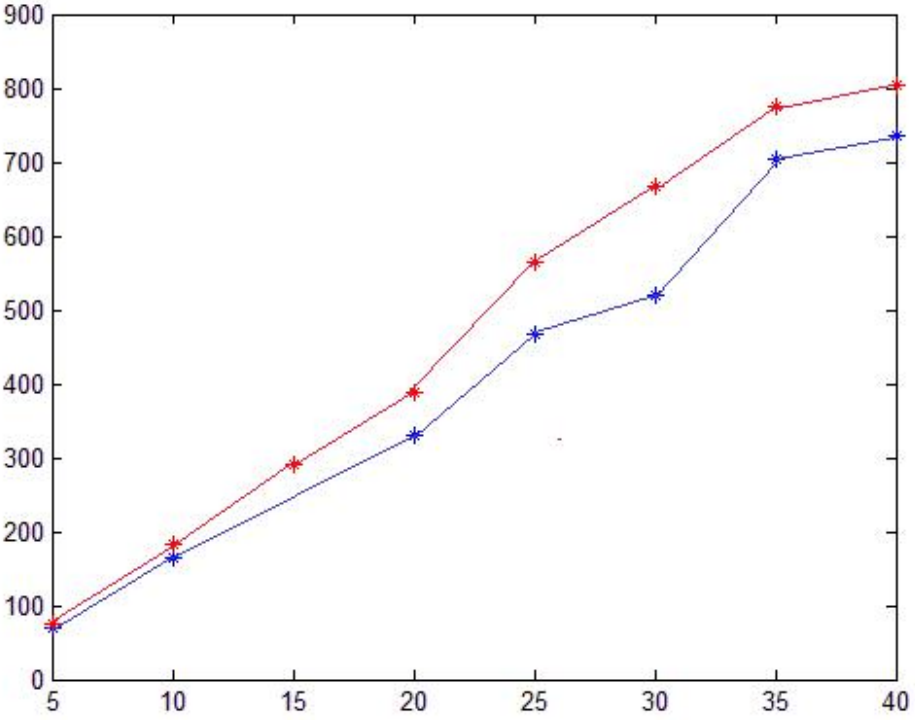

Figure 5

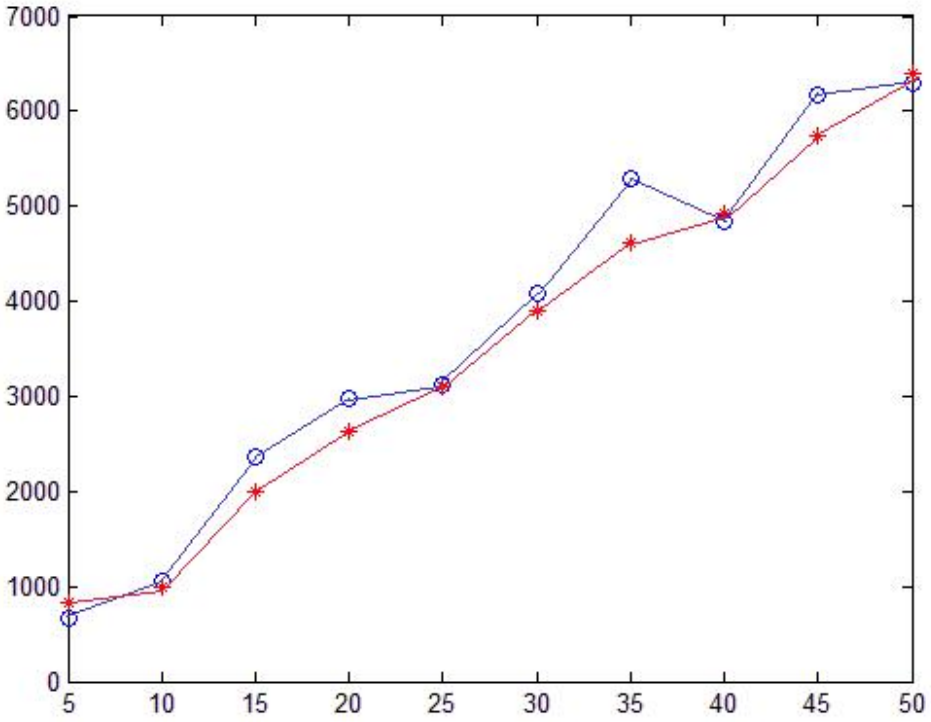

Figure 6

We will see from the figures above that the flow completed time is positive correlation to the size.Figure 5 is the size of flows which is linear. Figure 6 is the size of flows which follows Gaussian distribution.

In the practical scenario, the complexity of the second model is larger. And the first model consider the number of the flows. This information is not much difficult to get, but is important. So we are tend to use the 1st queuing model in the future work.

3.3 System Welfare

To describe the performance of the system, we introduce the system welfare. This is a function to of the description of the performance in the terms of the economic. To get the system welfare function, we need to know the value of each task. And we also need to know the cost of the system. The cost of the system is related to the flow completed time of each tasks. The longer the task waits in the queue, the more cost the system will have. The function of the system welfare is described as following.

$$\sum_{i} \sum_{r \in R_i} \sum_{t \in [t_i^1, t_i^2]} v_i \cdot X_{irt} - C(X).$$

V_i is the value of task i. X_{irt} is the capacity of the request i in link r during time t. C(X) is a function related to the waiting time of each task.

4 Future Work

In the future, we will how to get the C(X). This function is related to the flow completed time of each task. After we get the C(X), instead of queuing according to the flow completed time, we will use the system welfare to get the priority of each task.

Reference

[1] pFabric: Minimal Near-Optimal Datacenter Transport

[2] Dynamic Pricing and Traffic Engineering for Timely Inter-Datacenter Transfers

[3] Paragon- QoS-Aware Scheduling for Heterogeneous Datacenters

[4] Towards Practical and Near-Optimal Coflow Scheduling for Data Center Networks

[5] Fastpass- A Centralized “Zero-Queue” Datacenter Network

[6] Data Center TCP (DCTCP)

[7] Efficient coflow scheduling with varys

[8] Finishing Flows Quickly with Preemptive Scheduling

[9] Network-Aware Scheduling for Data-Parallel Jobs- Plan When You Can