



## “大学生创新训练计划”项目结题报告

中文题目 基于SDN的云中大数据网络优化的研究

英 文 Research of bigdata network optimization

题 目 based on SDN in cloud computing

学 院 信息学院

指导教师 张文逸

小组成员:

姓 名 孟思尧 学 号 PB12210076

姓 名 王笑霄 学 号 PB12210100

姓 名 张广帅 学 号 PB12210080

2016 年 5 月 10 日

## 摘 要

关键词：云计算，大数据，SDN，HADOOP，网络优化

## **ABSTRACT**

**KEY WORDS:** Cloud computing, Bigdata, SDN, HADOOP, network optimization

# Contents

<b>1 项目背景</b>	<b>5</b>
1.1 项目的意义和关键问题 . . . . .	5
1.2 研究方向 . . . . .	5
<b>2 项目平台的选择</b>	<b>5</b>
2.1 Hadoop . . . . .	5
2.1.1 为什么选择Hadoop . . . . .	5
2.1.2 Mapreduce计算框架 . . . . .	6
2.1.3 Shuffle阶段 . . . . .	7
2.2 SDN . . . . .	8
2.2.1 Floodlight . . . . .	8
2.2.2 OVS . . . . .	9
<b>3 基于Coflow的大数据网络优化</b>	<b>10</b>
3.1 coflow相关知识介绍 . . . . .	10
3.2 coflow调度算法设计 . . . . .	12
<b>4 实验平台搭建</b>	<b>12</b>
4.1 实验环境 . . . . .	12
4.2 Hadoop平台搭建 . . . . .	12
4.3 SDN开发环境搭建 . . . . .	13
4.4 网络拓扑 . . . . .	13
<b>5 系统设计与开发</b>	<b>13</b>
5.1 系统整体架构设计 . . . . .	13
5.2 Mapreduce Fetcher . . . . .	13
5.3 Python controller . . . . .	15
5.3.1 HTTPserver . . . . .	15
5.3.2 路由算法模块 . . . . .	15
5.4 Floodlight REST API . . . . .	15
5.4.1 Packet-in模块 . . . . .	15
5.4.2 REST API . . . . .	15
<b>6 项目的成果</b>	<b>16</b>
<b>7 项目总结</b>	<b>16</b>
<b>8 参考文献</b>	<b>16</b>

# 1 项目背景

## 1.1 项目的意义和关键问题

大数据应用正深刻改变人们的生活，已经成为当前学术界、工业界的关注焦点。大数据应用开发云平台纷纷推出大数据计算框架。如Amazon的EC2，微软的Azure。虽然云平台上的大数据框架方便了第三方开发者和用户，但其性能上还存在较多问题，尤其是网络的问题。在2014 Daytona GraySort排序赛上，基于Spark的系统它使用了207个EC2节点在23分钟内排序了100TB的数据而夺冠。而上届冠军Hadoop用了2100台Yahoo内置的机器，花了72分钟，这性能提升不言而喻！更重要的是这次比赛证实Shuffle真正的瓶颈在于网络。传统大数据网络架构是数据库服务器将应用服务器请求的数据通过网络传输到应用服务器上，处理后，将改写的数据再写回数据库服务器，而且处理过程中会出现大量的中间结果也需要网络传输。当数据量较大时，很可能堵塞网络。因而对于进一步提升大数据处理性能，研究云平台上大数据网络问题显得尤为重要。

## 1.2 研究方向

**软件定义网络（SDN）** 软件定义网络（SDN）是当前和未来网络研究的一个重要方向，通过将网络的数据转发层和逻辑控制层分离，提高了网络的灵活性和可编程性。可以在控制层面设计应用感知的控制策略，如网络接入、数据转发路由、流量工程等，从而提高应用的性能。因此，我们拟通过SDN来优化云平台上大数据框架的网络性能。

**coflow调度** 考虑到当前数据中心用并行计算的框架处理大规模的数据，充分考虑数据流的相关性（coflow），设计调度算法

# 2 项目平台的选择

## 2.1 Hadoop

### 2.1.1 为什么选择Hadoop

数据并行计算框架很多，如：Mapreduce，Spark，Google Dataflow等等，我们为什么选择Hadoop？首先，认识一下Hadoop，Hadoop是一种计算集群，它将数据分析的工作分配到多个集群节点上，从而并行处理数据。经过调研，Hadoop用于大数据处理，主要有如下几个优势：

**1、灵活的可扩展性** 要充分利用大数据最大的优势就需要实时或接近实时地对海量数据进行分析处理。大数据分析面临的一个巨大的难题是数据量的不断增加。而Hadoop集群的并行处理能力能明显提高分析速度，但随着要分析的数据量的增加，集群的处理能力会受到影响。Hadoop通过增加集群节点，可以线性地扩展集

群以处理更大的数据集。另外，在集群负载下降时，也可以减少节点，以高效使用计算资源。所以Hadoop的弹性很好

- 2、**Hadoop的设计适合大数据处理** 大数据一般都是分布广泛的并且是非结构化的。而Hadoop非常适合处理这类数据，因为Hadoop的mapreduce的计算框架工作原理是将数据拆分成片，并将每个“分片”分配到特定的集群节点上进行分析。数据不必均匀分布，因为每个数据分片都是在独立的集群节点上进行单独处理。
- 3、**Hadoop成本低** Hadoop的软件是开源的，同时，Hadoop支持商用硬件，可以运行在一般商业机器构成的大型集群上，如：亚马逊弹性计算云（Amazon EC2）等，不必花费重金购买服务器级别的硬件设备。所以我们可以低成本的实现计算能力强大的Hadoop集群，性价比很高。
- 4、**容错能力强** 在Hadoop集群进行大数据处理分析过程中，当一个数据分片发送到某个节点进行分析时，该数据在集群其它节点上会存有副本。通过备份的方式，即使一个节点发生故障，数据可以快速的恢复，继续进行分析处理。故障检测和自动恢复是Hadoop最初的设计目标，所以Hadoop很健壮。

由于Hadoop具有上述优势，使得Hadoop在学术界和工业界都大受欢迎。

### 2.1.2 Mapreduce计算框架

Mapreduce是什么？怎样完成大数据处理？

Hadoop的设计思路源于Google的GFS和MapReduce。它是一个开源软件框架，通过在集群计算机中使用MapReduce这个简单的编程模型，可编写和运行分布式应用程序处理大规模数据。

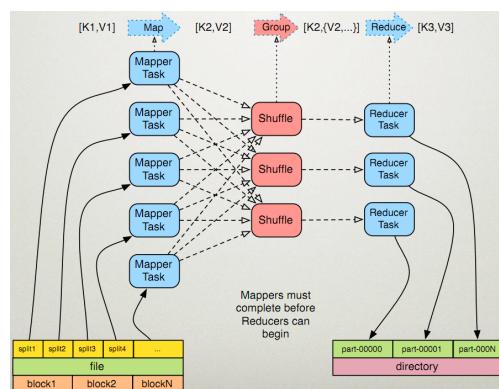


Figure 1: Mapreduce计算框架

MapReduce架构的大数据处理的操作流程如下：

1. 首先调用MapReduce库的输入流模块，将输入文件分成M个数据片度，每个数据片段的大小可以通过可选的参数来控制。然后用户程序在机群中创建大量

的程序副本。程序副本中包含一个特殊的master程序。其它的程序都是worker程序，由master分配任务。有M个Map任务和R个Reduce任务将被分配，master将一个Map任务或Reduce任务分配给一个空闲的worker。

2. 被分配了map任务的worker程序读取相关的输入数据片段，从输入的数据片段中解析出key/value pair，然后把key/value pair传递给用户自定义的Map函数，由Map函数生成并输出的中间key/value pair，并缓存在内存中。
3. 缓存中的key/value pair通过调用Partition模块分成R个区域，之后周期性的写入到本地磁盘上。缓存的key/value pair在本地磁盘上的存储位置将被传给master，由master负责把这些存储位置再传送给Reduce worker。
4. 当Reduce worker程序接收到master程序发来的数据存储位置信息后，使用RPC从Map worker所在主机的磁盘上读取这些缓存数据。当Reduce worker读取了所有的中间数据后，通过对key进行排序后使得具有相同key值的数据聚合在一起。由于许多不同的key值会映射到相同的Reduce任务上，因此必须进行排序。如果中间数据太大无法在内存中完成排序，那么就要在外部进行排序。
5. Reduce worker程序遍历排序后的中间数据，对于每一个唯一的中间key值，Reduce worker程序将这个key值和它相关的中间value值的集合传递给用户自定义的Reduce函数。Reduce函数的输出被追加到所属分区的输出文件。
6. 当所有的Map和Reduce任务都完成之后，master唤醒用户程序。在这个时候，在用户程序里的对MapReduce调用才返回。在成功完成任务之后，MapReduce的输出存放在R个输出文件中（对应每个Reduce任务产生一个输出文件，文件名由用户指定）。一般情况下，用户不需要将这R个输出文件合并成一个文件—他们经常把这些文件作为另外一个MapReduce的输入，或者在另外一个可以处理多个分割文件的分布式应用中使用。

### 2.1.3 Shuffle阶段

整体的Shuffle过程包含以下几个部分：Map端Shuffle、Sort阶段、Reduce端Shuffle。即是说：Shuffle过程横跨map和reduce两端，中间包含sort阶段。

在Hadoop集群中，大部分map task与reduce task的执行是在不同的节点上，Reducer通过Http方式得到map阶段输出文件的分区，所以很多情况下Reduce执行时需要跨节点去拉取其它节点上的map task结果。在Mapreduce的Shuffle阶段，如果集群正在运行的job有很多，而且需要跨节点拉取数据时，会产生大量的数据在网络中传输，对集群内部的网络资源消耗会很严重，尽可能地减少对带宽的不必要消耗，并且保证完整地从map task端拉取数据到reduce端。

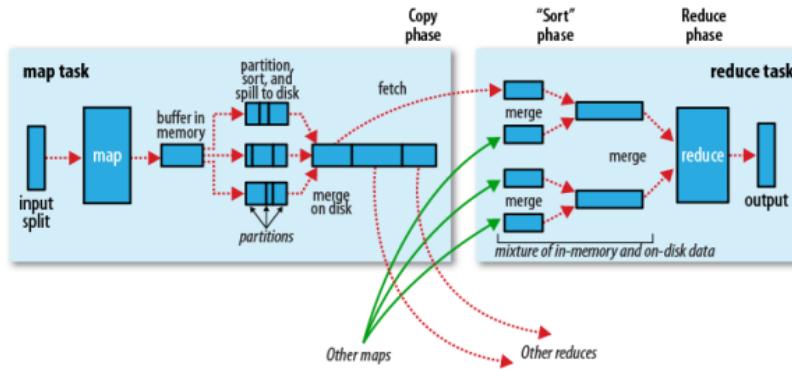


Figure 2: shuffle阶段

## 2.2 SDN

### 2.2.1 Floodlight

SDN（软件定义网络）利用OpenFlow协议，把路由器的控制平面（control plane）从数据平面（data plane）中分离出来，以软件方式实现。这个架构可以让网络管理员，在不改动硬件设备的前提下，通过集中式的控制器（Controller）以标准化的接口对各种网络设备进行管理和配置，那么这将为网络资源的设计、管理和使用提供更多的可能性，为控制网络流量提供了新的方法。

控制器作为SDN网络中的重要组成部分，我们选择Floodlight作为SDN控制器。因为Floodlight是目前主流的SDN控制器之一，它的稳定性、易用性已经得到SDN专业人士一致好评，由于其完全开源，这让SDN网络世界变得更加有活力。能集中地灵活控制SDN网络，为核心网络及应用创新提供了良好的扩展平台。

Floodlight控制器是一个企业级的，使用Java开发的OpenFlow协议的控制器。OpenFlow是一个由Open Networking Foundation (ONF)管理的开放标准。它定义了一种协议让远程控制器通过路由器可以修改网络设备的行为，使用定义良好的转发指令集。Floodlight被设计为同支持OpenFlow标准的设备（交换机，路由器，虚拟交换机）一起工作。

Floodlight controller模块为多数应用实现了一些通用的功能：

1. 发现网络状态和事件（拓扑结构，设备，流量）
2. 能够控制网络交换机（network switches）通信
3. 管理floodlight模块，共享存储，线程，测试等资源
4. 提供一个web界面和debug服务器（Python）

Floodlight控制器工作过程如下：

1. 控制器与交换机建立ofchannel通道，控制器通过ofchannel控制和管理交换机。
2. 当交换机收到一个数据包且流表中没有匹配条目，交换机会将数据包封装在packet-in消息发送给控制器，此时数据包会缓存在交换机中等待处理。

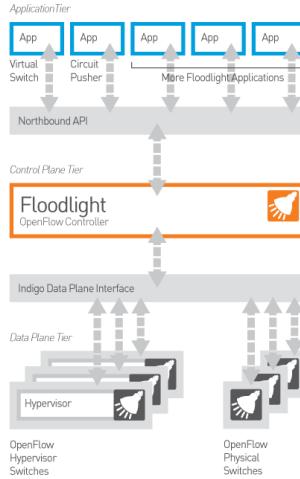


Figure 3: Floodlight控制器架构图

3. 控制器收到packet-in消息后，可以发送flow-mod消息向交换机写一个流表项，并且将flow-mod消息中buffer-id字段设置为packet-in消息中的buffer-id值。从而控制器向交换机写入了一条与数据包相关的流表项，并且指定该数据包按照该流表项的action列表处理。但是并不是所有的数据包都需要向交换机中添加一条流表项来匹配处理，网络中还存在多种数据包，它出现的数量很少（如ARP,IGMP等），以至于没有必要通过流表项来指定这一类数据包的处理法，此时控制器可以使用packet-out消息，告诉交换机某一个数据包如何处理。

### 2.2.2 OVS

Open vSwitch是一个由Nicira Networks主导的开源项目，通过运行在虚拟化平台上的虚拟交换机，为本台物理机上的VM提供二层网络接入，跟云中的其它物理交换机一样工作在Layer 2层。Open vSwitch充分考虑了在不同虚拟化平台间的移植性，采用平台无关的C语言开发。并且遵循Apache2.0许可；我们有传统的物理交换机，为什么还要开发Open vSwitch呢？

在传统数据中心中，网络管理员习惯了每台物理机的网络接入均可见并且可配置。通过在交换机某端口的策略配置，可以很好控制指定物理机的网络接入，访问策略，网络隔离，流量监控，数据包分析，Qos配置，流量优化等。但是在云平台上，如果没有网络虚拟化技术的支持，管理员只能看到被桥接的物理网卡，其上川流不息地跑着n台VM的数据包。仅凭物理交换机支持，管理员无法区分这些包属于哪个OS 哪个用户。而且难以满足以下需求：

**网络隔离** 物理网络管理员早已习惯了把不同的用户组放在不同的VLAN中，例如研发部门、销售部门、财务部门，做到二层网络隔离。Open vSwitch通过在host上虚拟出一个软件交换机，等于在物理交换机上级联了一台新的交换机，所有VM通过级联交换机接入，让管理员能够像配置物理交换机一样把同一台host上的众多VM分配到不同VLAN中去；

**QoS配置** 在共享同一个物理网卡的众多VM中，我们期望给每台VM配置不同的速度和带宽，以保证核心业务VM的网络性能。通过在Open vSwitch端口上，给各个VM配置QoS，可以实现物理交换机的traffic queuing和traffic shaping功能。

**流量监控** 物理交换机通过xxFlow技术对数据包采样，记录关键域，发往Analyzer处理。

进而实现包括网络监控、应用软件监控、用户监控、网络规划、安全分析、会计和结算、以及网络流量数据库分析和挖掘在内的各项操作。例如，NetFlow流量统计可以采集的数据非常丰富，包括：数据流时戳、源IP地址和目的IP地址、源端口号和目的端口号、输入接口号和输出接口号、下一跳IP地址、信息流中的总字节数、信息流中的数据包数量、信息流中的第一个和最后一个数据包时戳、源AS和目的AS，及前置掩码序号等。

xxFlow因其方便、快捷、动态、高效的特点，为越来越多的网管人员所接受，成为互联网安全管理的重要手段，特别是在较大网络的管理中，更能体现出其独特优势。有了Open vSwitch，作为网管的你，可以把xxFlow的强大淋漓尽致地应用在VM上！

**数据包分析** 物理交换机的一大卖点，当对某一端口的数据包感兴趣时（for troubleshooting, etc），可以配置各种span（SPAN, RSPAN, ERSPAN），把该端口的数据包复制转发到指定端口，通过抓包工具进行分析。Open vSwitch官网列出了对SPAN, RSPAN, and GRE-tunneled mirrors 的支持。

Open vSwitch 引入了以下模块，很好地满足以上需求：

1. ovs-openflowd — OpenFlow交换机；
2. ovs-controller — OpenFlow控制器；
3. ovs-ofctl — Open Flow 的命令行配置接口；
4. ovs-pki — 创建和管理公钥框架；
5. tcpdump的补丁— 解析OpenFlow的消息；

当Open vSwitch的一个接口收到数据包后，会按照上述流程图处理：收到数据包后，会交给datapath内核模块处理，当匹配到对应的datapath会直接输出，如果没有匹配到，会交给用户态的ovs-vswitchd查询flow，用户态处理后，会把处理完的数据包输出到正确的端口，并且设置新的datapath规则，后续数据包可以通过新的datapath规则实现快速转发。

### 3 基于Coflow的大数据网络优化

#### 3.1 coflow相关知识介绍

另外考虑到当前数据中心用并行计算的框架处理大规模的数据，有人提出了coflow的概念。coflow是一组相关的并行数据流的集合，coflow的完成时间由最后

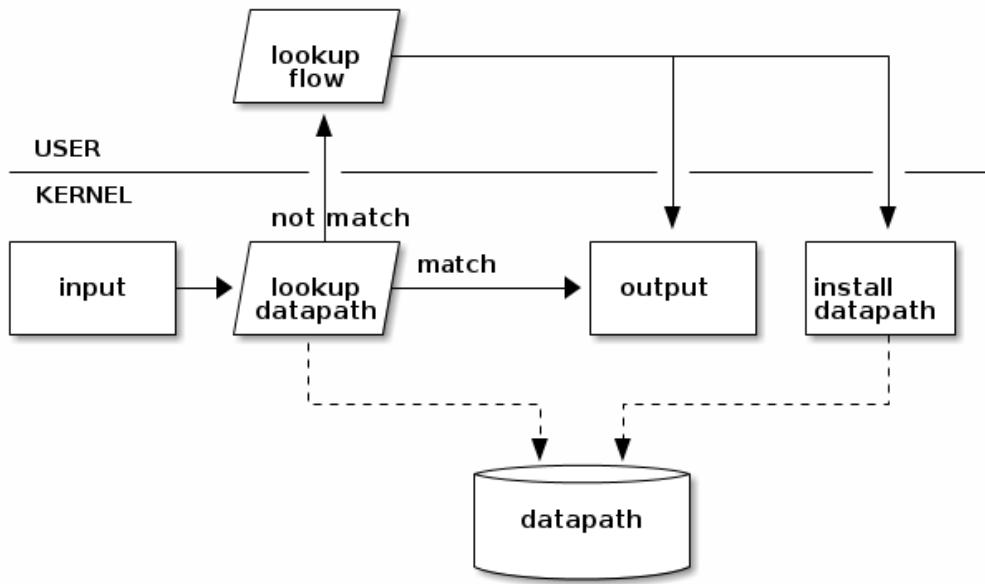


Figure 4: OVS 工作流图

一条数据流的完成时间决定。

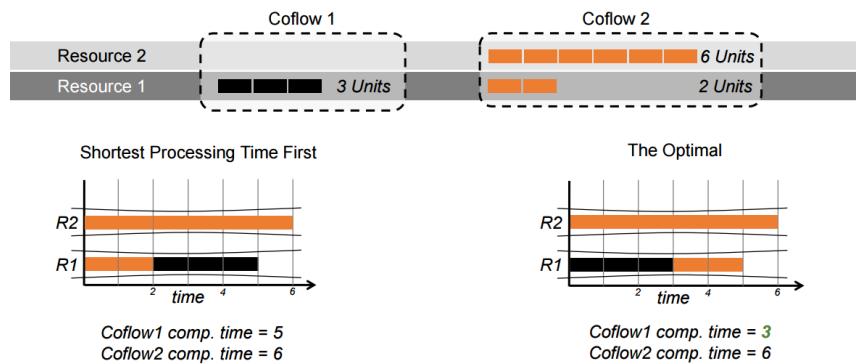


Figure 5: coflow完成时间的比较

此处可详细说明一下这个例子

有人设计了varys来提高大数据处理过程中网络通信的性能，他们以最小化完成时间和满足deadline为目标实现了FIFO、SCF和SEBF等启发式算法。采用varys 仿真器coflowsim完成Facebook真实数据的调度测试。

coflow研究现状中存在的问题:

1. 当前coflowsim中的调度策略都假设所有的flow都是同时出现，不太符合实际情况
2. 当前的各种启发式算法都是在coflowsim中实现的，还没有用varys调度真实的Facebook数据流，

3. 当前coflow的两层架构：inter-transfer负责多个transfer间的资源调度，intra-transfer负责一个transfer内的调度。inter-transfer策略主要是加权共享，并没有考虑多个flow间的dependency。

### 3.2 coflow调度算法设计

介绍我们实现的coflow调度算法

考虑路由

路由算法，最短路径路由，负载均衡/ECMP/多路径路由

varys是交换结构，考虑无阻塞的情况。

用matlab仿真，跑一些简单的结果，分析性能

## 4 实验平台搭建

### 4.1 实验环境



Figure 6: 真实的实验环境

服务器的性能：机器的型号，内存大小，网卡

### 4.2 Hadoop平台搭建

搭建了hadoop平台，完成多节点的配置，进行大数据的处理

Table 1: 系统软硬件及协议配置

名称	配置
服务器配置	CPU: 16 Intel(R) Xeon(R) CPU E5620 @ 2.40GHz MEM: 64G DISK: 1000G 网卡: 1000 Mbps
软件及协议	Linux ubuntu 14.04 Floodlight V1.0 OVS v2.02 OpenFlow1.3 Hadoop2.7 Python2.7

### 4.3 SDN开发环境搭建

搭建基于floodlight的SDN开发环境，进行网络的控制；

将多个虚拟交换机ovs挂载到了floodlight控制器上

利用ovs完成hadoop平台上的数据导流工作

### 4.4 网络拓扑

## 5 系统设计与开发

### 5.1 系统整体架构设计

### 5.2 Mapreduce Fetcher

1孟

Map任务结束之后，主要进行Copy、Sort、Reduce三个步骤；其中Copy阶段，就是从执行各个Map任务的节点获取map的输出文件。这是由ReduceTask.ReduceCopier类来负责。ReduceCopier对象负责将Map函数的输出拷贝至Reduce所在机器。在这个过程中，Reduce进程会启动一些数据复制线程(Fetcher)，通过HTTP GET方法请求由HTTP Server管理的Shuffle 数据块，这些数据块是先前成功完成的Map Task 的输出文件。这些等待Shuffle 的数据块可能在内存中也有可能在HDFS 中，取决于服务器当前的内存是否充足以及对Hadoop 的配置。如果大小超过一定阈值就写到磁盘，否则放入内存。

为了利用SDN controller建立全局的调度策略，优化代码插入位置，在NM获得全局调度信息而不用这样一个一个发送，以提高效率。我们做出了一个初步可用的HTTP Client POST，向PyCon控制器（SDN controller）POST必要的数据，如：源主机、目的主机、

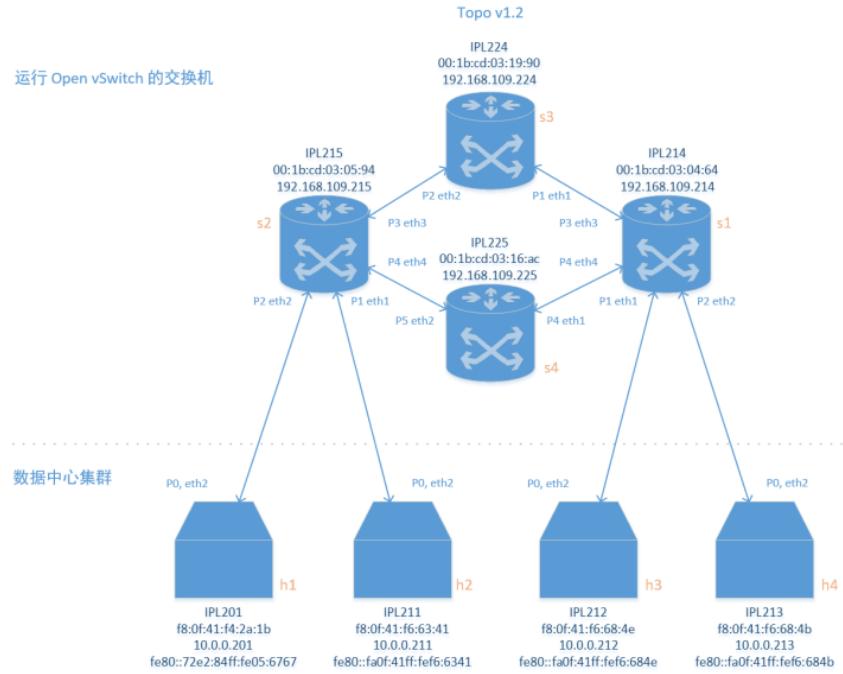


Figure 7: 网络拓扑结构

## 数据处理工作流图

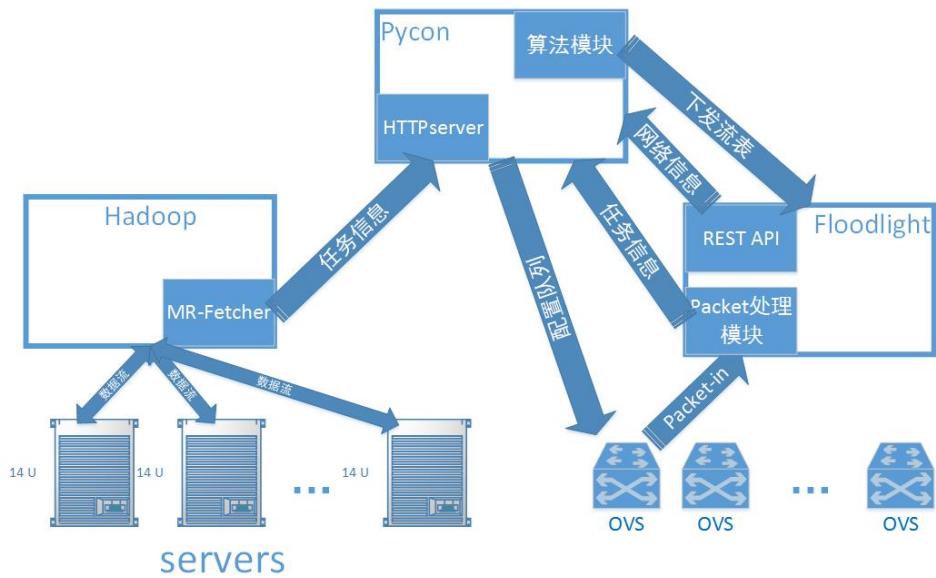


Figure 8: 基于SDN+Hadoop的系统架构设计

数据长度以及MapID等信息。

### **5.3 Python controller**

#### **5.3.1 HTTPserver**

3.孟

在hadoop中很多地方都用到了servlet，并且使用jetty作为servlet的容器来提供http的服务，其主要是通过org.apache.hadoop.http.HttpServer类实现的，HttpServer类是对Jetty的简单封装，通过调用HttpServer类的addServlet方法增加可以实现增加servlet到jetty的功能。

hadoop各个组件都会用到HttpServer,datanode/namenode,resourcemanager等。

Hadoop内嵌了Http服务器Jetty，主要有以下两方面的作用

1. Web访问接口，用于展示Hadoop的内部状态
2. 参与Hadoop集群的运行和管理

配置队列，进行限速

获取网络信息

下发流表

#### **5.3.2 路由算法模块**

4.孟

1.加权最短路径Dijkstra、矩阵处理利用可用带宽

2.最小瓶颈（Flow的传输时间最短）使用了网络带宽信息，最先完成优先

### **5.4 Floodlight REST API**

5孟

#### **5.4.1 Packet-in模块**

目的：取得源端口号

修改了Floodlight，添加了一个packet in 数据包处理模块。可以提取TCP Source Port 和Payload 内的Request URI 并且发送给PyCon（之所以这么做是因为在Hadoop MR 模块源码内无法通过HTTPURLConnection 直接拿到Source Port）

简单解释一下packet in 是什么包。

#### **5.4.2 REST API**

2.孟

获取流的端口信息

## 6 项目成果

## 7 项目总结

(1).当前coflow中所有的flow都是同时出现的。考虑到实际MR中task的启动时间、map完成时间都是不相同不确定的，带来的现实情况是coflow中的flow很大可能是不同时出现的。所以我们想采用simulator完成简单的various-start-time-coflow调度工作(2).将上述工作用varys实现，并采用ovs完成数据导流工作，实现整个实际平台的运行，并测试结果，对比分析。(3).考虑更多的研究方向，如：各个coflow/flow的依赖性，除了network层面的coflow调度，把task执行联合考虑进来，例如把task尽量放到所需data的机器上。

SDN性能瓶颈分析通过分析sdn网络的工作流程，可知控制器通过响应packet-in消息发送packet-out/flow-mod消息的速度是非常重要的，它的快慢直接影响了控制器拓扑发现，流表下发，mac地址学习能力，甚至整个网络的性能。而且SDN网络中通常采用反应式流安装，控制器的响应时间直接影响着流安装的处理速度，本文将重点测试在负载不同的情况下控制器处理packet-in消息的吞吐量和响应时间。同时也关注控制器支持创建openflow连接的能力与拓扑更新的速度。

## 8 参考文献

### References