MRKFE: Designing Cascaded Map Reduce Algorithm for Key Frame Extraction

Praveen Deshmane
B.V.B CET, Vidyanagar, Hubli - 580031, Karnataka, India
E-mail: deshmane.praveen@gmail.com Phone: 9538560939

***Abstract*—Key frame extraction is a technique used to compute measure of dissimilarity between successive frames in order to identify unique frames from a collection of video frames and thereby, eliminates redundant frames. It is widely used in image processing, video surveillance, cyber forensic, video browsing, and video indexing and retrieval applications. Though the state-of-the art computer vision algorithms provide efficient solutions for extraction of key frames, but often demands very high computational power as it involves computing very high dimensional image features. On the other hand, with the widespread use of Big Data and IOT applications, the storage, management and accessing of large volume, high velocity and variety of video data is increasing at exponential rates. Therefore, in order to address this problem we propose optimization of key frame extraction algorithms on Apache Hadoop distributed framework. As the process of extracting key-frames involves simultaneous execution of image processing tasks, the Hadoop framework divides and distributes these tasks to multiple nodes of the cluster. The MapReduce programming model has been redefined to perform the tasks of identifying key frames by color histogram difference between successive frames. The proposed optimized Hadoop's MapReduce algorithm has been tested on a data set of 1000 news videos each having approximately a length of 10 minutes. A performance speed up of 7 has been achieved by varying the size of the cluster from 1 node to 10 nodes.**

Keywords - Key Frame Extraction, Color Histogram, MapReduce, Hadoop

I. INTRODUCTION

Key frame extraction is a technique used to discover key frames and eliminate duplicate frames from a collection of video frames. It is widely used in image processing, video surveillance, cyber forensic, video browsing, video summarization and video indexing and retrieval applications. Due to tremendous growth of mobile internet users, social medias such as youtube, facebook, instagram users, communication channels such as TV and camera devices, the amount of video data produced by mankind is in terms of petabytes. For every minute 300 hours of videos are uploaded in youtube and 8 million video views per day in Facebook. To store, access and manage this big data is very challenging and complex. To save large amount of memory we need key frame extraction. With key frame extraction we can decrease the video data size by eliminating duplicate video frames and efficiently store in storage devices. Generally, a video is comprises of 20 to 30 frames for each second. Thus, a 30 minutes video would contain around 20x30x60 to 30x30x60 frames.

Key frame is a most informative frame that provides more meaningful information present in the video. Many researchers have tried to extract the key frame in a video with various techniques. Some of the techniques include color histogram, motion information, edge change ratio and frame correlation etc. Key frame extraction technique needs very high computational power as it involves in computing very high dimensional image features.

In previous work, Zhong Qu et al. [7] introduce an improved key frame extraction method that is based on HSV color space. The method regards "take the first frame (or the first frame and last frame) as the key-frame of the shot, so it is easy to implement, but the result is not reliable; second, extract key-frames according to the distances between every adjacent two frames. If one distance is above the threshold preset by users, then it's considered that there appeared a new key-frame; third, extract key-frames according to motion analysis". Guozhu et al. [5] introduce Key Frame Extraction from MPEG Video Stream. The method regards "Firstly, an improved histogram matching method is used for video segmentation. Secondly, the key frames are extracted utilizing the features of I-frame, P-frame and B-frame for each sub-lens. Fidelity and compression ratio are used to measure the validity of the method". Sheena C V et al. [9] introduce key-frame extraction using threshold of absolute difference of histogram of consecutive frames of video data. The method regards "to compute the key-frames using simple calculations of histogram of absolute difference of consecutive frames in video data". Fei, Mengjuan and Jiang et al. [4] introduce a new fusional framework combining sparse selection and clustering for key frame extraction. "The proposed framework overcomes issues such as information redundancy and computational complexity that afflict conventional SS methods by first obtaining candidate key frames instead of accurate key frames". Gavin L. Moir et al. [6] introduce an efficient method of key-frame extraction based on a cluster algorithm. The method regards "to investigate the effects of different configurations of repetitions within a set of deadlifts on the mechanical variables of concentric force, concentric time under tension, impulse, work, power, and fatigue". Ran Zheng et al. [10] introduce parallel key frame extraction for surveillance video service in a smart city. The method regards "to extract key frames from traffic surveillance videos based on GPU (graphics processing units) to ensure high efficiency and accuracy".

978-1-5090-6135-8/17/$31.00 ©2017 IEEE

Sentinelli et al. [8] introduce live key frame extraction in user generated content scenarios for embedded mobile platforms. The method regards "Key Frame Extraction solutions in embedded architectures for mobile platforms. In particular, in our scenario the list of key frames is requested right after a shooting session ends. The most interesting outcome has been the evaluation of performances in User Generated Content scenarios through an extensive survey based on Opinion Scores, interviews and other minor metrics testing five different solutions".

Several mechanism and methods have been proposed to extract key frames. The main drawback to most of these approaches may lack in computation time. Parallel Key frame extraction on GPU [8] is having synchronization and memory problems. Considering with respect to big data all these approaches may fail because most of the methods run on single machine. With the widespread use of Big Data and IOT applications, the storage, management and accessing of large volume, high velocity and variety of video data is increasing at exponential rates. To extract key frames for millions of videos, there is a requirement for time optimized key frame extraction calculation that processes large datasets very quickly on a scalable, fault tolerant, cost effective and distributed platform.

In this paper, we present time optimized cascaded MapReduce algorithm for extracting key frames for a large collection of videos. The MRKFE algorithm calculates and finds key frames on Hadoop's scalable and distributed platform. In particular, we used color histogram [3] based key frame extraction technique 1) first we store a large collection of video frames in to Hadoop Distributed File System (HDFS) 2) cascaded MapReduce blocks perform the computation of color histograms for each of the video frames and calculates the histogram difference between consecutive frames using Euclidean distance 3) threshold has been set by considering mean and standard deviations 4) key frames are extracted by comparing histogram difference with the threshold value 5) parallelism is involved in the process of computing histograms and their similarities is exploited. For calculating color histogram, finding histogram differences, to find mean and standard deviation to compute threshold T and to compare histogram difference with T we have designed a MapReduce algorithm and cascaded these MapReduce algorithms in light of the fact that, each MapReduce calculation reliant on the consequence of past MapReduce calculation aside from the first MapReduce. This whole MapReduce calculation cannot be executed parallel.

The proposed algorithm can speed up in performance with respect to time efficiency by distributing the video frames across the cluster of computers of Hadoop's distributed platform and processing on each of these computers through which achieves high parallelism. To setup hadoop cluster we need commodity of computes which are cost effective. There are no requirements for specialized hardware and server. As we go on increase the cluster size from 1 node to 10 nodes, there is a huge performance speed up in MRKFE computation time. Even one or more nodes goes down, the jobs which were assigned to failed nodes are re-assigned to other nodes that are up and running. Thus Hadoop provides high fault tolerance and highly interactive job submission. This experiment has been conducted on data set of 1000 news videos each having approximately a length of 10 minutes.

In the following section, we explain about Apache Hadoop, MapReduce framework and HDFS, in third section the implementation of MRKFE algorithm. In the fourth section we present experimental results followed by conclusion and future work.

II. APACHE HADOOP, MAPREDUCE FRAMEWORK AND HDFS

In this section, we look into Apache Hadoop, MapReduce Framework and HDFS

A. Apache Hadoop

Doug Cutting and Mike Cafarella created open source project called Hadoop in the year 2005. Hadoop is broadly utilized as a part of the territory of stock trade, power grid, online networking, transport, internet searcher applications and information distribution center. Using hadoop very large data sets can be distributively processed and stored on cluster of computers. Each and every one of the modules in hadoop is outlined with a key presumption that hardware failures are normal and ought to be consequently taken care of by the hadoop. Apache Hadoop consists two major core parts 1) for distributed processing model called MapReduce 2) for distributed storage called hadoop distributed file system (HDFS). Hadoop parts files into extensive pieces and circulates them crosswise over nodes in a cluster. To process information, Hadoop exchanges bundled code for nodes to process in parallel in view of the information that should be handled. This methodology exploits data locality to permit the data set to be process quicker and more proficiently [1].

B. MapReduce Framework

MapReduce is a software framework for effectively writing applications which process limitless measures of data set (petabytes) in-parallel on vast cluster (a large number of computers). A MapReduce jobs parts the input data set into fixed size chunks which are handled by the map task in a totally parallel way. The framework sorts the yields of the maps, which are then contribution to the reduce task. Normally both the input and the yield of the job are put away in a file system. The framework deals with planning tasks, checking them and re-executes the fizzled tasks. Framework works on <key, value>pairs i.e. the framework looks input to the job as an arrangement of <key, value> pairs and produces an arrangement of <key, value> pairs as the yield of the job [1].

C. Hadoop Distributed File System (HDFS)

HDFS was created utilizing distributed file system plan. It is reasonable for distributed storage and processing. Hadoop

978-1-5090-6135-8/17/$31.00 ©2017 IEEE

gives a command line interface and java API [1] to interact with HDFS. HDFS holds substantial measure of information and gives simpler access. To store such colossal data, the files are put away over different machines. These files are put away in excess style to safeguard the system from conceivable data misfortunes in the event of failure. HDFS additionally makes applications accessible to parallel preparing. The inherent servers of namenode and datanode help clients to effortlessly check the status of cluster. HDFS gives file authorizations and authentication [2].

III. MRKFE (MAPREDUCE KEY FRAME EXTRACTION)

In this section, we explain of MRKFE algorithm

A. MRKFE

Normally, when we convert video to frames we will get 20 to 30 frames per second. For example if we have 10 seconds of video then we will get 200 to 300 frames. When we have converted video to frames we have got 25 frames per second. These 25 frames are redundant frames which looks almost similar. We have considered 13th frame i.e. middle frame in one second video as sample frame.

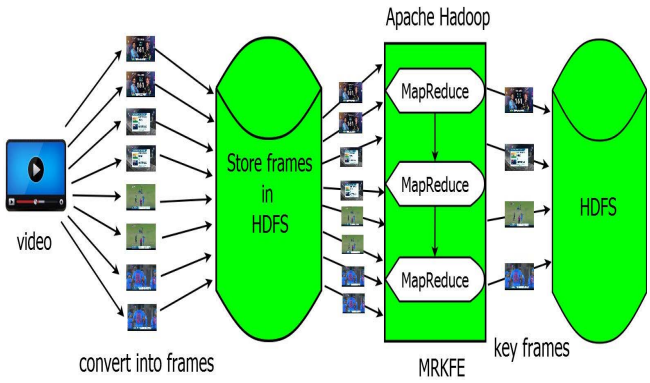

Figure 1. Key Frame Extraction Process

In this algorithm, all sample frames are stored in HDFS for key frame extraction. For each of these video frames color histograms are calculated first. Then histogram differences between consecutive frames are calculated. Then mean and standard deviation are calculated for all of these histogram differences. The mean and standard deviation are summed up and taken it as threshold value. If each histogram difference is greater than the threshold value then the frame is considered as key frame or else discarded. The figure 1 shows the key frame extraction process. In this algorithm, first frame is considered as key frame. Then algorithm applied on other frames that are stored in HDFS. The algorithm 1 contains the steps for extracting key frames.

Algorithm 1 Steps for Key Frame Extraction

1: take frames one by one
2: calculate color histogram for each frame
3: calculate the histogram difference between consecutive frames using Euclidean distance
4: calculate mean and standard deviation of absolute difference
5: compute threshold T = mean + standard deviation
6: compare the difference with T. If the difference is greater than T select it as a key frame else discard.

Our cascaded MRKFE algorithm contains three MapReduce algorithms for all the above steps. First MapReducer contains only Mapper for calculating color histogram, no reducer. Second MapReduce for calculating histogram difference. Third MapReduce for calculating mean and standard deviation, to compute threshold T and for selecting key frames. The algorithm 2 calculates the color histogram for each frame. A color histogram represents to the dispersion of color in an image. The color histogram can be ascertained for any sort of color space, as HSV or RGB. A histogram of a picture is delivered first by discretization of the color in the picture into various bins, and tallying the quantity of picture pixels in every bin. The algorithm 2 utilizes four bins of RGB color space for ascertaining histogram. Bin 0 compares to intensities 0-63, bin 1 is 64-127, bin 2 is 128-191, and bin 3 is 192-255. The algorithm 3 and 4 calculates histogram differences between every consecutive frame. There are several methods available for calculating histogram difference. But here Euclidean distance is used to calculate histogram differences.

$$D(h1, h2) = \sqrt{\Sigma i(h1\ (i)\ - h2\ (i))^2}$$

Here, h1 and h2 are histograms of frame1 and frame2, i is each pixel count in the histogram. Similarly, the algorithm 5 and 6 finds the key frames. To find the key frames we need to calculate mean and standard deviation of all the histogram differences. To calculate mean,

$$Mean = \frac{D1+D2+D3+\ldots+Dn}{total\ number\ of\ histogram\ differences}$$

and to calculate standard deviation first calculates the deviation of each difference from mean and square the result of each.

$$Vi = (Di - Mean)^2 \ \ldots \ Vn = (Dn - Mean)^2$$

The variance is the mean of these values and standard deviation is equal to the square root of the variance.

$$Variance = \frac{V1+V2+V3+\ldots\ldots+Vn}{total\ number\ of\ histogram\ differences}$$

$$Standard\ Deviation = \sqrt{variance}$$

After finding mean and standard deviation we need to calculate the threshold value.

Threshold T = Mean + Standard Deviation

Now compare each histogram difference with threshold value T. If it is greater than T then select it as key frame otherwise discard the frame. For example, if histogram difference between h1 and h2 is greater than T then select h2 as the key frame.

IV. EXPERIMENTAL RESULTS

In this section, we test the execution of computing MRKFE on pseudo distributed mode and fully distributed Hadoop cluster. We have considered time as a parameter for measuring the efficiency of pseudo and fully distributed Hadoop cluster. To start with, we test the execution of the MRKFE algorithm on small cluster. Later, we expand the cluster size to check the performance of the algorithm on different size cluster. For every cluster with size n, we repeat the experiment for 5 times. And we note down the average execution time. This time is considered as the time taken to process video data-sets. The same MRKFE algorithm can be executed on both the clusters, there is no separate implementation.

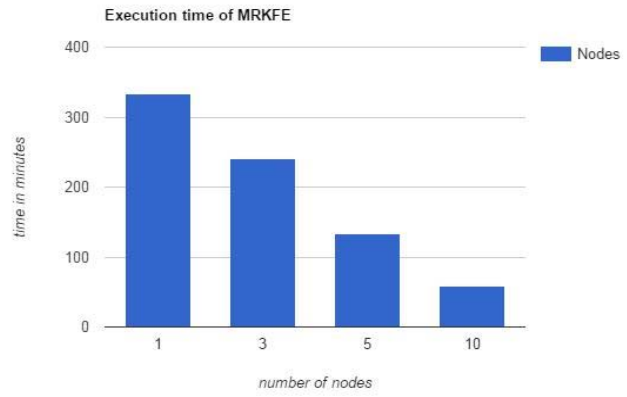

Figure 2. Execution time of MRKFE on different size cluster

For smaller video data-set, pseudo distributed mode and fully distributed Hadoop cluster setup will give practically comparable results. As we increase data-set size, fully distributed cluster will take less time to execute the tasks when contrasted with pseudo distributed or cluster with little size. After converting 1000 videos, each having approximately a length of 10 minutes, to frames we got 1, 50, 00000 frames.

After taking sample frames from each video we got 6, 00, 000 frames which are then stored in HDFS for process. The MRKFE algorithm applied on these frames. We got 599 histogram differences for each video. Finally we got key frames which depend on the video. We can see from the Figure 2 is that, as we expand the cluster size, we can find an incredible change in the velocity of execution of the algorithm. As size of the videos are exceptionally gigantic and number of videos, running computer vision algorithms on such data-sets often makes memory deficiency issues. In any case, porting of such algorithms on Hadoop framework can take care of the issue of memory deficiency to a more prominent degree. Since, Hadoop framework divides and disperses the tasks to different nodes of the cluster.

V. CONCLUSION AND FUTURE WORK

In this paper, we have examined the need of efficient computing of MRKFE algorithm. We have exhibited how MRKFE algorithm is designed using cascaded MapReduce on Hadoop framework. Composing MRKFE algorithm utilizing Hadoop MapReduce framework can give extraordinary change in time optimization. Ordinarily by distributing the work and data-set on fully distributed cluster. Our experimental results demonstrate the adequacy of embracing such a novel, cost effective and fault tolerant distributed system in solving MRKFE algorithm. A performance speed up of 7 has been gained by varying the cluster size from 1 to 10 nodes. In the future work we can write the same MRKFE algorithm on Apache Flink for more noteworthy velocity.

APPENDIX

Algorithm 2 Key Frame Extraction - MRColourHistogram Mapper

Require: videoname and framenames
Ensure: emit videoname-framename, histogram
1: function MAP(key; value)
2: get the name of the frame(image)
3: get height and width of the frame
4: histogram = get RGB values of each pixel using height and width and count
5: emit<videoname-framename, histogram>

Algorithm 3 Key Frame Extraction - MRHistogramDifference Mapper

Require: videoname-framename, histogram
Ensure: emit videoname, framename-histogram
1: function MAP(key; value)
2: line = value.toString()
3: tokenize the line
4: while (tokenizer.hasMoreTokens()) do
5: videoName = tokenizer.nextToken()
6: histogram = tokenizer.nextToken()
7: video = videoName.split("-")[0]
8: frame = videoName.split("-")[1]
9: emit<video, frame-histogram>

Algorithm 4 Key Frame Extraction - MRHistogramDifference Reducer

```
Require: video, frame-histogram
Ensure: emit video-frame-nextFrame, difference
result = 0, sqrt = 0
frameHisto //initialize HashMap<String,String>
function REDUCE(key; values)
    for each val in values do
        value = val.toString()
        tokenize the value
        while (tokenizer.hasMoreTokens()) do
            frame = tokenizer.nextToken()
            histogram = tokenizer.nextToken()
            frameHisto.put(frame,histogram)
    no-of-frame = frameHisto.size()
    hashA = null // ArrayList of Double
    hashB = null // ArrayList of Double
    for (i=1;i<=no-of-frame-1;i++) do
        hashA //initialize ArrayList of Double
        hashB //initialize ArrayList of Double
        first = frameHisto.get(i)
        tokenize the first
        while (tokenizer.hasMoreTokens()) do
            rgbcount = tokenizer.nextToken()
            tokenize the rgbcount
            while (tokenizer1.hasMoreTokens()) do
                tokenizer1.nextToken()
                tokenizer1.nextToken()
                tokenizer1.nextToken()
                hashA.add(tokenizer1.nextToken())
        second = frameHisto.get(i+1)
        tokenize the second
        while (tokenizer2.hasMoreTokens()) do
            rgbcount = tokenizer.nextToken()
            tokenize the rgbcount
            while (tokenizer3.hasMoreTokens()) do
                tokenizer3.nextToken()
                tokenizer3.nextToken()
                tokenizer3.nextToken()
                hashB.add(tokenizer3.nextToken())
        for (j=0;j<=63;j++) do
            a = hashA.get(j)
            b = hashB.get(j)
            c = a - b
            d = c * c
            result = d + d
        difference = (long) Math.sqrt(result)
        emit<key-frame(i)-frame(i+1), difference>
        result=0
        sqrt=0
        hashA = null
        hashB = null
```

Algorithm 5 KeyFrame Extraction – MRMeanAndKeyFrame Mapper

```
Require: video-frame(i)-frame(i+1),difference
Ensure: emit video,FrameDiff-difference
function MAP(key; value)
    line = convert value to string
    tokenize line
    while (tokenizer.hasMoreTokens()) do
        videoAndFrameDiff = tokenizer.nextToken()
        histoDifference = tokenizer.nextToken()
        video = videoAndFrameDiff.split("-")[0]
        FrameDiff = videoAndFrameDiff.split("-")[1]
    emit<video, FrameDiff-difference>
```

Algorithm 6 Key Frame Extraction - MRMeanAndKeyFrame Reducer

```
Require: video,FrameDiff-difference
Ensure: emit video, frame
mean = 0, stdDeviation = 0, threshold = 0
histogramDiff //ArrayList of Double
frameDiff //ArrayList of String
sum = 0, add = 0, histDiff = 0, no-of-difference = 0
function REDUCE(key; values)
    for each val in values do
        value = val.toString()
        tokenize the value
        while (tokenizer.hasMoreTokens()) do
            frames = tokenizer.nextToken()
            histDiff = tokenizer.nextToken()
            frameHisto.put(frame,histogram)
            sum = sum + histDiff
            histogramDiff.add(histDiff)
            frameDiff.add(frames)
        no-of-difference = no-of-difference + 1
    mean = sum / no-of-difference
    a=0, temp=0
    for i = 0 to less than histogramDiff.size() do
        temp = histogramDiff.get(i) - mean
        a = temp * temp; add = add + a
    stdDeviation = Math.sqrt(add / no-of-difference)
    threshold = mean + stdDeviation
    cmp = 0, tempkey = null
    for i = 0 to less than histogramDiff.size() do
        cmp = histogramDiff.get(i)
        tempkey = frameDiff.get(i)
        frame = tempkey.split("-")[1]
        if cmp is greater than threshold then
            emit <key, frame>
        else
            discard the frame
    emit <key, firstframe>
    histogramDiff = null, frameDiff = null
```

978-1-5090-6135-8/17/$31.00 ©2017 IEEE

REFERENCES

[1] Apache hadoop. http://hadoop.apache.org/docs/r2.6.1/. Ac-cessed: 2016-03-02.

[2] Lam Chuck. Hadoop in action, 2011.

[3] Naveed Ejaz, Tayyab Bin Tariq, and Sung Wook Baik. Adap-tive key frame extraction for video summarization using an aggregation mechanism. Journal of Visual Communication and Image Representation, 23(7):1031–1040, 2012.

[4] Mengjuan Fei, Wei Jiang, Weijie Mao, and Zhendong Song. New fusional framework combining sparse selection and clustering for key frame extraction. IET Computer Vision, 2016.

[5] Guozhu Liu and Junming Zhao. Key frame extraction from mpeg video stream. In Information Processing (ISIP), 2010 Third International Symposium on, pages 423–427. IEEE, 2010.

[6] Gavin L Moir, Bruce W Graham, Shala E Davis, John J Guers, and Chad A Witmer. An efficient method of key-frame extraction based on a cluster algorithm. Journal of Human Kinetics, 39(1):15–23, 2013.

[7] Zhong Qu, Lidan Lin, Tengfei Gao, and Yongkun Wang. An improved keyframe extraction method based on hsv colour space. Journal of Software, 8(7):1751–1758, 2013.

[8] Alexandro Sentinelli and Luca Celetto. Live key frame extraction in user generated content scenarios for embedded mobile platforms. In International Conference on Multimedia Modeling, pages 291–302. Springer, 2014.

[9] CV Sheena and NK Narayanan. Key-frame extraction by analysis of histograms of video frames using statistical meth-ods. Procedia Computer Science, 70:36–40, 2015.

[10] Ran Zheng, Chuanwei Yao, Hai Jin, Lei Zhu, Qin Zhang, and Wei Deng. Parallel key frame extraction for surveillance video service in a smart city. PloS one, 10(8):e0135694, 2015.

978-1-5090-6135-8/17/$31.00 ©2017 IEEE