第 43 卷 第 6A 期
2016 年 6 月

计 算 机 科 学
Computer Science

Vol. 43 No. 6A
June 2016

基于 HDFS 的海量视频数据重分布算法

郭建华 杨洪斌 陈圣波

(上海大学计算机工程与科学学院 上海 200444)

摘 要 基于视频数据的分布式计算与基于文本类型数据的分布式计算存在很大的差异。视频数据本身是非结构化的,并且对于同样大小的视频,若其内容不同会导致任务执行消耗的时间也不同。对于简单的结构化数据,HDFS 默认的负载均衡器能够解决负载均衡的问题。但是视频文件存在热点访问以及复杂度不一致的问题。使用 HDFS 默认的数据分布机制不能很好地解决计算负载均衡问题。因此提出了一种基于 HDFS 的海量视频数据重分布算法。首先对视频文件的访问次数以及历史视频分析对视频文件的访问时间进行记录,然后对数据进行量化之后将其加权作为该视频文件的负载度,最后使用文件置换手段将负载高的视频与低的视频进行置换,直到每个节点的负载达到均衡为止。实验结果表明,使用提出的数据重分布算法可以减少海量视频数据的处理时间。

关键词 HDFS,数据重分布,视频复杂度,视频热度

中图法分类号 TP399 文献标识码 A

Weight Distribution Algorithm for Massive Video Data Based on HDFS

GUO Jian-hua YANG Hong-bin CHEN Sheng-bo

(School of Computer Engineering and Science, Shanghai University, Shanghai 200444, China)

Abstract There is a big difference between the distributed computing based on the video data and the distributed computing based on the text type data. The video data are unstructured, and the same size of the video that has different content will lead to different execution time. For simple structured data, the default load equalizer of HDFS can solve the problem of load balancing. But the video file has the problem of different access times and complexity inconsistency. Using the default data distribution mechanism of HDFS are not well solve the load balancing problem. In this paper, a new algorithm for massive video data redistribution based on HDFS was proposed. Firstly, the access times and the history analysis time of the video file are recorded. Secondly, the data are quantified and weighted as the load of the video file. Lastly, the means of file replacement are used to exchange high load video and low load video, until each node achieves load balancing. Experimental results show that using the data redistribution algorithm proposed in this paper can reduce the processing time of massive video data.

Keywords HDFS, Data redistribution, Video complexity, Video popularity

1 引言

云计算是一种基于互联网的计算方式,包括虚拟化、大数据分布式计算、高性能并行计算、大规模数据与网络存储等技术。其主要原理是将一个大的任务划分为若干小任务,然后将这些小任务分配给多台机器并行执行,多台机器通过网络通信的方式协作处理,最后将所有的计算结果进行汇总。云计算在近几年得到了飞速的发展,其中 Apache 基金下的 Hadoop 在这几年最为火热,在工业界、学术界引起了广泛的关注。但是 Hadoop 计算架构是基于磁盘的计算模式,效率方面存在很大的问题,随着内存价格越来越低,基于内存的分布式计算 Spark 得到了飞速的发展。Hadoop 分布式系统的基础架构主要包含两大模块:HDFS 和 MapReduce[1],其中 HDFS 分布式文件系统负责系统的数据存储、文件管理和容错性,能够存储海量的数据,并且对数据的格式没有限制;MapReduce 基于磁盘的分布式计算,在性能上存在一些问题,Spark 所提供的基于内存的分布式计算在效率方面较 MapReduce 有巨大[2,4,8]提升。

基于视频监控的安防系统在现实生活中得到了广泛的使用,现在各个街道、校园、商店、广场等都布满了各种各样的摄像头。由于摄像头越来越多且视频越来越高清,视频数据量不断增多,如何在海量的视频数据中找到有价值的信息对视频处理提出了新的挑战。例如对于在 2012 年 1 月 6 日,南京发生了一起枪击抢劫案,事后公安局为了获得破案线索不得不通过查看监控视频的形式,其中派了五百多名警官不间断地观看事发一周的监控视频,投入了大量的人力和时间。由于人眼长时间下观看视频会出现注意力不集中等问题,有时会漏掉重要信息,因此靠人力的方式去挖掘视频中的价值是不现实的且代价很大。所以,迫切需要一个高效快速的智能化的海量视频数据的分析平台。

郭建华 硕士生,主要研究方向为大数据并行化处理;杨洪斌 副教授,主要研究方向为高性能计算等;陈圣波 讲师,主要研究方向为软件工程等。

基于 Hadoop 的大数据智能分析平台在这几年得到了广泛的应用[3]，但是视频数据与传统的文本数据存在很大的区别。对于同一个算法，大小相同的视频由于内容的不同会导致在计算时间上存在很大的差异性，这就使得基于文件大小来进行任务分配的机制已经不能满足视频大数据的处理要求。分布式平台的计算时间取决于最长的任务的执行时间，所以基于 HDFS 本身的特性加上视频内容的复杂度的特质，如果能够实现将复杂度小的文件块与复杂度大的文件块同时分配给相同的任务执行，那么系统的总执行时间就趋于均衡化了，这样系统的执行速度最快。

本文主要从两个方面来分析传统的视频存储存在的问题，首先是视频分析存在着热点视频，其次是基于视频的复杂度方面。视频文件存储在分布式文件系统之上，肯定会存在某些视频被频繁访问而有些视频则不会的情况，对于热度高的视频，我们希望它能够尽量均匀分布在文件系统中，因为基于数据本地性的任务调度方式能够最大化利用 CPU 资源，同时降低磁盘和网络的 IO 消耗[5,6,12]。而视频复杂度高的视频往往需要消耗更多的 CPU 计算资源，相同大小的视频文件在相同计算能力的情况下往往会需要不同的计算时间，例如对于人脸识别算法，如果某个视频中人脸的个数较多，该视频文件的计算时间就会长[7]。因此，如果能够考虑上述问题，就能够很好地使视频计算趋于负载平衡。

本文的主要思想是对视频文件的访问进行统计，同时对历史的视频分析进行统计，之后对统计的信息进行量化，然后加权，利用视频的热度和复杂度来决定该视频文件的负载程度。之后利用提出的文件置换策略以及置换算法对视频文件进行置换。

本文第 2 节介绍了目前利用 HDFS 存储视频文件时，上层分布式计算框架进行计算时存在的问题；第 3 节提出了视频存储的重分布策略，并提出了视频置换的算法；第 4 节分析与讨论实验结果；最后总结全文并对未来作出展望。

2 相关工作

Hadoop 设计之初是为了部署到廉价的普通机器之上，因此对系统的容错性方面要求较高。在 HDFS 中，为了保证数据的安全，采用了冗余备份的机制，这里的备份是将数据复制到多台节点之上，默认的备份数是 3[1,9]。为了更好地了解 HDFS 的数据备份机制，首先需要了解客户端是如何将数据写入 HDFS 中的。

如图 1 所示，客户端先对 namenode 发送请求创建文件，namenode 在检查确保元数据中不包含此文件之后，赋予客户端创建文件的权限。之后客户端开始将数据写入 datanode 中。在数据进行备份的时候，多个 datanode 会构成一个管线，数据先以流的形式传输到第一个 datanode，然后第一个 datanode 负责将存储的数据发送给第二个 datanode，同样地，第二个 datanode 存储完数据会将数据发送到第三个 datanode。在 HDFS 中数据的备份机制采用的是运行客户端的节点存放第一个副本（客户端如果运行在集群之外，会随机选择一个节点作为第一个副本存放节点，系统会避免挑选那些负载太大或者太忙的节点），备份一份数据到本机架内的另外一个节点，另外一份备份到跨机架的一个节点。

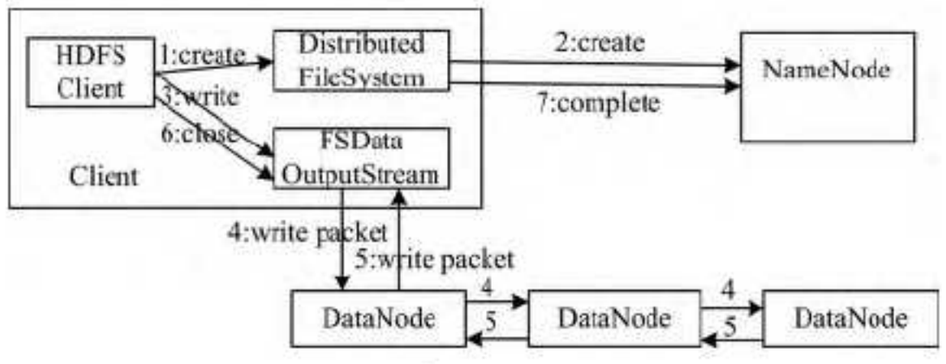

图 1 数据写入 HDFS

在 HDFS 中默认的副本存放策略一方面能够为系统提供高容错性，另一方面能在一定程度上增加并行计算框架（如 Spark、MapReduce）的并行计算程度[5,10]。但是这种副本存放策略并没有深入考虑哪些数据经常需要被执行以及数据自身的复杂度情况。下面分两个方面来分析基于 HDFS 的视频智能分析存在的不足。

在文件系统中，总有那么一部分视频数据会要求经常用于智能分析，而有的时间段的视频数据则不会，例如地铁里凌晨的视频因为重要度不高，所以往往存储在文件系统中作为记录，并不会被拿来分析。将经常用来分析或者分析次数多的视频称为热点视频文件。

视频智能分析是一个极度消耗 CPU 和内存的计算任务，当热点文件集中在少数节点上执行的时候，少数节点会出现 CPU 和内存长时间处于峰值状态的情况，可以通过 ganglia 集群监控系统进行观察，如果出现上述情况，说明了系统存储着计算不均衡的状态。定义热点视频文件为统计该文件在过去的一周被用来分析的次数与所有视频文件分析总次数的除数，那么 i 文件的热度为：

$$Heat_i = \frac{C_i}{\sum_{i=1}^{N} C_i} \tag{1}$$

其中，C_i 为 i 文件一周内计算的次数，N 为总文件数。当热点视频存储在少数节点的时候，会严重影响系统的执行效率。基于数据本地性的任务调度方式，热点视频大部分任务会安排在存储的机器上进行执行，剩余的任务会安排到其他空闲节点上进行执行，那么空闲节点则需要通过远程访问视频文件的方法来获取数据，数据需要先经过网络传输到空闲节点所在的物理节点上，一方面增加了系统的网络 IO，影响了系统的性能，另一方面因为数据需要传输才能启动任务进行执行，空闲节点的 CPU 会出现长时间的等待状态，降低了系统的资源利用率。为了解决上述问题，需要根据实际情况进行优化。

在视频智能分析领域，相同的算法对于具有相同清晰度而内容不同的视频需要不同的计算资源。以机器视觉算法人脸识别为例，人脸识别技术大致分为 3 个过程，即人脸特征库建立、人脸检测、人脸识别[7,11]。而上述 3 个过程对单帧数据计算所需要的 CPU 计算量具有很大的影响。当视频环境一致（每帧图像包含人脸数量一致，图像的色彩度一致）时，人脸识别的计算量会随人脸特征库规模的增大而增加；当特征库不变时，人脸识别需要的计算量随每帧图像包含的人脸增多而需要更多的计算资源；当特征库一致、人脸数一致时，人脸识别需要的会随图像的黑暗程度不同而计算量有所变化。如图 2—图 4 所示，相同的摄像头下录制的视频中 1 张人脸、2 张人脸、4 张人脸所需要的计算资源是依次增加的，光照情况对计算量的影响也很大。

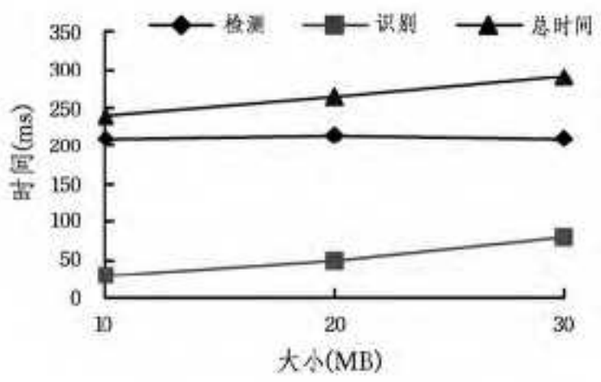

图 2　训练集大小对视频分析的影响

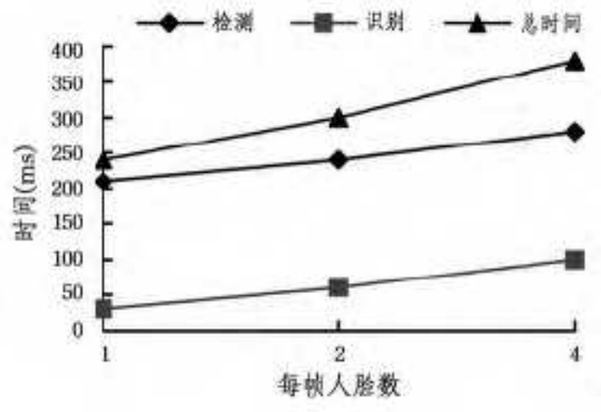

图 3　每帧人脸数对视频分析的影响

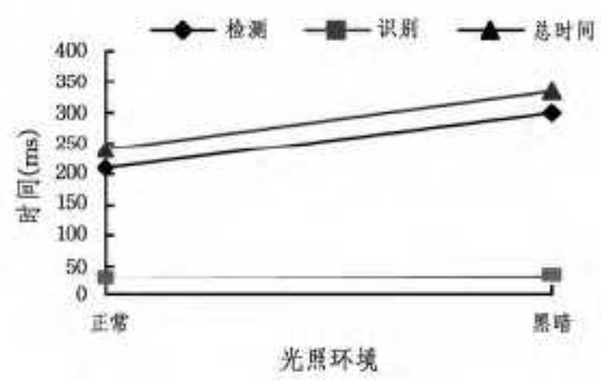

图 4　光照环境对视频分析的影响

3　设计与实现

3.1　数据重分布策略

为了提高系统的执行效率，达到计算负载均衡的目的最简单的方法就是按热度和复杂度对视频数据进行重新分布。重新分布的策略是将热度高的文件与热度低的文件存储在一起，计算复杂度高的与计算复杂度低的视频存储在一起。为了使问题更加清晰，针对副本数为 1 的情况进行分析，所得理论理论上对于副本数为多份的情况也是适用的。

假设一组视频文件在 HDFS 上按下述方式存放。如图 5 所示，在各个 Datanode 节点中，存放的小矩形代表着该文件的复杂度。Datanode1 中两个视频文件的复杂度和为 14，Datanode2 为 12，Datanode3 为 4，Datanode4 为 6。显然当需要进行视频智能分析的 8 个视频文件按如下进行数据存放时，系统的总执行时间取决于 Datanode1（仅考虑每台机器的执行能力是一致的），因为分布式系统的执行时间是由最后完成任务的节点的时间决定的，而该节点的视频数据需要的执行时间最长。

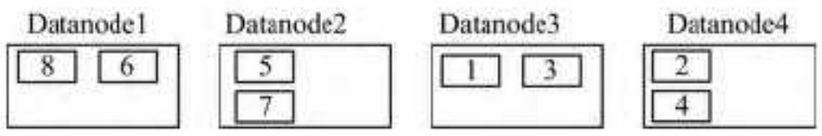

图 5　默认 HDFS 中的数据存放

考虑到视频的复杂度，重新对上述视频进行分布，如图 6 所示。通过交换一些数据的存放位置，不仅考虑到了 HDFS 数据的存放负载策略，同时还对视频处理本身所携带的特性——负载度予以考虑。这样对数据进行重新分布之后各个节点的负载度都一样，从而系统的执行时间都一样，比上述复杂度为 14 的节点任务计算时间节省了 5 个单位。

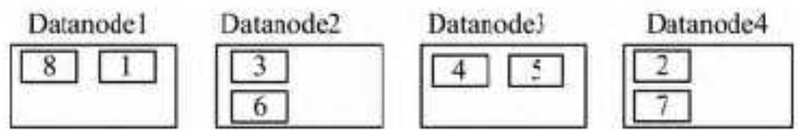

图 6　优化后的数据存放

再考虑视频的热度分布情况，暂且忽略视频的复杂度情况，如图 7 所示。

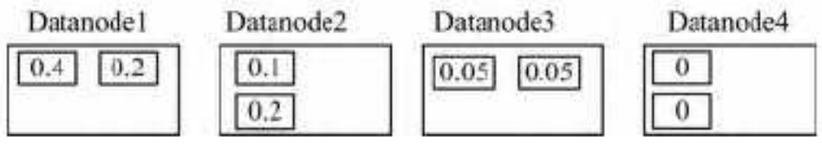

图 7　文件热度

Datanode1 中的视频文件热度最高，热度和为 0.6。而 Datanode4 中的视频文件则出现了并没有访问的情况，在这种情况下，如果能够通过数据的重新分布将热度高的视频文件与热度低的视频文件进行置换，那么系统的整个负载将趋于平衡。

上面分别对视频的热度以及视频的负载度进行了讨论，并分别给出了两者的数据重分布策略。下面将结合这两种情况进行分析，并给出数据的重分布策略。

由上述分析可知，每一个文件存在一个热度参数以及一个视频的复杂度参数，所以只需要简单地将上述参数进行加权求和，然后根据计算的结果对数据按照高负载与低负载进行置换的原则进行视频的置换即可。

视频的智能分析需要消耗大量计算机资源，而文件的置换需要在 HDFS 各个物理节点之间不停地传输文件，这需要消耗大量的磁盘和网络 IO 资源，会影响正在执行任务的性能。因此，我们在整个系统计算量不大的情况下，例如凌晨 1—4 点钟根据过去 5 天统计的视频热度和复杂度情况对视频进行重分布。下面将给出视频文件的复杂度计算方式、视频的热度统计以及如何根据上述统计信息进行文件的置换。

3.2　数据置换算法

由上述的分析得出：需要进行物体的检测和提取的算法，在视频中需要检测的物体越多，处理相应视频的时间也就越长。根据此特性，可以统计每个文件块的执行时间长短作为该视频的负载度，而将计算访问过的视频次数作为该文件的热度。根据上述的情况来对存储在 HDFS 上的数据进行重分布，能够提高上层并行计算框架的执行效率。

针对视频文件，没有好的方法能够预知该视频的复杂信息，也无法预知视频可能会被分析的次数。如果在数据上传到 HDFS 之前利用特定的检测视频复杂度的算法对视频进行处理，一方面破坏了视频的热度情况，一方面需要消耗大量的计算资源。视频分析本来就需要消耗大量的计算资源，如果增加这样一步预处理，那么相当于对每个视频文件需要处理一次，增加了系统的负担。因此，比较现实的方法就是利用历史数据处理情况作为数据重分布的依据。每次当用户对特定的视频文件进行采用相应的算法进行处理之后，我们在用户执行的算法中加入一些信息统计的方法，并将收集好的信息保存到 Hbase 中。

视频文件负载和热度信息收集如图 8 所示。首先利用 HDFS 提供的接口获取所有的视频文件名字信息，对没有存放在 Hbase 中的文件名即新存储到 HDFS 上的数据名，将该文件名作为 Key 值存储到 Hbase 中，并将热度值和复杂度赋值为零。之后开始执行计算任务，当该文件执行完成后，将该文件执行的时间在 Hbase 中进行更新，并将热度（即访问次

数)加1。当所有的任务执行完后,整个流程结束,HDFS中的视频负载信息也就获取完成。

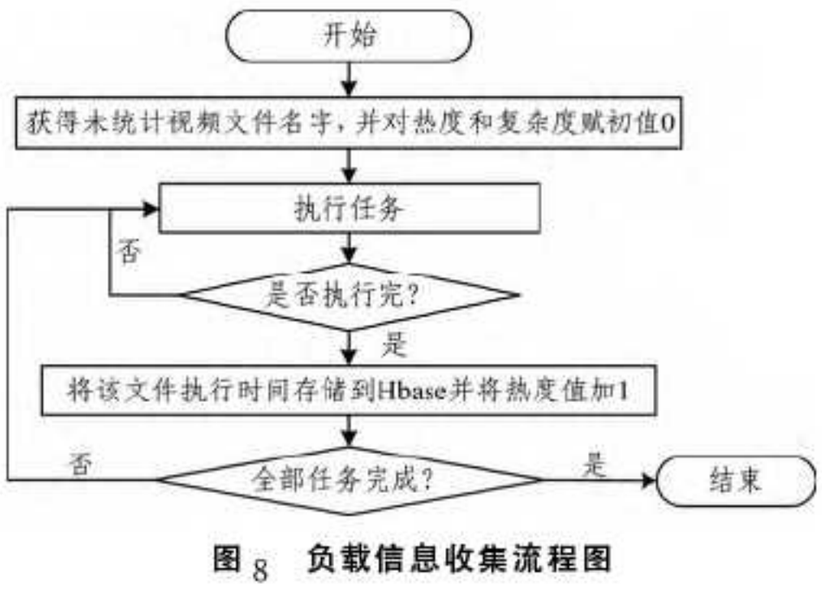

图8 负载信息收集流程图

由上面的分析可知,我们是定期地获取上述负载信息,然后进行数据的重新分布。为了和所在的物理节点挂钩,设视频文件的热度以及复杂度表示为:

$$Heat_{i,j}=\frac{C_{i,j}}{\sum_{i=1}^{N}\sum_{j=1}^{n}C_{i,j}} \tag{2}$$

$$CPX_{i,j}=\frac{T_{i,j}}{\sum_{i=1}^{N}\sum_{j=1}^{n}T_{i,j}} \tag{3}$$

其中,$C_{i,j}$表示在i机器上的第j个文件访问的次数,通过Hbase数据可以获得;N表示物理机器的个数,n表示各个节点上的文件数量;$Heat_{i,j}$表示i机器上的j文件的热度,可以反映该文件在接下来进行计算的可能性的大小;$T_{i,j}$表示在i机器上的第j个文件任务执行的时间;$CPX_{i,j}$表示i机器上的j文件智能分析所花费的时间比重,反映出该文件的需要的计算量,其中$\sum_{i}^{N}\sum_{j}^{n}Heat_{i,j}=1$且$\sum_{i}^{N}\sum_{j}^{n}CPX_{i,j}=1$。最后对热度和复杂度进行加权求和来表示文件的负载因子信息,公式如下:

$$Load_{i,j}=0.5*Heat_{i,j}+0.5*CPX_{i,j} \tag{4}$$

其中,$\sum_{i}^{N}\sum_{j}^{n}Load_{i,j}=1$。

根据上述分析,我们最终的目标是让各个节点的负载达到均衡的状态,也就是负载均衡状态。即当前节点的负载经过数据的重新分布之后,负载量在均值的10%左右浮动,使各节点满足下式。

$$\left|\sum_{j=1}^{n}Load_j-\frac{\sum_{i=1}^{N}\sum_{j=1}^{n}Load_{i,j}}{N}\right|\leqslant\frac{\sum_{i=1}^{N}\sum_{j=1}^{n}Load_{i,j}}{N}*10\% \tag{5}$$

其中,$Load_{i,j}$表示第i台机器上的j文件的负载度,N表示N个节点,上述公式表示各个节点中文件的负载度和减去集群的平均负载在平均负载的10%左右浮动。

算法 基于视频复杂度以及热度的文件置换算法

开始

1. 在Hbase中获得文件访问次数和执行时间;
2. 根据式(2)和式(3)分别求每个文件的热度和复杂度信息;
3. 根据式(4)求出每个文件的负载信息;
4. 统计每个物理节点的总负载情况;
5. 求出集群的均匀负载数据;
6. 对每个节点的负载进行排序;
7. 将负载高的数据文件与负载低的数据文件进行置换;
8. 当前节点的负载是否满足式(5)要求,如果满足回到步骤6,不满足则回到步骤7。

结束

4 实验

4.1 实验环境

工作节点分别由5台ThinkServer RD640服务器组成,每台配置CPU Xeon E5-2620 2.1GHz 24核处理器,内存DDR3 32GB,硬盘容量1TB,4个千兆以太网卡;主节点是联想T4900V商业台式机,配置CPU Intel酷睿i7 3.6GHz,内存DDR3 16GB,硬盘ITB;一台思科24口千兆交换机。

集群每台机器的软件配置环境包括操作系统Ubuntu 12.04 LTS,spark-1.0.2,JDK1.7,FFmpeg 2.1.5,OpenCV 2.4.8,ganglia3.6.0。详细的配置情况如表1、表2所列。实验使用的测试程序是opencv中的人脸检测和识别程序。

表1 主节点配置情况

硬件配置		操作系统	基础软件配置
CPU类型	Intel(R) Core(TM) i7-4790S CPU@3.60GHz	Ubuntu12.04 64	spark-1.0.2, JDK1.7, ganglia3.6.0的数据收集组件、监控组件、Web组件
CPU核数	8		
内存	16GB		
IP	192.168.1.93		

表2 从结点配置情况

硬件配置		操作系统	基础软件配置
CPU类型	Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10GHz	Ubuntu12.04 64	spark-1.0.2, JDK1.7, FFmpeg 2.1.5, OpenCV2.4.8, ganglia3.6.0的监控组件。
CPU核数	24		
内存	32GB		
IP	192.168.1.107—192.168.1.111		

4.2 实验结果

将多个视频文件存储在HDFS中,然后利用Spark分布式计算平台对这些视频文件进行人脸识别分析。首先不断地对相同的视频进行分析,让这些视频成为热点视频,然后观察每次执行的时间情况,然后仅考虑视频的复杂度的情况,记录每次的执行时间,最后同时考虑视频热度和视频复杂度的情况,记录每次的执行时间。实验结果如图9所示。

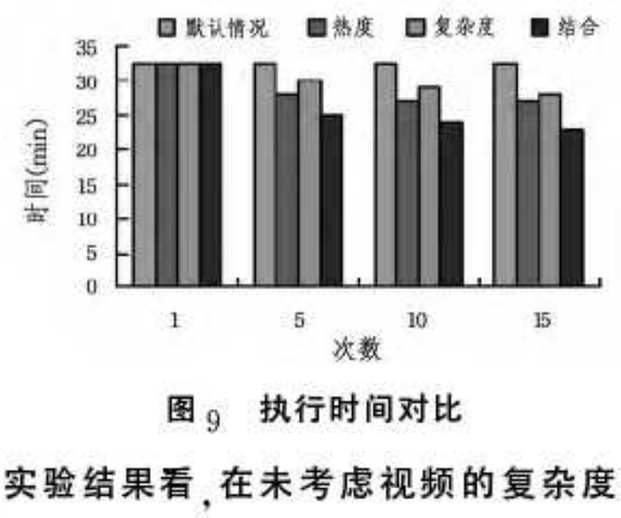

图9 执行时间对比

从图9实验结果看,在未考虑视频的复杂度和热度的情况下,40G视频大概需要32min才能执行完,而分别考虑视频的热度和复杂度的情况下,视频的执行时间有所减少;当综合考虑视频的热度和复杂度时,视频的执行时间是最少的。在现实中,我们往往会在夜间或者是系统负载小的情况下对数据进行置换,这样能够尽量小地影响系统的其他任务的执行效率。

下面从另外一个角度来分析该算法的效率问题。我们知道,当数据分布不均匀的时候,有的视频会很早地执行完毕,而有的视频则需要较长的执行时间,这样系统的CPU的整

体利用率是不一样的。当数据的负载不均匀时，会出现有的 CPU 处于空闲等待状态而有的 CPU 处于一直计算的状态。

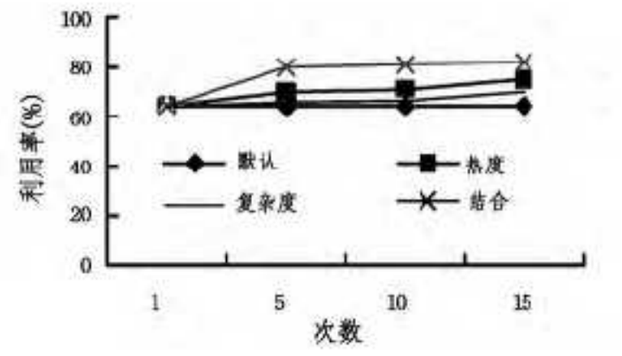

图 10 集群 CPU 利用率

图 10 显示了 4 种方式对相同的摄像头数据处理的情况下，集群 CPU 的利用率情况。本文实验通过周期性地取 ganglia 监控数据作为实验最终数据，如图所示，经过所设计的置换算法对数据进行置换置换，集群的整体 CPU 利用率提高了。因为数据更好地基于本地性的执行，不需要经过网络传输，所以能够更好地将任务及时分配给 CPU 计算，从而 CPU 的利用率提高了。因此上面 4 个部分的实验数据统计也正好验证了我们提出的置换算法可以更好地应用于大规模的实时视频流计算。

结束语 针对大规模的视频文件，我们一般利用 MapReduce 或者 Spark 来对其进行计算。为了提高数据的计算效率，使用 HDFS 来对数据进行存储，因为 HDFS 本身的数据存放机制能够让数据均衡地存放在集群中。但是视频文件自身较传统的文本文件、基于大小的块文件的计算消耗的资源往往不一样，甚至差异很大，这样上层分布式计算框架在对视频进行计算的时候，如果某个节点的视频数据负载度大，那么该节点的执行时间将比其他节点长，而系统的执行时间取决于最长任务的执行时间。采用有效的数据重分布算法能够很好地将热度高以及复杂度高的视频与热度低以及复杂度低的文件存储在一起，从而降低该节点的负载。实验结果表明，所提出的算法能够提高系统的计算效率，减少系统的计算总时间。

基于视频访问热度以及视频的复杂度能够在一定程度上解决大部分算法的负载均衡问题。但是本文所提算法是适用于需要进行物体的检测和提取的算法。而当有些算法不一定需要对物体进行检测以及提取的时候，视频的复杂度定义将不一样，未来将会继续探索这些算法及如何提高其计算效率。

参 考 文 献

[1] White T. Hadoop 权威指南[M]. 北京：清华大学出版社，2011：1-123

[2] Zaharia M, Chowdhury M, Das T, et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing[C]// Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. USENIX Association, 2012: 2-2

[3] Geng Chen-yao, et al. Distributed Video Processing Platform Based on Map Reduce[J]. Computer Engineering, 2012, 38(10): 280-283

[4] Zaharia M, Chowdhury M, Franklin M J, et al. Spark: cluster computing with working sets[C]// Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing. 2010: 1765-1773

[5] Zaharia M, Borthakur D, Sen Sarma J, et al. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling[C]// Proceedings of the 5th European Conference on Computer Systems. ACM, 2010: 265-278

[6] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113

[7] Lin S H. An introduction to face recognition technology[J]. Informing Science, 2000, 3(1): 1-8

[8] Spark job scheduling[OL]. http://spark.apache.org/ docs/latest/job-scheduling.htm

[9] Borthakur D. HDFS architecture guide. HADOOP APACHE PROJECT[OL]. (2008). http://hadoop.apache.org/common/docs/current/hdfs design.pdf, 2008

[10] Kapil B S, Kamath S S. Resource aware scheduling in Hadoop for heterogeneous workloads based on load estimation[C]// 2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT). IEEE, 2013: 1-5

[11] Orrite C, Bernues E, Gracia J J, et al. Face detection and recognition in a video sequence[C]// Defense and Security. International Society for Optics and Photonics, 2004: 94-105

[12] Bezerra A, Hernández P, Espinosa A, et al. Job scheduling for optimizing data locality in Hadoop clusters[C]// Proceedings of the 20th European MPI Users' Group Meeting. ACM, 2013: 271-276

(上接第 456 页)

否为聚类中心；小簇合并花费时间较短，但也存在一定问题，即小簇之间的边界并不一定在两个聚类中心的中点附近，下一步希望能够找到一种合理、简单的小簇合并方法。

参 考 文 献

[1] 于海涛，贾美娟，王慧强，等. 基于人工鱼群的优化 K-means 聚类算法[J]. 计算机科学，2012，39(12)：60-64

[2] 张建朋，陈福才，李邵梅，等. 基于密度与近邻传播的数据流聚类算法[J]. 自动化学报，2014，40(2)：277-288

[3] Lei Xiao-Feng，谢昆青，Lin Fan，等. 一种基于 K-Means 局部最优性的高效聚类算法[J]. 软件学报，2008，19(7)：1683-1692

[4] 张丽，崔卫东，邱保志，等. 基于划分与层次方法的混合聚类算法[J]. 计算机工程与应用，2010，46(16)：127-129

[5] 邱保志，陈本华，张真，等. 一种新的快速混合聚类算法[J]. 微电子学与计算机，2008，25(7)：78-80

[6] 李晓翠，孟凡荣，周勇，等. 一种基于代表点的快速聚类算法[J]. 南京大学学报(自然科学版)，2012，48(4)：504-512

[7] 马儒宁，王秀丽，丁军娣，等. 多层核心集凝聚算法[J]. 软件学报，2013(3)：490-506

[8] Rodriguez A, Laio A. Clustering by fast search and find of density peak[J]. Science, 2014(344): 1492-1496

[9] Clusteringdatasets [EB/OL]. (2008-04-15) [2015-03-05]. http://cs.joensuu.fi/sipu/datasets