19th International Computer Science and Engineering Conference (ICSEC)
Chiang Mai, Thailand, 23-26 November, 2015

Parallel Random Forest with IPython Cluster

Wasit Limprasert
Department of Computer Science,
Faculty of Science and Technology,
Thammasat University, Pathumthani, Thailand
wasit@cs.tu.ac.th

***Abstract*—recently research studies require analytic tools capable to interpret patterns and find hidden knowledge from huge amount of data. Random Forest, an ensemble-tree classifier based on bagging method, is one of many well-known classifiers to find hidden model from data. The classifier has been applied to recognize various kind of data, e.g. human pose from depth images, plankton images and time-series pattern analysis. In this paper, an implementation of optimized parallel Random Forest has been designed and implemented on IPython, which is an interactive Python with parallelization functionalities and convenient to be deployed in most of computing platforms. The implementation shows 80% of CPU utilization when performing a training of 10^7 samples in 12hrs on EC2 cluster with 32 cores. This implementation shows capability to analyses large amount of data.**

Keywords—Parallel Algorithm, Random Forest, IPython

I. INTRODUCTION

Random forest is an ensemble-tree classifier based on bagging method introduced by Leo Breiman [1]. Random Forest is a well-known algorithm and applied to recognize many kind of data, e.g. human pose recognition from depth images [2], plankton silhouette images [3] and time-series pattern analysis[4]. The study of depth image recognition in [2] also showed modification of the splitting method by using two translation vectors as a splitting parameters to give more flexibility for generating generalized feature extraction, this introduces exponential growth of search space. Even the training time of random forest is longer than other methods such as SVM but the fast classifying speed and ability to avoid over fitting is very attractive. The algorithm is also proved to be very useful when a number of parameters or search space comparably large to a number of samples, e.g. Gene selection and classification [5]. Random forest creates multiple decision trees by finding optimized splitting parameter in every branch on the decision tree based on information theory similar to C4.5[6]. It creates discrimination boundary between classes before averaging all soft classifiers to construct a complete classifier. It is capable to find pseudo optimum parameters for the trees by randomly exploring the search space[7]. This makes the algorithm able to investigate high dimensional data and also makes an automatic feature selection from high dimension feature dataset.

Random Forest and similar algorithms use information entropy and maximizes information gain of each splitting and recursively perform splitting until reaching the terminating cases. One of many challenges of using Random Forest is that a process to construct a decision tree also known as "training" takes long computation time. For example a study of human pose recognition [2] using depth images in the training process with the following setup; 3 trees, 20 deep, 300k training images, 2000 training samples pixels per image, on a 1000 cores cluster took about 24 hours to finish. This shows the training need to be performed on a multi-cores processor or a cluster in order to accelerate the process.

According to work depth model as introduced in [8], the complexity of Random Forest algorithm is $O(N.logN)$ assuming the balance tree. The depth of algorithm is $logN$. The sequence of algorithm along its depth is unable to be accelerated by parallel algorithm. Therefore, most of studies aimed to generate many of narrow decision trees and then combining all trees from all processors to achieve ensemble learning. For example, a parallel Random Forest for R [8] showed significant speed-up ratio when calculating 8,192 trees for a dataset with 23292 genes and 62 cases from microarray on a High Performance Computing (HPC) system of 128 cores. However, in practical applications, most of the time a few number of trees is enough to classify complex a data set. In [2], a random forest constructed 3 trees of 20 depth resulted in saturated accuracy. The study showed that a few number of trees gave good result. There is very small accuracy gain because of increasing the number of trees when the accuracy already saturated. In general, there is a tradeoff between short training time and improvement of accuracy. A larger number of trees may be required for a larger sample size. This suggests that the few number of trees can give a good classification results. Mapping tree generating processes to cores without splitting training data can lead to duplication of redundant decision model in some computing nodes because all nodes have the same data to generate the decision model. An alternative idea [9] emphasis on histogram calculation, where some part of histogram calculation are performed on computing nodes and then sub-histograms are merged by using an approximation. The method removing duplication of redundant data and communication cores.

Due to a variety of parallel computing platform, there are many implementations on different platforms. For example in [10], MPI is used for implementing a C4.5 to generate decision tree. Recently, Map-Reduce parallelism is also applied to accelerate the process of calculating an ensemble tree classifier, e.g. in [12] used breadth first search and Map-Reduce to tackle marketing research problems. Other hardware platform such as

978-1-4673-7825-3/15/$31.00 ©2015 IEEE

GPU and FPGA [13] also possible but they may not suitable for large scale calculation because of smaller memory size. IPython Parallel [11] is an alternative distributed computing platform that allows user to interactively checking intermediate variables as similar as MATLAB. The most interesting capability of IPython Parallel is ability to spawn parallel jobs in a cluster. IPython Parallel can create a python instance to control all parallel jobs, this instance is called "controller" and also able to spawn many instances to be computing engines on a cluster. The controller can access local variables on engines by using IPython direct interface called "direct view". This allows users to debug both controller and engines effectively.

II. OUR METHOD

In this paper, the parallel Random Forest is implemented by using IPython[11], an interactive python with visualization tools and parallel computing functionality. IPython is able to launch a MPI-like parallel task and broadcasts a message over a HPC cluster using a few lines of code. The IPython code for cluster can be executed on a normal multi-processor computer allowing the same code to run on multiple platforms. This is very convenient for intermediate programmer to start the parallel programing and there is no requirement of a huge HPC in development stage. IPython provides a parallel programing module, which allows a controller computer to read/write local variables from/to engine computers using a very compact code. Comparing Map-Reduce to IPython parallel, Map-Reduce distributes data to many nodes attempting to reduce bottleneck of communication. However, overhead of communication when launching a Map-reduce job may take unexpected delay up to a minute [12]. Generating a deep tree using Map-Reduce may cost large latency overhead in a small size infrastructure. Therefore, the method proposed in this paper is suitable for medium size dataset, quick installation, interactive and fast learning process.

The proposed parallel Random Forest distributes dataset to all computing nodes by dividing the large original dataset equally and transferring to all computing nodes. When data is already in the computing node, the communication between a controller and nodes reduces. Unlike previous design, the proposed algorithm allocates computing processors to perform a single tree generating at a single time. This method allows computing system to process larger data because large amount of training dataset required large size of memory to store intermediate information. In [8] assigned a calculation of a tree (or trees) to a computing node without any help from other nodes, therefore the available memory to be used in calculation is limited by cache or memory of a single node. Moreover using a large number of decision trees do not contribute much on recall rate as shown in [2]. Increasing a number of decision trees larger than 3 trees makes saturated recall rate and much longer training time. Generating many trees is ineffective solution. The proposed algorithm also uses approximation of entropy to accelerate the merging histogram process. All detail of the new algorithm is discussed in the next section.

A. Sequential Algorithm

As similar as other supervise learning methods, the training process requires labels of class (C) and a table of dataset (D) for training, where the index of column is denoted by θ and x represents an index of row (or record). The value in the dataset at column θ and row x represented by $D(x,\theta)$.

```
Initialization Q = {x} samples of a root node
-For each attempt to find splitting parameters:
   -Propose the splitting parameters θ randomly
   -Sample x* for generating the thresholds
   -Generate threshold τ = D(x*, θ)
   -Allocate empty lists for QL and QR
      -For x in  :
      -If D(x, θ) < τ :
         -Put x in QL
      -Else:
         -Put x in QR
   -Compute information gain G
   -Find (θ, τ)* = argmax_(θ,τ) G
-splitting Q into QL and QR using (θ, τ)*
-Then recursively call the for loop to process QL
 and QR until reaching terminating cases
```

Fig. 1: Pseudo code of sequential Random Forest

```
Initialization Q = {x} samples of a root node
Initialization of engines
Append root node into a queue
-While queue is not empty:
   -Controller and engines dequeuer
   -Controller checks depth and |Q|
   -Engines generate a set of splitting candidates
    {(θ, τ)}
   -Controller gathers all splitting parameters from
    engines
   -Controller broadcasts all parameters back to
    engines
   -Engines calculate sub-entropy
   -Controller compute information gain G
   -Controller split the current node using the best
    splitting parameters (θ*, τ*) and append left and
    right nodes to the queue
```

Fig. 2. Pseudo code of Parallel Random Forest

The sequential process of generating a decision tree based on extremely random forest [7] can be summarized as shown in Fig. 1. At training initialization, the root node of the tree consists of a collection of samples($Q = \{x\}$). Then θ is randomly selected and some set of records x^* from dataset are uniformly selected and the indexes of records are stored in a parent node for calculation of threshold τ by using lookup table $D(x^*, \theta)$. In this splitting test, the value $D(x,\theta)$ of all x is compare to τ. If the value $D(x,\theta)$ less than τ, the record will be moved to the left node stored in node QL, otherwise the data will be stored in node QR. For each attempt of splitting, the objective is to maximize information gain G as in

$$G = H_Q - \frac{|QL|}{|Q|}.H_{QL} - \frac{|QR|}{|Q|}.H_{QR}.$$

Where H_Q is Shannon entropy [13] of the sample set Q. The size of sample set Q is denoted by $|Q|$. Once the information gain of all attempts are calculated, the best splitting parameter that yields largest information gain G is set to decision tree and permanently split the parent node Q into QL and QR. The

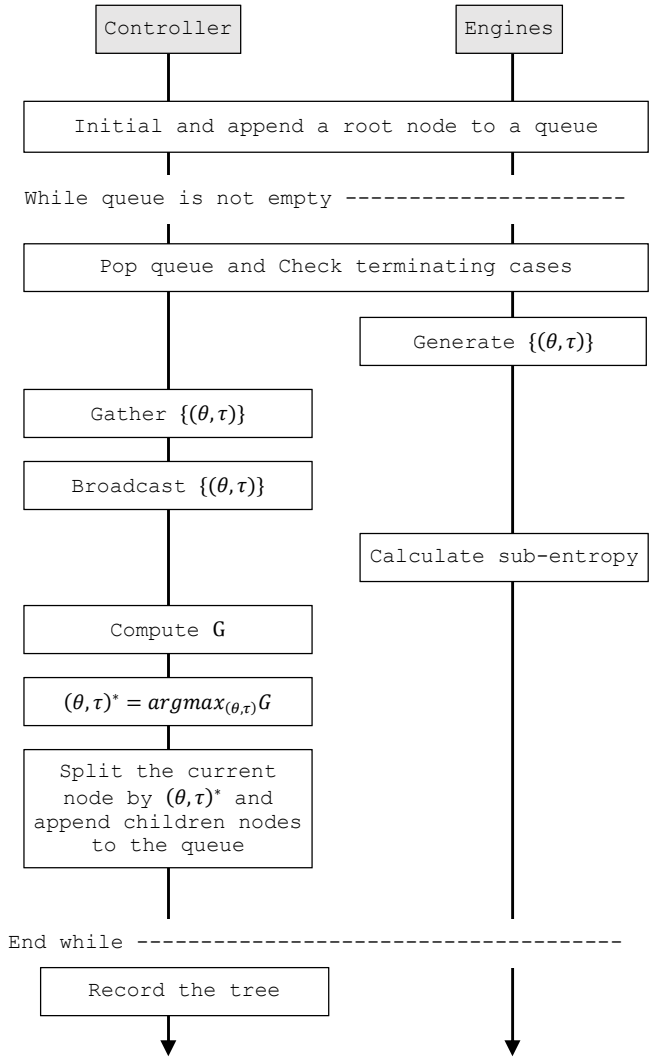

Fig. 4: sequential diagram of task sharing between controller and engines

process recursively calls splitting for QL and QR until reaching the terminating cases as following;

1. Depth of the decision tree reaching maximum depth
2. $|Q|$ is less than minimum bag size
3. There is no information gain after split test.

The complexity of the sequential algorithm to generate a tree is $O(kNlogN)$, where k a number of column of the dataset is and N is the number of the samples in the dataset.

B. Parallel Algorithm

The proposed method distributes dataset uniformly over the computing system as similar to SPMD (singe program multiple data). In order to minimize communication between nodes, small amount of information is transmitted over the cluster. To achieve the goal, the critical and data dependent tasks are computed by a controller. All computing nodes are assigned to help the controller to generate a single tree at a time. The calculation of the "for loop" in sequential algorithm is divided

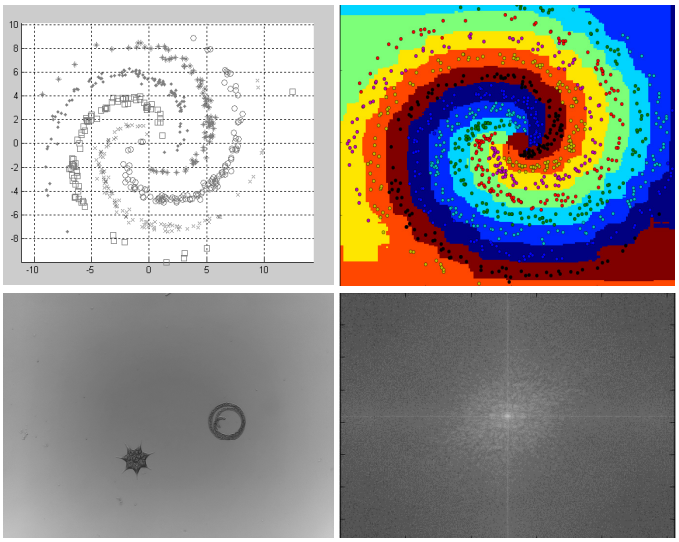

Fig. 3(top-left) synthetic data, (top-right) the result classification, (bottom-left) a sample of algae image and (bottom-right) FFT of the image before converted to feature vector

and distributed to all engines. Synchronization of recursion functions are replaced by queueing implementation giving cleaner code and better encapsulation.

As the sub-entropy is computed from every engine, the overall information gain can be computed by

$$G = H_Q - \sum_i \frac{|QL_i|}{|Q|} . H_{QL_i} - \sum_i \frac{|QR_i|}{|Q|} . H_{QR_i}.$$

, where i is the index of computing node. The calculation of entropy has complexity of $O(kNlogN)$ for sequential algorithm. The entropy calculation contributes high proportion of computational time. Profiling the sequential code with $1x10^5$ samples shows that entropy calculation takes about 97% of computational time. Distributing the task should significantly accelerates the over process. The proposed parallel version shares the entropy calculation.

The Parallel Random Forest algorithm starts with initialization of both controller and engines as shown in Fig. 2 and Fig. 4. Then the queues of all engines and controller are appended with initial root node to be processed. In while loop, the recent node is removed from the queue for the following calculations; generating the candidates splitting parameters by engines, gathering the splitting parameters to controller, and broadcasting all gathered parameters back to engine to calculate the sub-entropy. The best splitting parameters are used for make a next permanent splitting and the new children nodes are added into the queue. The source code of the proposed algorithm is in a repository [14].

III. EXPERIMENT

The experiment is divided into three parts. The objective of the first experiment is to verify the algorithm and implementation. The first experiment evaluates the algorithm by synthetic data on a private cloud.

The second experiment compares training time of our method with other standard machine learning tool.

In the third experiment, the algorithm is applied for image recognition. The extracted feature vector has high dimension, which makes the problems slightly difficult and takes longer time for training.

A. Synthetic Data on private cloud

The synthetic dataset in the first experiment is generated by Python script as can be found in [14]. The synthetic dataset is a collection of 2D coordinates of a spiral shape, where the points are labeled into many classes. The training dataset consists of two columns to represent (x, y) coordinate. The visualization of the synthetic dataset is shown in Fig. 3. The training set up of this experiment is following: maximum depth of tree of 20, minimum bag size of 2, number of splitting candidates of 100 per dimension of feature vector and the dimension of the feature vector is 2. The training is performed on private cloud infrastructure based on OpenStack [15]. The generated dataset is serialized and stored into a file before uploading into virtual machines. Each machine has same amounts of size but different data to be performed. The output from the training is set of decision trees that can be saved into a file. The result decision trees are later loaded for classification. The classifiers constructed by averaging output trees is evaluated by 10-fold cross validation to confirm the result accuracy. The classification normally take far shorter computing time compared to training process.

B. Training time comparision

In order to compare our implementation with other methods, the training uses the benchmark dataset CIFAR-100 [16], which consists 50,000 images divided into 100 classes each small image is RGB color 32x32 pixels. The dataset is used for comparing the training time with the standard machine learning tool scikit-learn 0.14 and scikit-learn 0.15[17]. The training processes are performed in the same infrastructure as in the first experiment using 8 computing nodes. In this experiment the maximum depth is set to 20 and dimension of the feature vector is 128 per color channel. The information of the raw image is reduced by using FFT to remove the high frequency spectrum.

C. Image recognition

In the second experiment a collection of algae images are used for generating dataset. The images are captured by a digital optical microscope attached by a Raspberry Pi CCD sensor. Fig. 3 shows an original algae image and the FFT image before extracting low frequency spectrum to be used in training. The image collection consists of 11 types of algae images. The collection of feature vectors are extracted from images by cropping into size of 200 by 200 pixels around a particular sample pixel. Each sample is converted to a feature vector using Fast Fourier Transform (FFT). Then high frequency domain is removed to reduce dimension a feature vector, which has remaining feature vector of 128 dimension. The feature vectors are pre-calculated once before training to reduce computational time. The setup of this experiment are:

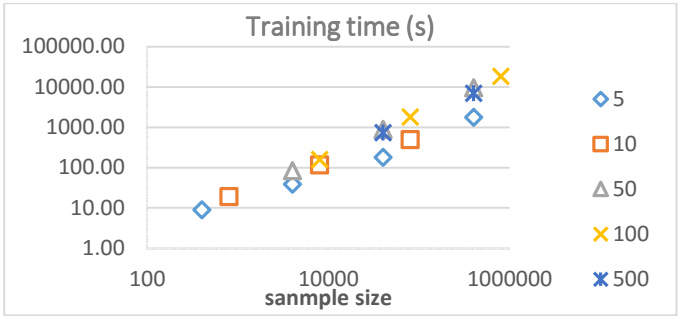

Fig. 5: training time of synthetic data using 8 cores on a vCPU

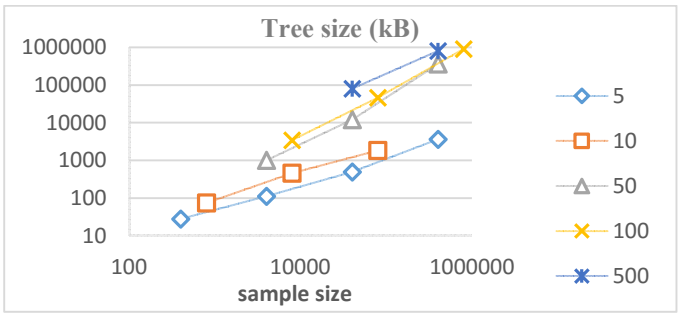

Fig. 6: output tree size from synthetic data using 8 cores on a vCPU

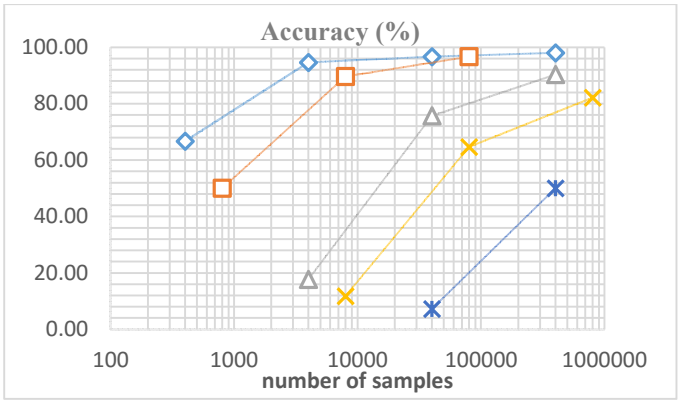

Fig. 7: result accuracy with synthetic data using 3 decision trees

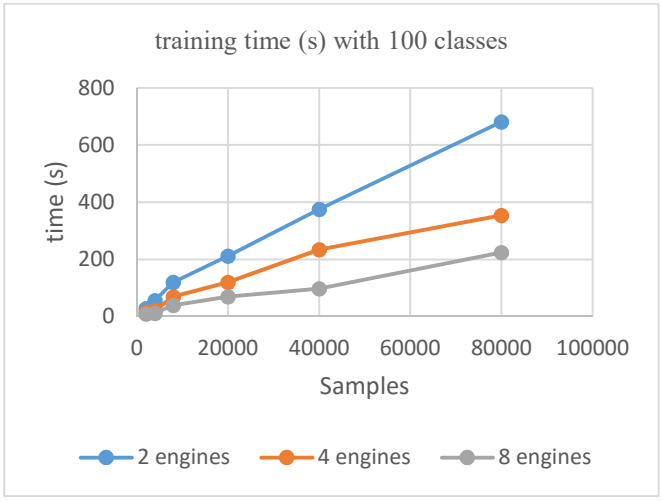

Fig. 8: computation time when varies a number of CPUs

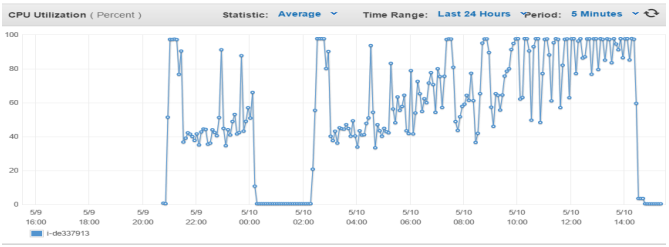

Fig. 9: utilization of vCPU during training of the algae image

10,724,760 sample cropped images, 11 classes of labels, a number of engines of 31 and 1 controller using EC2 elastic cloud, maximum depth of 20, minimum bag size of 2. A number of splitting parameter candidates is 100 per dimension of feature vector (the total number of candidates is 12,800). The feature extraction process is also computed in distributed computing system. The image collection is divided and stored in all computing engines equally. In this experiment, the cluster is initialized by using StarCluster [18] and the dataset is stored in the Elastic Block Storage volume. The volume is mounted before performing training. The training normally takes several hours. During the training the cluster is monitored by AWS console and via a SSH terminal. Therefore, the status of cluster and also utilization of all computing engines can be observed remotely through the internet.

IV. RESULTS

The results from both experiments, are observed and recorded by both SSH and AWS console to collect the following data: computation time, size of output decision tree and accuracy. In the first experiment, the number of engines is varied from 2, 4 to 8 engines and during the training the computation time is recorded. The relation between computation time and samples size is examined and result is shown in Fig. 5. The computation time of training process when the number of classes in the synthetic data varies as following 5, 10, 50, 100 and 500 classes. From the synthetic dataset, the computation time is approximately proportional to the number of samples used in the training. The training time also approximately proportional to the number of classes in the dataset. The proposed Parallel Random Forest can process simple 2D feature space scenario with sample size of $1x10^4$ samples within about 100 seconds.

A. Results from synthetic data

From the experiment, the computation time is fairly proportional to the sample size as shown in Fig. 5. An interesting thing we found that number of dimension has small effect on the computation time. This can explain by the small commination and no for-loop of classes in the algorithm. The output tree is serialized and saved using pickle module [19] and the size of decision tree is shown in Fig. 6. The size of output tree increases dramatically when the number of classes increasing because the complexity of feature space makes the algorithm continue split the tree to deep level. This may cause overfitting because the number of classes is too large compare to a small sample size. In Fig. 7, the accuracy clearly depend of the sample size and the number of classes. The large number of classes in dataset requires more complicated decision boundary in the hyper-dimensional feature space. The low accuracy from the experiment on 500 class dataset implies the number of samples is too small and samples are very sparse in the feature space. When a sample size increases, the accuracy rises until nearly touch 100% accuracy. The result accuracy increases as the number of sample increases and saturated just below 100%. This trend of improvement of accuracy is a characteristic of Random Forest [2].

Fig. 8 shows computation time when varying the number of engines and samples. Larger dataset size requires more computational time. At around 80,000 samples, the 2-engine process takes computational time around 3.6 times longer than 8 engines process. Converting 94% of sequential program to a parallel version and executed on 8 cores can have maximum speedup ratio around 6 times as predicted by Amdahl's law. However, in many practical cases overhead due to communication always reduces speedup ratio.

The results verified the algorithm works as expected. In next experiment, dataset from image recognition problem is tested.

Table 1: training time comparison

SciKit-Learn 0.14	SciKit-Learn 0.15	Our method
1207s	445s	430s

B. Comparison result

In this experiment, the dataset of 50k samples with 384 dimension of feature vector is used in the training process, in order to compare, the computing time between our method and others. The result from comparison the training time is shown as in Table 1, where our method is about 4% faster than SciKit-Learn0.15.

C. Result from algae image

After the original images are cropped and extracted the feature vectors of 128 dimensions. All 10 million feature vectors are distributed equally over the cluster and then the training process is ready to start. The training process takes around 12 hours. In Fig. 9, the CPU utilization is recorded during the training. The training starts just after 2:00 and finishes around 14:00. The utilization in this training process is around 80% and peak memory used by 32 CPUs is around 135GB. The results labeled images are shown in Fig. 10. The output trees are evaluated by cross validation and gives 98% of maximum accuracy, when training dataset of 11 classes and 1,116 images. The image pixels are labeled by hand before training. The manual label procedure is done similar to [2]. Note that the debris and white space in the image are labeled as a background class.

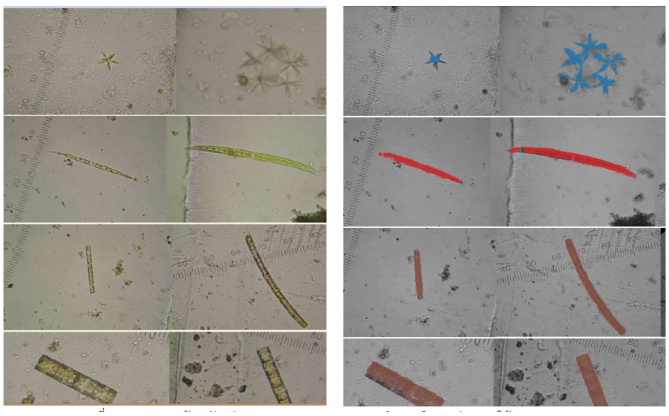

Fig. 10: (left) original unlabeled original images, (right) labeled images by using output tree from the training

V. CONCLUSION

In this paper, a new design of a parallel Random Forest is proposed to use many engines concurrently for computing sub-entropy in the splitting process in order to generate a single tree at a time. This gives opportunity for engines to store larger intermediate data in memory, therefore the proposed algorithm can handle medium to large dataset. The implementation shows capability to process large data with a size of 10 million samples (11 classes 128 dimension) in about 12 hours on 32 virtual CPUs ASW EC2. The proposed method based on IPython, which can be deployed easily in most of computing platforms.

ACKNOWLEDGMENT

This project is supported by Thailand Research Fund (TRF), contract No RDG5720034. The project receives remarkable helps from Kasidit Chanchio to set the private cloud for experiments. The author also gratefully acknowledges the partial support provided by Central Scientific Instrument Center (CSIC), Faculty of Science and Technology, Thammasat University.

REFERENCES

[1] L. Breiman, "Random Forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, Oct. 2001.

[2] J. Shotton, T. Sharp, A. Kipman, A. Fitzgibbon, M. Finocchio, A. Blake, M. Cook, and R. Moore, "Real-time Human Pose Recognition in Parts from Single Depth Images," *Commun ACM*, vol. 56, no. 1, pp. 116–124, Jan. 2013.

[3] T. Luo, K. Kramer, D. B. Goldgof, L. O. Hall, S. Samson, A. Remsen, and T. Hopkins, "Recognizing plankton images from the shadow image particle profiling evaluation recorder," *Syst. Man Cybern. Part B Cybern. IEEE Trans. On*, vol. 34, no. 4, pp. 1753–1762, 2004.

[4] H. Deng, G. Runger, E. Tuv, and M. Vladimir, "A time series forest for classification and feature extraction," *Inf. Sci.*, vol. 239, pp. 142–153, Aug. 2013.

[5] R. Díaz-Uriarte and S. A. de Andrés, "Gene selection and classification of microarray data using random forest," *BMC Bioinformatics*, vol. 7, no. 1, p. 3, Jan. 2006.

[6] J. R. Quinlan, *C4.5: Programs for Machine Learning*. Elsevier, 2014.

[7] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Mach. Learn.*, vol. 63, no. 1, pp. 3–42, Mar. 2006.

[8] L. Mitchell, T. M. Sloan, M. Mewissen, P. Ghazal, T. Forster, M. Piotrowski, and A. S. Trew, "A Parallel Random Forest Classifier for R," in *Proceedings of the Second International Workshop on Emerging Computational Methods for the Life Sciences*, New York, NY, USA, 2011, pp. 1–6.

[9] Y. Ben-Haim and E. Tom-Tov, "A Streaming Parallel Decision Tree Algorithm," *J Mach Learn Res*, vol. 11, pp. 849–872, Mar. 2010.

[10] J. Pješivac-Grbović, G. Bosilca, G. E. Fagg, T. Angskun, and J. J. Dongarra, "Decision Trees and MPI Collective Algorithm Selection Problem," in *Euro-Par 2007 Parallel Processing*, A.-M. Kermarrec, L. Bougé, and T. Priol, Eds. Springer Berlin Heidelberg, 2007, pp. 107–117.

[11] F. Pérez and B. E. Granger, "IPython: A System for Interactive Scientific Computing," *Comput. Sci. Eng.*, vol. 9, no. 3, pp. 21–29, May 2007.

[12] B. T. Rao and L. S. S. Reddy, "Survey on Improved Scheduling in Hadoop MapReduce in Cloud Environments," *ArXiv12070780 Cs*, Jul. 2012.

[13] S. L. Hartman, R. M. Howard, L. Johnson, J. S. Lacy, S. P. McMurray, and M. L. Ruehl, "Web-based system and application for collaborative planning of a networked program schedule," US8131579 B2, 06-Mar-2012.

[14] L. Wasit, "wasit7/parallel_forest · GitHub," *parallel forest*. [Online]. Available: https://github.com/wasit7/parallel_forest. [Accessed: 01-Jul-2015].

[15] "Home » OpenStack Open Source Cloud Computing Software." [Online]. Available: https://www.openstack.org/. [Accessed: 30-Jul-2015].

[16] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," 2009. [Online]. Available: https://scholar.google.co.uk/scholar?q=Learning+Multiple+Layers+of+Features+from+Tiny+Images%2C+Alex+Krizhevsky%2C+2009.&btnG=&hl=en&as_sdt=0%2C5. [Accessed: 07-Oct-2015].

[17] "Ensemble methods — scikit-learn." [Online]. Available: http://scikit-learn.org/stable/modules/ensemble.html#forest. [Accessed: 07-Oct-2015].

[18] "STAR: Cluster - Home." [Online]. Available: http://star.mit.edu/cluster/. [Accessed: 30-Jul-2015].

[19] "11.1. pickle — Python object serialization — Python 2.7.10 documentation." [Online]. Available: https://docs.python.org/2/library/pickle.html. [Accessed: 01-Jul-2015].