一种基于多 SDN 控制器的交换机迁移机制

李 婉[1],沈苏彬[1],吴振宇[2]
(1. 南京邮电大学 计算机学院,江苏 南京 210003;
2. 南京邮电大学 物联网学院,江苏 南京 210003)

摘 要:为了解决 SDN(软件定义联网)控制平面的扩展性问题,业界提出了逻辑上集中、物理上分布的多 SDN 控制器架构,但是该架构中 SDN 控制器和 SDN 交换机之间是静态的映射关系,无法动态适应网络中流量的变化,造成了控制器负载不平衡。为了有效解决多 SDN 控制器部署方案中控制器之间负载不平衡的问题,需要实时监控和共享 SDN 控制器之间的负载情况,提出一种基于多 SDN 控制器的交换机动态迁移机制。该机制在控制器失效或控制器负载过大时能够将 SDN 交换机无缝迁移到其他正常的或负载较轻的控制器,避免了控制器单点失效或者负载过大。实施原型系统基于 Floodlight 控制器。实验结果表明,该方法能够很好地实现控制器之间的负载均衡,减少控制器对 Packet-in 消息的响应时间,提高系统的整体性能。

关键词:软件定义联网;交换机迁移;多 SDN 控制器;负载均衡
中图分类号:TP393 **文献标识码**:A **文章编号**:1673-629X(2018)01-0089-06
doi:10.3969/j.issn.1673-629X.2018.01.019

A Switch Migration Mechanism Based on Multiple SDN-controllers

LI Wan[1], SHEN Su-bin[1], WU Zhen-yu[2]
(1. School of Computer, Nanjing University of Posts and Telecommunications, Nanjing 210003, China;
2. School of IoT, Nanjing University of Posts and Telecommunications, Nanjing 210003, China)

Abstract: In order to solve the issue of scalability in SDN (Software Defined Networking) control plane, the multiple SDN-controllers architecture with logical centralization and physical distribution is put forward in the industry. However, it cannot dynamically adapt to the change of traffic in the network and causes a load-imbalance among SDN-controllers due to the statically configured mapping between a SDN-controller and a SDN-switch. In order to solve the problem above, the load is needed to be monitoring and sharing between SDN controllers, and a dynamic migration mechanism based on multiple SDN controllers is proposed which can seamlessly migrate a SDN-switch to a normal SDN-controller or that with less load, avoiding the single point faulure or high load of controller during the event of controller failure or overload. A prototype system based on Floodlight is built. Experimental results show that the proposed method achieves a better load balancing between controllers, reduces the response time of the Packet-in message, and improves the performance of the whole system.

Key words: software defined networking; switch migration; multiple SDN-controllers; load balancing

0 引 言

软件定义联网(Software Defined Networking, SDN)是一种新型的网络架构,实现了控制平面和数据平面的分离,能够更好地管理网络流量,为网络及其应用提供了良好的平台。

随着支持 OpenFlow 设备的增加,数据平面大量的流量被发送到控制平面,引起网络中 SDN 控制器的负载增加,SDN 控制器受到性能和容量的限制,有可能限制了网络的性能。传统的 SDN 设备依赖于单个集中化的控制器,控制器是 SDN 网络的核心,拥有全局网络视图,集中控制整个网络,处理交换机发送过来的大量请求。虽然 SDN 控制器从单线程发展为多线程,但是单控制器在可扩展性、可靠性、安全性等方面存在固有缺陷[1],因此,业界提出了逻辑上集中、物理

收稿日期:2017-01-21 **修回日期**:2017-05-25 **网络出版时间**:2017-09-27
基金项目:国家自然科学基金资助项目(61502246);江苏省科技计划项目(未来网络前瞻性研究项目)(BY20130951108)
作者简介:李 婉(1991-),女,硕士研究生,研究方向为计算机网络;沈苏彬,博士生导师,教授,CCF 高级会员(E200005482S),研究方向为计算网络、下一代电信网及网络安全;吴振宇,博士,讲师,CCF 会员(200061869M),研究方向为数据挖掘和人工智能。
网络出版地址:http://kns.cnki.net/kcms/detail/61.1450.TP.20170927.1000.076.html

上分布的多 SDN 控制器部署方案 HyperFlow[2]、Kandoo[3]和 Onix[4],多个控制器协同工作,实现对整个网络的集中式控制,有着更好的扩展性。

最早提出的分布式控制平面 HyperFlow,实际上由物理上分布、逻辑上集中的多 SDN 控制器组成,在提供可扩展性的同时保持网络集中控制的优点。HyperFlow 根据网络规模设置多个管理域,每个管理域中的控制器独立管理自己区域的交换机,只与邻近的控制器交互,不主动地与远处的控制器进行通信,减少了控制器和交换机之间的流配置时间。Kandoo 采用分层思想将控制器分为两层:根控制器和本地控制器。根控制器维护网络的状况,本地控制器主要负责管理交换机。

Onix 主要包括物理基础设施层、连接基础设施、Onix 和控制逻辑四部分。物理基础设施层由路由器、交换机和其他网络元素组成,主要为 Onix 提供读写状态的 API;连接基础设施为物理基础设施层和 Onix 提供通信通道;Onix 为控制逻辑提供到网络的可编程访问接口;控制逻辑决定网络的行为,其包含一个 NIB(Network Information Base,网络信息库),用于维护全网的状态。由此可知,目前的分布式控制平面研究的重点是 SDN 控制器在正常工作的情况下,如何实现分布式控制器实例之间一致的状态。多控制器部署方案[2-4]中控制器和交换机之间的关系是静态配置的,但是网络中的流量是动态变化的。如果一段时间内,某一控制器管辖的交换机中流量突增,将导致控制器负载过大,紧接着导致控制器响应时间过长,从而降低了整个网络的性能。而其他控制器有可能处于闲置状态,引起控制器之间负载不均衡,造成了控制器资源的浪费。

为解决 SDN 控制器负载不均衡问题,文献[5-6]提出的弹性分布式控制器 ElastiCon 能够根据网络中的负载动态减少或增加控制器。为了实现弹性分布式控制器,提出了交换机迁移协议。但是该方案考虑的情况比较简单,只有一个控制器负载过载,采用就近迁移的策略。虽然可以减少交换机和控制器之间的延迟,但是如果邻近控制器的负载过大,将导致多次迁移交换机,效率并不是很高,甚至需要添加新的控制器。

针对上面多控制器负载失衡的问题,文中提出一种交换机动态迁移机制。该机制基于控制器角色实现交换机的迁移,并且提出一种调度算法,该算法综合考虑了控制器的负载和交换机与控制器之间的传输时延两个因素,选择合适的交换机和目标控制器后,控制器通过改变其角色、与其连接的活跃交换机的个数,完成交换机的迁移。为了评价其性能,验证了控制平面的响应时间和开销量,并和传统的静态配置作对比。

1 相关技术分析

和传统网络相比,SDN 在灵活性和管理性等方面表现出了很多优点,但随着网络的急剧扩张,控制器的性能、可扩展性和可靠性逐渐成为瓶颈。另外,一旦控制器失效,将导致整个网络面临瘫痪。所以单 SDN 控制器很难应用于大型网络中。为此,研究人员提出了分布式控制器架构来提高 SDN 控制平面可扩展性。但是,目前大多数方案中交换机与控制器实例之间静态的映射关系会因为流量的动态变化造成控制器负载不均衡。针对上面的问题,ADVAIT 等提出了弹性分布式控制器 ElastiCon。ElastiCon 提出了一种负载窗口机制,当负载超过上、下阈值时,采取增加新控制器、减少已有控制器或迁移交换机的动作,提高了控制器的资源利用率。但是 ElastiCon 并没有明确怎样选择增添和删除的控制器数量及位置。另外,文献[5]是将交换机迁移到最近的控制器,减少了控制器和交换机之间的时延,但是没有考虑到邻近控制器的负载,有可能导致邻近控制器又超载,紧接着可能多次迁移交换机。因此,如何精细控制控制器的负载,减少交换机的迁移频率,实现多控制器之间的负载均衡是必须要解决的问题。文献[7]提出了一种弹性方案,在不同的情况下,动态改变控制器的个数和位置。文献[8]介绍了在集群中动态添加或移除控制器但没有产生引起网络中断的算法,同时也提到了给控制器动态分配交换机的必要。文献[9]使用利用率最低的迁移策略,每次都将交换机迁移到剩余容量不经常使用的控制器下,没有考虑到控制器和交换机之间的传输时延;另外,也没有实现总体的负载均衡。

目前的交换机迁移算法大部分都是选择负载较大的控制器下的交换机进行迁移,然后判断迁移后的控制器负载是否过大,如果过载则继续进行迁移。虽然这种迁移方式的单次迁移效率较高,但是对于大规模传输速率较大的网络来说,该迁移方式仍需要花费较大的代价。所以,如何对负载过大的控制器下的交换机进行迁移,提高迁移效率也是一个必须要解决的问题。

文中提出的算法综合考虑了控制器的负载和交换机与控制器之间的传输时延两个因素,然后在控制器和交换机位置确定的情况下,当控制器之间的负载大于事先设置的阈值时,交换机将原来连接的主控制器变为从控制器,将其中的一个从控制器变为一个新的主控制器。发生迁移的交换机必须满足迁移到的新的主控制器的负载没有超过其最大负载和交换机迁移之后控制器之间的负载更均衡两个条件。当两个条件都满足的情况下,选择与控制器之间传输时延最小的交换机进行迁移。

2　方案设计

交换机迁移的过程其实是根据网络中的流量重新部署交换机和控制器之间的关系。OpenFow1.3.4 协议[10]提到每个交换机可以连接一个主控制器和多个从控制器。另外,OpenFlow 协议还介绍了多种消息,交换机如果收到控制器发送的 Role-request 消息,会根据请求的控制器角色类型做出回复,并向控制器发送 Role-reply 消息,告诉控制器是否允许修改角色。主控制器域内的交换机在某时刻如果流请求不断激增的话,需要对该域内交换机实现向外迁移,迁移之前,需要从多个从控制器集合中选择一个从控制器作为该交换机新的主控制器,同时把原来的主控制器变更为从控制器。

本节展示了如何定义和估算控制器负载,并讨论了基于交换机迁移的负载均衡模式。管理框架包含三个功能,说明如下:

(1)负载监测功能:周期性收集和计算控制器的负载,并协调维护全局负载信息表。

(2)负载调度功能:检查每个控制器负载信息表,并决定是否执行交换机迁移,哪个控制器应选为主控制器实施负载转移,哪个交换机执行迁移。

(3)交换机迁移功能:控制器协调交换机迁移和改变交换机与控制器之间的映射。

2.1　负载监测

负载监测主要用于周期性监测控制器的负载,设置阈值,判断控制器的负载状态。文献[11]主要考虑两个负载信息:交换机指标和控制器指标。交换机指标包括与控制器连接的活跃交换机的个数和与控制器相连的所有交换机的 Packet-in 消息请求速率。控制器指标是控制器收集到的所有负载信息,包括当前的 CPU 使用率和内存使用率。使用具体的负载值表示控制器真实的负载信息,但是对于不同的网络状况,可能会有不同的负载信息。所以,在实际网络中,为了表示不同的负载信息,可以引入系数,指示了综合负载计算中不同负载信息的权重。由文献[5]可以看出,CPU 是控制器的主要限制。因此,控制器的负载主要监测主机的 CPU 使用率、内存使用率和控制器每秒处理 Packet-in 的消息个数。

假设网络中有 N 个控制器$\{C_1,C_2,\cdots,C_n\}$和 M 个交换机$\{S_1,S_2,\cdots,S_m\}$。根据网络中的状态调节收集和计算负载信息的时间间隔。如果时间间隔较短,虽然可以获得一个准确的负载信息,但是会引起额外的系统开销和通信开销。如果时间间隔较长,则不能准确地描述控制器的负载情况。根据控制器的负载动态调整收集和计算负载信息的时间间隔,不仅可以减少系统开销,还可获得准确的负载信息。

2.2　负载调度

负载调度算法根据负载信息表决定是添加控制器、减少控制器还是执行交换机迁移。迁移交换机过程中哪个控制器应选为主控制器实施负载转移,以及哪个交换机执行迁移。

算法 1:整体算法。

```
While (true) do
if ( Ldif_i > C )  then
Balance( C_i , S_j );
end if
if (Max_Load_Ratio> α ) then
Add_controller;
Else if( Max_Load_Ratio< β )then
Sub_cohtntroller;
End if
```

(1)是否增加/减少控制器。

根据文献[1]可知,控制器每秒能够处理有限的数据包,如果控制器每秒处理的数据包个数较少,则会造成控制器资源的浪费,因此为控制器每秒处理数据包的个数设置上下限阈值,分别为 α 和 β。应该做什么操作由算法 1 确定。如果资源池中控制器的负载超过事先设置的阈值 α,则添加新的控制器。如果资源池中控制器的负载低于事先设置的阈值 β,则关闭或者移除部分控制器。如果在两者之间,则需要一直监测控制器的负载,判断是否需要平衡控制器之间的负载。

(2)是否执行交换机迁移。

是否执行交换机迁移由控制器的负载状况决定。具体的迁移策略参见算法 2。其中,LOAD(C_i)表示控制器 C_i 的负载,按照从小到大的顺序列出控制器的负载,最重的负载和最轻的负载之差为 Ldif_i。当 Ldif_i 比阈值 C 大时,交换机发生迁移。一个合适的阈值 C,可以实现预想的负载平衡,避免频繁地迁移交换机。

算法 2:平衡控制器之间的负载。

```
While (LOAD( C_i )Table! =NULL) do
Sort_Descending(LOAD( C_1),…,LOAD( C_n ))
if ( Ldif_i > C ) then
Sort_ Descending( S_1,…, S_m ) to Migrate
根据式(1)计算函数值
MigrateS_j to C_i
End if
UpdateC_iS_j mapping
end while
```

(3)哪个控制器选为主控制器。

该系统中设置一个协调节点,一个特殊的控制器。协调节点负责收集每个控制器的负载信息,并且计算系统具体的负载。协调节点也有交换机和控制器之间

的静态映射和负载信息表。协调节点通常是第一个加入到系统中的节点。一旦协调节点发生故障,其他控制器可以迅速修改角色变成协调节点。根据负载信息表,负载最轻并且时延满足的控制器选为主控制器,实现某个交换机的迁移。为了实现控制平面的扩展性,当在该组加入新的交换机时,负载较轻的控制器作为主控制器。

(4)哪个交换机应该迁移。

具体选择哪个交换机实现迁移对其负载平衡有着重要的影响。如果选择迁移 Packet-in 请求速率较高的交换机,将导致新的主控制器的负载较大。如果选择迁移 Packet-in 请求速率较小的交换机,将导致多次迁移。

文献[6]中提到控制器的响应时间直接影响着流安装的处理速度。控制器的响应时间表示交换机和控制器相连后,交换机发送一个消息给控制器,控制器对该消息进行处理,并将结果返回给相应的交换机的过程所使用的时间。对于交换机是否应该迁移,使用响应时间这个参数,设置一个最初的权值 W_i,表示交换机选择的优先级。

$$W_i = W_i + R * \sqrt[3]{RTT_i - RTT_i'} \quad (1)$$

其中,RTT_i 表示交换机迁移之前,按照前一时间段内数据流的请求情况下控制器的响应时间。假设该交换机可以迁移;RTT_i' 表示迁移之后,按照迁移之后的数据流请求情况下的响应时间;R 是一个系数。

如果 W_i 权值变大,说明交换机不适合发生迁移,接着查找其他的交换机,如果所有的 RTT_i' 都比 RTT_i 大,则在负载最小的控制器下,选择 RTT_i' 最小的交换机进行迁移。

根据算法 2,对该控制器管控的交换机在时间间隔内产生的平均流请求进行降序排序,初始时选择平均流请求最大的交换机作为待迁移的对象,然后按照上面介绍的方式依次遍历该控制器管控下的所有交换机,直到选择合适的交换机和控制器。

2.3 交换机迁移

设计交换机迁移机制,当控制器负载较大时,实现将交换机资源迁移到负载较小的控制器管理域内[5-6]。为了保证交换机迁移过程中消息的正常处理,需遵循两个原则:

(1)活跃性:交换机在迁移过程中,要保证与该迁移交换机至少有一台控制器处于正常运行状态,不能直接将控制器的角色改成主模式来完成交换机的迁移工作。例如控制器在迁移开始之前,发送了一条 Flow-mod 删除消息到交换机中,在交换机迁移之前还没有回复 Flow-removed 消息到控制器中,那么迁移之后会造成信息**丙考数据**状态的不一致性。

(2)安全性:交换机在迁移过程中,要保证只能有一台控制器处理交换机的异步消息。例如多台控制器对同一交换机的 Packet-in 消息进行处理,会造成流表项的多次安装,还会造成分布式数据存储的不一致性。

3 系统实现

3.1 控制器部署

文中采用 ZooKeeper[12] 技术搭建多个控制器,ZooKeeper 是一个简单高效的分布式协调器。当服务器启动后,从配置文件设置的服务器中选择一台 SDN 控制器作为领导者,其余控制器便成为跟随者。领导者负责对多个 SDN 控制器进行管理并提供北向接口服务,跟随者主要负责管理控制交换机。

ZooKeeper 采用了原子广播协议,分为广播模式和恢复模式。广播模式可用于数据同步,保证了控制器之间的一致性。服务启动或在领导者崩溃后,恢复模式用于选举领导者。ZooKeeper 组服务的功能,可以感知集群中控制器的添加和移除。Zookeeper 的观察机制可以实时同步控制器的状态信息。

控制器之间可以通过 Zookeeper 进行通信,协商完成交换机的迁移,并且更新交换机和控制器之间的映射关系。

每台控制器都有两部分信息:控制器节点的负载信息和控制器与交换机之间的角色映射关系。为了更好地模拟大型网络,将整个系统划分为若干个更小的范围。这里采用分组的形式,每组控制器只知道自己的完整拓扑结构,不知道其他组的拓扑结构,减少了网络上的通信量。

如图 1 所示,对控制器进行分组,在部署过程中,各分组物理上是分布的,交换机就近选择控制器,减少了控制器和交换机之间的传输时延。每组至少有两台控制器,在主控制器失效的情况下,集群能够从多台从控制器中选择一个作为新的主控制器,避免了控制器的单点故障。当其中一台控制器负载过大,交换机可以迁移到其他的控制器,实现了集群的负载均衡。同时,控制器对交换机透明化,交换机不需要知道接受哪台控制器的控制,保证了逻辑上的集中。

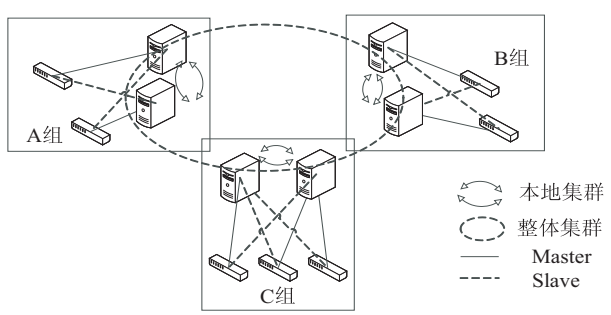

图 1 多控制器部署方式

3.2　负载均衡的实现

主要通过修改运行中的控制器角色完成交换机的迁移过程,从而均衡控制器之间的负载。通过对 OpenFlow 协议的学习,文中提出的迁移机制由从控制器发送 Role-request 消息开始,后续阶段对其响应,逐步完成交换机的迁移。假设交换机(S)分别与主控制器(C_1)和从控制器(C_2)相连,当 C_1 的负载较大、C_2 的负载较轻时,通过 S 实现负载的转移。交换机的迁移过程如图 2 所示,主要分为 4 个阶段。

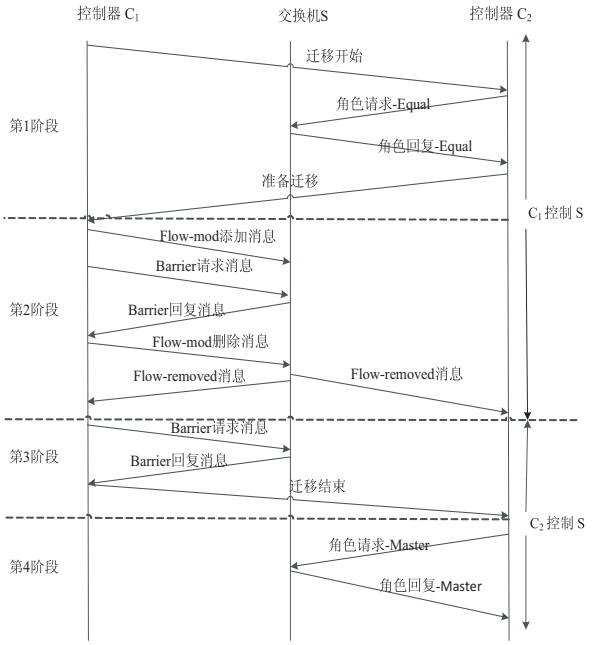

图 2　交换机的无缝迁移过程

第 1 阶段:C_2 请求修改角色为对等角色。对于待迁移的 S 而言,C_2 首先由从控制器变为对等控制器。C_1 通过控制器间信道发送一个开始迁移消息给 C_2,开始交换机迁移阶段。C_2 发送一个 Role-request 消息到 S 中,请求将角色变成过渡角色对等角色。在 C_2 从 S 收到 Role-reply 之后,角色将变为对等。只有当 C_2 变为对等角色后,才可以接收到 S 发送的异步消息,但是此时 C_2 忽视这些消息,不需要响应这些消息。在该阶段,C_1 仍是唯一的主控制器,能够处理来自 S 的所有消息,保证了活跃性和安全性。

第 2 阶段:插入和删除空流表。为了确保迁移的精确时刻,C_1 发送一个空的 Flow-mod 命令给 S,用于添加一个空流表项,但是该流表项无法匹配即将到来的任何数据包。这里,假设所有的控制器都知道该空流表项。然后,C_1 再发送另外一个 Flow-mod 消息用于删除该流表。相应的,S 将会发送 Flow-removed 消息给 C_1 和 C_2。Flow-removed 事件用于确认删除空流表项。之后,只有 C_2 处理 S 的异步消息。另外,在插入和删除空流表中间使用 Barrier 消息,用于确保在插入任何消息之前的所有消息都被处理完了,而不是被删除了。

第 3 阶段:使用 Barrier 消息刷新待处理请求。虽然 C_2 在第 2 阶段已经接管了 S,但 S 中可能还有正在处理的请求,C_1 所需要做的就是处理完所有在 Flow-removed 之前的消息并且下发给 S。因为,交换机并没有明确地确认信息可以确保所有的消息都被 C_1 处理了。如果 C_1 没有发送一个信号给 C_2,则 C_1 无法变为从控制器,S 也将忽略之前的所有操作。因此,为了确保所有的消息都能被处理,C_1 发送一个 Barrier-request 消息到 S,并等待 S 回复 Barrier-reply 消息。此时,C_1 与 S 之间的迁移正式结束。

第 4 阶段:C_2 请求修改角色为主控制器。C_2 发送 Role-request 消息给交换机将其角色改为主控制器。同时,C_2 更新分布式数据。S 在收到 Role-request 消息后,返回 Role-reply 消息给 C_2,S 自动将 C_1 的角色修改为从控制器。至此,S 的迁移过程正式完成,C_2 开始处理 S 的请求。

4　仿真实验与分析

4.1　实验环境

实验基于 Ubuntu14.04 系统,采用 Floodlight[13] 控制器组成的控制器集群,在 Mininet[14] 仿真平台上进行仿真验证。

4.2　性能评价和测试

在 SDN 网络中,控制器的性能主要体现在短时间内尽可能完成流表项的制定和下发,可以采用吞吐量和响应时间两个指标测试控制器的性能。

吞吐量评估不同的控制器在不同数量线程条件下,能够处理交换机发过来的数据流请求的最大平均流量。响应时间主要评价控制器在被动流表安装模式下,流表项设置过程中产生的最大、最小和平均时延。

(1)吞吐量。

本节测试文中提出的基于集群的部署方案和交换机迁移机制的性能,并且与传统的静态配置方案进行对比。

使用 Cbench 测量 Floodlight 控制器的性能,受到本身硬件的影响,测量时参数设置每台交换机连接 100 台主机,通过改变交换机的个数,经过多次测量,四核 CPU 的单台 Floodlight 控制器每秒最多大约可以处理 48 600 个数据包。Packet-in 消息是 Floodlight 控制器众多模块处理的重点,因此,统计某一段时间内 Floodlight 控制器处理 Packet-in 消息的个数反映了系统的处理性能。在 Floodlight 控制器源码中添加了一个统计 Packet-in 消息的模块,用于计算系统所处理的消息个数。

逐渐部署 1 到 5 台 Floodlight 控制器,每台控制器

连接一台交换机,每台交换机连接一台主机,主机用于在网络中产生 UDP 数据流。流表的下发有主动和被动两种方式。为了使到达控制器的数据流的速率最大,交换机采取被动安装流表的策略。在控制平面,对其应用做一些修改,禁用交换机安装流表项的功能。这意味着交换机接收到的数据包都是新的流,都要封装成 Packet-in 消息发送给控制器。

图 3 表明 CPU 是限制控制器性能的主要原因,系统整体的吞吐量随着控制器个数的增加大约呈线性增加的趋势,表明系统具有良好的伸缩性,可以通过增加网络中控制器的个数来提高控制平面的处理能力。因此基于集群的多控制器部署方案具有良好的可扩展性。

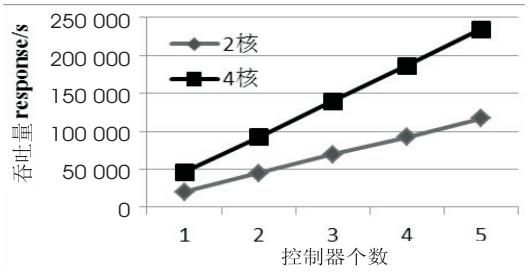

图 3 不同控制器个数的吞吐量

为了测量迁移交换机的方式可以实现负载均衡,实验中部署了 5 台控制器和 20 台交换机,交换机随机连接控制器,然后模拟向不同控制器中不均匀地注入大量的数据流,分别测试迁移之前和之后每台控制器的资源使用情况。

从图 4 中可以看出,静态配置关系中控制器 1、2、3 的资源使用率都超过 80%,而控制器 4、5 的资源使用率在 20% 左右,说明控制器之间负载很不均衡。使用迁移机制之后,每台控制器的资源使用率在 60% 左右,说明交换机迁移的机制可以较好地平衡各控制器负载。

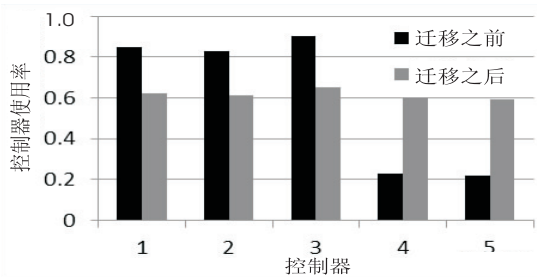

图 4 迁移前后控制器的资源利用率

(2)响应时间。

对于交换机迁移过程中响应时间的测量,实验环境中,集群中部署了两台 Floodlight 控制器 C_1 和 C_2,8 台 OpenvSwitch 分别连接两台控制器,每台交换机分别连接一台主机。模拟三种不同的负载,主机使用 Iperf 工具发送不同速率的数据包。

为了模拟控制器之间负载失衡的情况,刚开始时,交换机 S_1、S_2、S_3 和 S_4 连接控制器 C_1 作为主控制器和 C_2 作为从控制器,交换机 S_5、S_6、S_7 和 S_8 连接控制器 C_2 作为主控制器和 C_1 作为从控制器。在交换机和控制器之间静态映射的情况下,测量三种不同负载(见表 1)下控制器的响应时间,具体结果如表 2 所示。

表 1 三种不同的负载(数据包个数)

负载	S_1 和 S_2	S_3 和 S_4	S_5 和 S_6	S_7 和 S_8
A	4 000	8 000	4 000	8 000
B	4 000	8 000	8 000	16 000
C	4 000	8 000	12 000	20 000

表 2 不同负载下控制器的响应时间 ms

负载	控制器 C_1		控制器 C_2	
	静态	迁移	静态	迁移
A	23.5	25.0	24.8	25.3
B	24.5	26.2	30.3	26.4
C	25.3	30.1	38.4	31.2

负载 A 情况下,在静态配置和迁移机制下,控制器的响应时间差别不大。对于负载 B 和负载 C 两种情况,控制器 C_2 的响应时间有很大差别。根据实验可知,集群带来的时延是可以接受的。当在重负载情况下,通过迁移交换机,集群可以使时延更低。

5 结束语

针对多控制器系统中负载不平衡引起的控制器性能下降的问题,提出将集群技术应用到多控制器系统中。为了解决多控制器负载不均衡的问题,提出了交换机资源的实时转移。与传统的静态映射关系相比,该方案可以有效提高系统的处理性能以及网络的吞吐量。下一步工作是考虑到交换机迁移的次数,进一步优化负载调度算法。

参考文献:

[1] TOOTOONCHIAN A, GORBUNOV S, GANJALI Y, et al. On controller performance in software-defined networks[C]// USENIX conference on hot topics in management of internet, cloud, and enterprise networks and services. [s. l.]: USENIX Association, 2012: 10.

[2] TOOTOONCHIAN A, GANJALI Y. HyperFlow: a distributed control plane for OpenFlow[C]//Proceedings of the 2010 internet network management conference on research on enterprise networking. [s. l.]: USENIX Association, 2010: 3.

[3] YEGANEH S H, GANJALI Y. Kandoo: a framework for efficient and scalable offloading of control applications[C]//Proceedings of the first workshop on hot topics in software defined networks. [s. l.]: ACM, 2012: 19-24.

(下转第 99 页)

低,耗时比文中方法稍短。为了减少文中方法的耗时,可以考虑使用更加优化的奇异值分解算法。

5　结束语

传统使用奇异值向量作为区分特征的人脸识别算法,包含的人脸有效信息较少,识别率较低。基于此,提出了一种基于 SVD 的两步人脸识别方法。首先将图像划分成块,把整体与局部奇异值向量组合成模板奇异值向量作为识别特征进行人脸的初步识别,获得候选人脸集;然后求待测样本图像与候选人脸整体正交矩阵的相似程度,以此作为识别特征进行二次识别,得到最佳决策脸。实验结果表明,该方法在识别率上明显优于常规方法,具有很大的现实意义。

参考文献:

[1] CAO D, YANG B. An improved face recognition algorithm based on SVD[C]//International conference on computer & automation engineering. [s. l.]:[s. n.],2010:109-112.

[2] TOLBA A S, EL-BAZ A H, EL-HARY A A. Face recognition: a literature review[J]. International Journal of Signal Processing, 2005, 2(1):88-103.

[3] 陈良瑜,朱振福,刘忠领,等. 图像奇异值特征矢量缩放不变性分析及应用[J]. 红外与激光工程,2003,32(5):498-501.

[4] HONG Z Q. Algebraic feature extraction of image for recognition[J]. Pattern Recognition, 1991, 24(3):211-219.

[5] HE Yunhui. An efficient method to solve small sample size problem of nonlinear discriminant vectors in feature space for face recognition[C]//International conference on computational & information sciences. [s. l.]:[s. n.], 201:117-120.

[6] TIAN Y, TAN T, WANG Y, et al. Do singular values contain adequate information for face recognition? [J]. Pattern Recognition, 2003, 36(3):649-655.

[7] 张慈祥,刘　辉,强振平. 基于稀疏表示和奇异值分解的人脸识别[J]. 计算机应用,2013,33(S1):233-235.

[8] 程永清,庄永明,杨静宇. 基于矩阵相似度的图象特征抽取和识别[J]. 计算机研究与发展,1992,29(11):42-48.

[9] 吴俊政. 一种基于奇异值分解的图像压缩方法[J]. 计算机与数字工程,2009,37(5):136-138.

[10] 孙静静,张宏飞,孙　昌. 一种基于奇异值分解的人脸识别新方法[J]. 科学技术与工程,2010,10(25):6204-6208.

[11] 高全学,梁　彦,潘　泉,等. SVD 用于人脸识别存在的问题及解决方法[J]. 中国图象图形学报,2006,11(12):1784-1791.

[12] LALIBERTE A, KOPPA J, FREDRICKSON E, et al. Comparison of nearest neighbor and rule-based decision tree classification in an object-oriented environment[C]//International symposium on geoscience and remote sensing. [s. l.]:IEEE, 2006:3923-3926.

[13] 王孝青,党亚民,成英燕,等. 基于矩阵相似度的 InSAR 图像配准方法研究[J]. 测绘科学,2008,33(6):44-46.

[14] BAI L, VELICHKO A, DRINKWATER B. Ultrasonic characterization of crack-like defects using scattering matrix similarity metrics[J]. IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control, 2015, 62(3):545-559.

[15] LIU Yu, CHEN Xun, WARD R K, et al. Image fusion with convolutional sparse representation[J]. IEEE Signal Processing Letters, 2016, 23(12):1882-1886.

(上接第 94 页)

[4] KOPONEN T, CASADO M, GUDE N, et al. Onix: a distributed control platform for large-scale production networks[C]//USENIX conference on operating systems design and implementation. [s. l.]:USENIX Association, 2010:351-364.

[5] DIXIT A, HAO F, MUKHERJEE S, et al. Towards an elastic distributed SDN controller[J]. ACM SIGCOMM Computer Communication Review, 2013, 43(4):7-12.

[6] DIXIT A A, HAO F, MUKHERJEE S, et al. ElastiCon: an elastic distributed SDN controller[C]//Proceedings of the tenth ACM/IEEE symposium on architectures for networking and communications systems. [s. l.]:ACM, 2014.

[7] BARI M F, ROY A R, CHOWDHURY S R, et al. Dynamic controller provisioning in software defined networks[C]//IEEE/ACM/IFIP international conference on network and service management. [s. l.]:IEEE, 2013:18-25.

[8] YAZICI V, SUNAY M O, ERCAN A O. Controlling a software-defined network via distributed controllers[C]//Proceedings of the 2012 NEM summit. [s. l.]:[s. n.], 2012:16-20.

[9] YAO G, BI J, LI Y, et al. On the capacitated controller placement problem in software defined networks[J]. IEEE Communications Letters, 2014, 18(8):1339-1342.

[10] Open Networking Foundation. OpenFlow switch specification version 1.3.4[EB/OL]. 2014-05-27. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.3.4.pdf.

[11] LIANG C, KAWASHIMA R, MATSUO H. Scalable and crash-tolerant load balancing based on switch migration for multiple open flow controllers[C]//Second international symposium on computing and networking. [s. l.]:IEEE, 2014:171-177.

[12] Apache Software Foundation. Apache Zookeepr[EB/OL]. 2016. http://zookeeper.apache.org/.

[13] Floodlight OpenFlow Controller-Project Floodlight[EB/OL]. 2012. http://www.projectfloodlight.org/floodlight/.

[14] LANTZ B, HELLER B, MCKEOWN N. A network in a laptop: rapid prototyping for software-defined networks[C]//ACM workshop on hot topics in networks. Monterey, CA, USA: ACM, 2010:1-6.