A Motion Accuracy Evaluator based on Body Parts Movement by MapReduce Video Processing

Chonho Lee[1], Yu Terada[2], Yi Liu[1], Bu-Sung Lee[1]
[1]Nanyang Technological University, Singapore
[2]Osaka University, Japan

***Abstract*— This paper presents a healthcare application that tracks body parts movement in video recording persons who exercise, analyzes the motor performance (e.g., motion speed and space), and evaluates fitness status (e.g., motion accuracy and abnormality). We design a MapReduce video processing for collecting the data of body parts movement from a large number of video files and successfully shorten the execution time. Analyzing the video files, the application correctly detects persons who have difficulties in their motion speed and space.**

***Index Terms*— Abnormal motion detection, MapReduce**

I. Introduction

Integration of healthcare and IT gets a lot of attentions from not only medical field but general public. The basic purpose of the integration is to automatically monitor the status of humans and environment, diagnose disease and injury, and provide appropriate assistance to them. Such automation in an acceptable accuracy will help doctors to reduce their workload and to give quick treatment.

In recent aging society, motor skill decline is one of the serious problems for people such as the elderly and patients with Parkinson's disease. Medical doctors normally suggest people do regular exercise and/or receive proper rehabilitation to maintain the strength, flexibility, and balance [1]. Several research attempt to monitor the people's behavior using video camera [2] and body sensors [3] to recognize their activities and/or detect abnormal actions.

However, it is hard for doctors/trainers/assistants to handle all of the individual's motor performance due to the size of records or data. Besides, putting body sensors is cumbersome for some of the elderly and patients. The application to automate the collection of motor performance records and the evaluation of motion accuracy is desired.

This paper presents an application that tracks body parts movement from video recording persons who exercise, analyzes the motor performance (e.g., motion speed and space), and evaluates fitness status (e.g., motion accuracy and abnormality). In this paper, we focus on a case of arm exercise.

To develop the application, there are some efficiency issues in video file processing when dealing with a large number of persons and a large scale of video files. To speed up the process, we design a MapReduce video processing mechanism on hadoop cluster. For body parts detection, we utilize a Haar-like-feature-based detection algorithm [4], well known as a simple, fast object detection algorithm, implemented in *opencv*.

The contribution of the paper is as follows.

- We propose a MapReduce video processing mechanism to speed up the collection of body parts movement data from a large number/scale of video files. We verify that the execution time is shorten to 20-25% by combining the input video files.
- We develop a healthcare application that analyzes the motor performance of persons who exercise and evaluates the fitness status such as motion accuracy and abnormality. We successfully detects persons who have faced some difficulties to follow the instructions in terms of the motion speed and space of body parts.

II. Dataset

This paper focuses on video files that record persons doing arm exercise by following instructions on a screen. Fig. 1 shows some snapshots of video instructions for the exercise. The instructions include up-down (UD), left-right (LR), and back-forth (BF) exercises at two different speeds, slow (S) and fast (F). Video camera records persons from their left side at 45degree and about 2m distance.

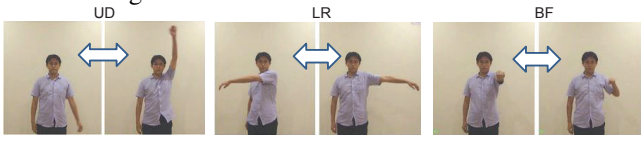

Fig. 1. Snapshots of instructions for the exercise, UD, LR, BF

III. Body Parts Detection in Video

This section describes how we process the video files to collect tracking data of the body parts, e.g., hand and face. Fig. 2 illustrates the data processing overview.

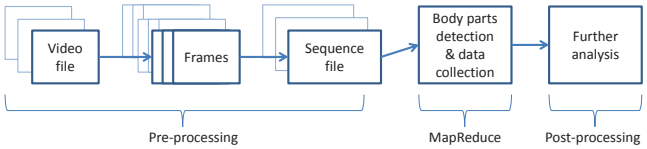

Fig. 2. Data processing overview

First, we split the video into many frames (at 30fps frame rate) and convert those frames into the sequence files. Then, we input the sequence files to hadoop system to detect body parts in those frames (i.e., images). MapReduce framework automatically classifies persons and their body parts into

978-1-4799-2131-7/14/$31.00 ©2014 IEEE

different groups and then collect each person's tracking data for further analysis.

Before explaining the design and implementation details of the MapReduce (Subsection B), we introduce how we detect body parts in images (Subsection A).

A. Haar-like-feature-based algorithm

A variety of techniques have successfully detected body parts (e.g., face, hands, legs, etc.) of human in images. There are two basic approaches for the object detection in images, which are ones using global image features (e.g. histograms of image hue, saturation, and intensity) and local image features (e.g., lines, contours of the particular region within images), respectively.

In this paper, we have applied an adaptive boosting (AdaBoost) algorithm based on Haar-like features (i.e., one of the local features) and generated a classifier consisting of multiple weak-classifiers in cascades.

Haar-like features have scalar values that represent differences in average intensities between two rectangular regions. They capture the intensity gradient at different locations, spatial frequencies and directions by changing the position, size, and shape of rectangular regions according to the base resolution of the detector or image patch. For example, when the resolution is 24-by-24 pixels, over 10,000 features are generated from the base feature sets (a) to (c) shown in Fig. 3. For each feature, a difference in average intensities is calculated by Equation (1) from all training sample images. R_1 and R_2 respectively indicate black and white regions, and $I(\cdot)$ indicates the average intensity of the region.

$$D(R_1, R_2) = I(R_1) - I(R_2) \quad (1)$$

To generate a classifier F in Equation (2), weak classifiers, $H_1, \ldots, H_T$, are selected through T iterations. At each iteration, a threshold is obtained for each feature i to minimize the classification error over training image samples including positive (containing target objects) and negative samples as shown in Fig. 4. The feature (and the corresponding threshold, $threshold_t$) whose error is the smallest is selected as a weak classifier among all features. After each iteration, AdaBoost adaptively updates weights on training samples, whose values are initially set with a uniform distribution, and it increases the weight values for the samples that were misclassified.

A classifier F is constructed as a linear combination of weak classifiers. Weights to the selected weak classifiers, W_t, are calculated based on the classification error. Note that these weights are different from the sample weights mentioned above. Finally, a series of weak classifiers are applied to every sub-window within testing images.

$$F = sign(W_1H_1 + W_2H_2 + \cdots + W_TH_T)$$
$$where\ H_t = +1\ if\ D_t > threshold_t; -1\ otherwise. \quad (2)$$

We prepared 500 positive samples (containing target objects) and 500 negative samples for training, and we used a Haar-like feature xml generator [5] to build a classifier after 20 iterations.

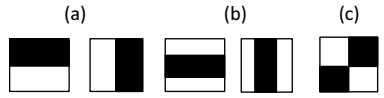

Fig. 3. A base set of Haar-like features

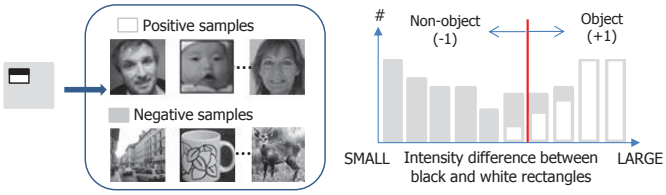

Fig. 4. An example of threshold determination for a feature

B. MapReduce-based video processing

MapReduce is a programming framework for parallel distributed processing on Hadoop. An execution of MapReduce consists of consecutive three phases: map, shuffle, and reduce phases as illustrated in Fig. 5. In each phase, multiple tasks are executed in parallel. First, a distributed file system, HDFS, splits the input data into fixed size blocks, 64 MB by default. In the map phase, each map task processes the given block as users defined and outputs the result in the form of key-value pairs (KVP). The shuffle phase automatically classifies the KVP into groups by their keys and also sorts the KVP based on secondary key. In the reduce phase, each task processes a group of KVP as users defined.

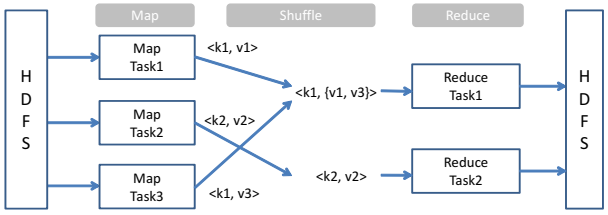

Fig. 5. Schematic diagram of MapReduce framework. In our design, Key = {PersonID, Parts, FrameNo} and Value = {Coordinates}

Map Method: HDFS assigns the input data (i.e., a set of frames or images) to map tasks automatically, and each map task processes the data in parallel. Since we cannot maintain the order of the images on Hadoop, we use frame numbers in the filename and the KVP. The main process in a map function is to detect body parts and emit the coordinates to reduce task. So, we specify the input format as the binary data using *setInputFormat* method at Job Class. By doing that, input key of map tasks is the filename of each image. Input value of map tasks is the binary data of each image.

Applying the Haar-like-feature-based detection algorithm into the binary data, we find coordinates of body parts. We assign the coordinates to output values of map tasks. Output key is composed of three factors, *PersonId*, *Parts* and *FrameNo*. Hadoop framework classifies the KVP based on PersonId (a primary key), and sort them based on a pair of Parts and FrameNo (a secondary key). As a result, each reduce task gets the coordinates of each person's detected body parts sorted by Parts and FrameNo.

Reduce Method: Reduce tasks receive the KVP aggregated by the PersonId and make a list of Coordinates for each person's Parts in order of FrameNo. Table I shows one of the examples that a reduce task outputs on HDFS.

TABLE I
AN EXAMPLE OUTPUT OF A REDUCE TASK

PersonId	Parts	Coordinates
1	hand	(10,25), (13,27), (12, 29), ...
1	Face	(15,45), (23,47), (22, 49), ...
2	hand	(12,28), (16,29), (22, 39), ...
2	Face	(23,40), (33,44), (32, 59), ...

Technical Issue: On Hadoop, each map task applies map function to each of input KVPs divided into fixed-size blocks. However, when considering a large number of small files, HDFS generates a large number of blocks without combining those small files to make the defined-size blocks. So, each tasktracker has to read and write many files on the disc, which causes slowness and ineffectiveness. To solve this problem, we combine image files and generate sequence files. If the size of each sequence file is smaller than block size, the number of sequence file is the same as the number of nodes; thus, we can utilize all map tasks. If the size of each sequence file is bigger than block size, then HDFS splits the file and assigns them to all map tasks. In either case, we can reduce the disc traffic and utilize all nodes effectively.

C. Experimental results

This subsection evaluates the MapReduce video processing in terms of execution speed and scalability. We compare the result when combining the input files in subsection III-C.1, and we show the results for different size of data at the different number of nodes in subsection III-C.2.

For performance characterization, we ran at most 6 slave nodes and 1 master node. All nodes had Intel(R) Xeon(R) CPU X5680, 3.33GHz, and used 33-Ubuntu, Hadoop framework 1.2.1, and Java 1.7.0. Table II shows the detail of input files we used, and Hadoop parameters were set as follows.

$mapred.child.java.opts =$ Xmx256m
$mapred.tasktracker.reduce.tasks.maximum = 1$
$dfs.block.size =$ 128MB
$dfs.replication = 3$

TABLE II
DETAIL OF INPUTS FILES

Input file	# of frames	Data size (MB)
inputs1	2000	104
inputs2	3500	161
inputs3	5000	218
inputs4	7000	322
inputs5	10500	483

1) Execution time: We compare the execution time when we combine input files and do not. Fig. 6 shows that the execution time "with combine" is shorten to 20-25% compared to "without combine". This indicates that, by converting input files to sequence files, we could reduce the number of map tasks. Table III supports the result by showing the number of launched map tasks with and without combining.

TABLE III
NUMBER OF LAUNCHED MAP TASKS

Input file	Launched map tasks with combine	Launched map tasks without combine
inputs1	6	2030
inputs2	6	3519
inputs3	6	5012
inputs4	6	7031
inputs5	6	10550

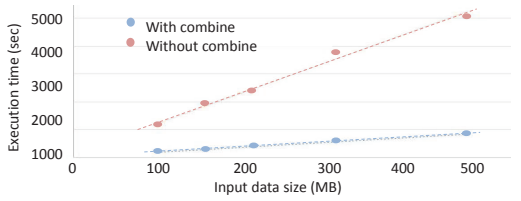

Fig. 6. Execution time when input files are combined or not

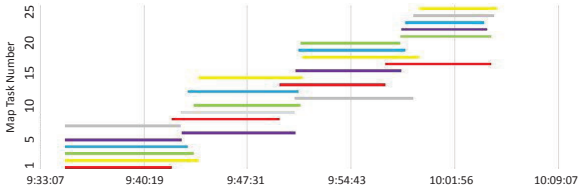

Fig. 7. Gantt chart of map tasks with combining input files using 6 nodes. Different colors indicate different map tasks.

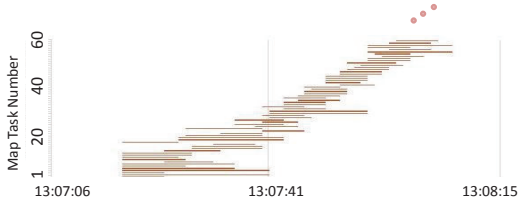

Fig. 8. Gantt chart of map tasks without combining input files.

For more detail, Fig. 7 and Fig. 8 show the Gantt chart of map tasks with and without combining, respectively. When combining input files, 6 map tasks effectively run in parallel, and the workload of each map task is almost equal. On the other hand, without combining, many map tasks are generated, and each map task runs in a short time. This causes slowness because each map task has to access the file on disc, and the disc traffic is increased.

2) Scalability: Fig. 9 shows the execution time in different numbers of nodes, 1, 2, 4 and 6. The line plot indicates the improvement rate of execution time to the time of a single node case. We verify that the video processing becomes faster as the number of nodes increases.

However, the bias of loads on reduce tasks sometimes lowers the scalability if the amount of input KVPs for reduce tasks vary at each reduce task. We utilize shuffle phase effectively and make the process of reduce tasks simple as described in Section III-B. Then, we observe that the loads of reduce tasks are equally distributed similar to map tasks. The run time of each reduce task was 28 seconds at most except shuffling time. So, even if the loads of reduce tasks are biased, the influence is very small. Thus, the scalability of our MapReduce design is high regardless of the data size.

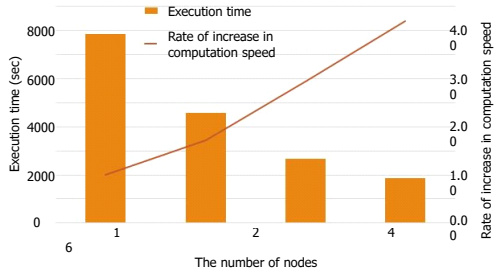

Fig. 9. Execution time in different number of nodes.

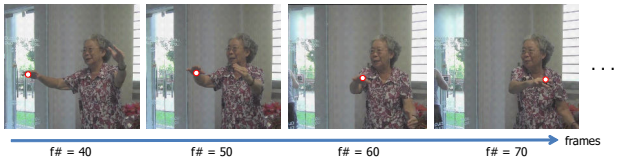

Fig. 10. Snapshots of hand's coordinates detected during LR exercise

IV. Motion Accuracy Evaluation

Based on the collected data, i.e., coordinates of the body parts (hand), we detect whether if persons correctly follow the instructions. Fig. 10 is example snapshots of the detected hand's coordinates. To evaluate the motion accuracy, we compute two features, motion speed and space from the coordinates. The motion speed is calculated based on the distance between coordinates at two consecutive frames, whose frame rate is 30 fps. Note that the distance is approximately transformed form the pixel coordinates and given camera angle (45 degree). The motion space is an angle from the line where arm horizontally stretched forth (for LR) or outward (for UD). In the case of LR (UD) exercise, the angle ranges from 90 degree toward one side, i.e., Left (Up), to 90 degree toward another side, i.e., Right (Down). No angle is considered for BF exercise.

Fig. 11 shows the hand speed and arm angle for instructions LR-S (the above) and LR-F (the bottom), respectively. The left and right movement is repeated 5 times as highlighted in green dotted boxes. Each pair of left and right is denoted by {LR-S1, LR-S2, ...} for LR-S, and {LR-F1, LR-F2, ...} for LR-F, respectively.

Fig. 12 is a line plot of the hand speed of a person (let's say Person A) who follows the instruction LR-F. Fig. 13 is a line plot of the arm angle of a different person (let's say Person B) who follows the same instruction LR-F. Computing the time to complete a pair of left and right movement, we observe that Person A has the delay (indicated by arrows) in his motion speed. Also, the average speed (173.51cm/s) is slower than instruction (263.91cm/s). It seems that Person A has a difficulty to follow the fast speed. On the other hand, Person B successfully keeps up the fast speed but cannot bring his arm to the left 90 degree (indicated by arrows). It seems that Person B has a limitation in his motion space.

V. Conclusion

We present a healthcare application to evaluate motion accuracy from video files. We design a MapReduce-based

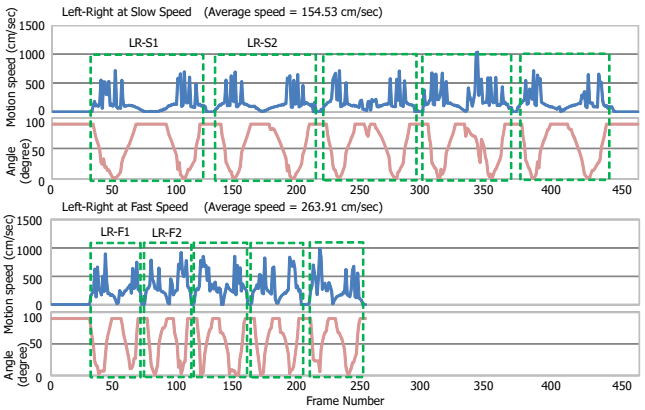

Fig. 11. Line plots of the hand speed (blue) and arm angle (red) of instructions LR-S (the above) and LR-F (the bottom).

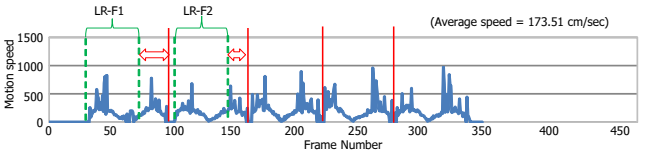

Fig. 12. Line plot of the hand speed of Person A who follows LR-F instruction. The arrow indicates the delay of arm movement.

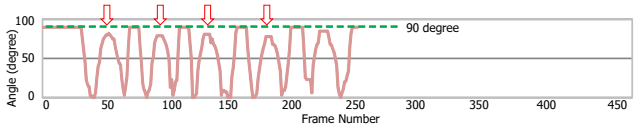

Fig. 13. Line plot of the arm angle of Person B who follows LR-F instruction. The arrow indicates the insufficient arm movement.

video processing for collecting the data of body parts movement from a large number of video files and successfully shorten the execution time. Analyzing the video recording persons who exercise, the application correctly detects persons who have difficulties in their motion speed and space.

In future, more samples, e.g., more persons, more instructions with different movements, and different body parts, will be collected as training data to improve the evaluation accuracy and to make the application general-purpose.

Acknowledgment

This work is a collaboration with the Joint NTU-UBC Research Centre of Excellence in Active Living for the Elderly (LILY). The authors thank Prof. Theng Yin Leng at WKW School of Communication and Information, NTU to provide video files.

References

[1] T. Crocker, A. Forster, J. Young, L. Brown, S. Ozer, J. Smith, J. Green, J. Hardy, E. Burns, E. Glidewell, and D. Greenwood, "Physical rehabilitation for older people in long-term care," *Cochrane Database of Systematic Reviews*, vol. 2, 2013.

[2] O. Brdiczka, M. Langet, J. Maisonnasse, and J. L. Crowley, "Detecting human behavior models from multimodal observation in a smart home," *IEEE Trans. on Automation Sci. and Eng.*, vol. 6, no. 4, 2009.

[3] D. Minnen, T. Starner, J. Ward, P. Lukowicz, and G. Troster, "Recognizing and discovering human actions from on-body sensor data," *IEEE Int'l Conference on Multimedia and Expo*, 2005.

[4] R. Lienhart and J. Maydt., "An extended set of haar-like features for rapid object detection," *IEEE Int'l Conf. on Image Processing*, 2002.

[5] A. A. Nyak, "How to make your own haar trained xml files." Robotics@Cyborg.