Building Application-Aware Networks Infrastructure with Software Defined Networks

ABSTRACT

1. INTRODUCTION

2. MOTIVATING EXAMPLES

In this section, we charaterize the workloads of two widely deployed service in today's data centers, highlighting the features influencing the design decisions of A^2Net.

2.1 Shuffle

Hadoop [4] is an open source implementation of MapReduce [3]. It's widely deployed in many data centers handling many throughput preferred tasks, e.g. update/backup the online/offline database, refresh the web index, etc.. The workflow of MapReduce/Hadoop jobs is divided into two phases, Map, generating the key-value pairs by processing the job input via user-defined *Map()* function, and Reduce, which takes the output of Map phase as input and yields the final output by executing the user-defined *Reduce()*. Both of these two phases are completed by the distributed tasks, *Mappers* and *Reducers*. The workflow of Shuffle is shown in Figure 1.

The output of Mappers are transfered through networks to Reducers, which is referred as *Shuffle* in Hadoop context. The key space of the Mappers' output is split into multiple subspaces. By default, the number of subspaces is equal to the number of *Reducers*. For a certain key-value pair, which reducer it will be sent to is determined by another function *Partitioner()*. The keys of the key-value pairs generated by Mappers are passed as the input parameter of *Partitioner()* yielding destination of the key-value pair.

2.1.1 Load Skew in Shuffle

We have an insight to the default implementation of *Partitioner()* in Hadoop. The receiver of the pair is calculated by hashing the key of the pair:

```
public int getPartition(K2 key, V2 value,
                        int numPartitions) {
return (key.hashCode()&Integer.MAX_VALUE)
%numPartitions;
}
```

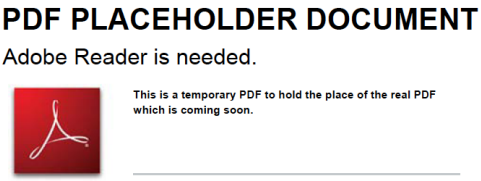

Figure 1:

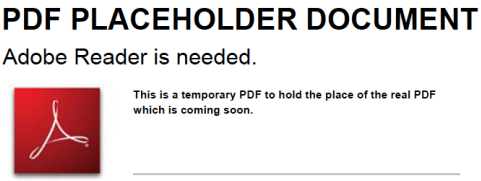

Figure 2:

The random hash function makes the Shuffle stage be vulnerable to load skew. As shown in Figure 2, a Mapper may have more data to be sent to certain Reducer than others, while fairness-striving TCP protocol allocates the same bandwidth to all flows sent by this Mapper. So the overloaded flow may last longer than others, unnecessarily prolonging the completion time of the task, and in turn, the task becomes the *Stragglers* or *Outliers* [2] [6].

To elaborate the problem of uneven load in Shuffle, we present a simulation on a 128-server FatTree topology with a typical oversubscription factor setup, [5]

2.2 Partition-Aggregator

Partition-Aggregator [1] workloads are usually generated by the data center applications handling user requests, e.g. emails, search engine and social networks

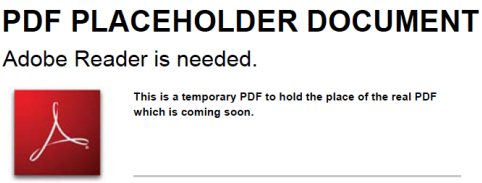

Figure 3:

content composition, which are usually built with soft real-time style for interactive. As shown in Figure 3, requests from users are split into multiple *sub-requests* by an aggregator in the root layer and delivered to the distributed worker processes in the system via aggregators on different layers.

To meet the all-up SLA [1], aggregators usually assign deadlines to the worker nodes in lower layers. When deadline arrives, the aggregator will directly return the collected results back to the upper layer instead of waiting for all nodes to complete. Those discarded results in lagged behind nodes can ultimately lower the quality of the responses. As we explained in Introduction section, network transfer occupies a considerable part of total workloads of data center applications. The worker deadlines mean that network flows carring requests and responses have deadlines. Only the network flows finished within deadlines can contribute to the throughput of application, on another side, the slow and ultimately discarded flows can waste network bandwidth.

2.2.1 *Causes of Missing Deadlines*

However, the commonly used TCP protocol strives for maximizing network throughput while keeping fairness, being unaware of the deadlines of flows. The lack of prioritization among flows causes that *urgent* flows wait behind latency-insensitive flows just for fairness, making them miss the deadlines. To examine this issue, we reproduce the workload consisting of both throughput-preferred but latency-insensitive flows and latency flows....

3. TECHNICAL DESIGN

In this section, we will introduce the design principles of our system. We begin by discuss the stack of our system. Then examine the problems in current OpenFlow design which make it unsuitable for high performance networks and propose our solutions. Following that, we describe the the methods to maximize the throughput of latency-sensitve applications and reduce the straggler large flows

3.1 system stack

- API
- Flow Controller

3.1.1 *challenges in openflow*

1. involves controllers too often, those latency-sensitive flows will be suffered from it

3.1.2 *solution*

application table (deadline): static(intialized when warming up of switches)
dynamic table : job input information: priority, and flows input size

- for latency sensitive services applications, locating it in static one, operators should setup it manually, openflow controller sync the application information with the routers when warming up)
- for large ones, when invoking flows, the flow size is known by the controller through API, assigning rate according to input

4. REFERENCES

[1] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 conference*, SIGCOMM '10, pages 63–74, New York, NY, USA, 2010. ACM.

[2] Ganesh Ananthanarayanan, Srikanth Kandula, Albert Greenberg, Ion Stoica, Yi Lu, Bikas Saha, and Edward Harris. Reining in the outliers in map-reduce clusters using mantri. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–16, Berkeley, CA, USA, 2010. USENIX Association.

[3] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[4] Apache Foundation. Hadoop project. http://hadoop.apache.org/.

[5] Urs Hoelzle and Luiz Andre Barroso. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 1st edition, 2009.

[6] Matei Zaharia, Andy Konwinski, Anthony D. Joseph, Randy Katz, and Ion Stoica. Improving mapreduce performance in heterogeneous environments. In *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, OSDI'08, pages 29–42, Berkeley, CA, USA, 2008. USENIX Association.