Data Access Request Tagging in Distributed Storage *

Joyanta Biswas
Temple University
Computer and Information Science
Philadelphia, PA 19122, USA
tug54519@temple.edu

ABSTRACT

Data access request tagging is an import aspect in prefetching, cache optimization and QOS (quality of service) management in high performance computing. In a distributed storage system, each of the access is sort of anonymous to the storage server. This make it hard to to undertake any action for better QOS management. The main goal of my project was to classify each of the request, by observing the data access pattern. I have used both the naive based approach and recurrent neural network (RNN) for the classification and found RNN better in term of accuracy.

1 MOTIVATION AND RELATED WORKS

Most of the storage related work that involves machine learning, targets "prefetching" [2] [1]. This is a technique the system uses to predict which storage unit is going to be accessed next, by analyzing the request pattern. By doing that, it can pre-fetch the data into the cache, thus minimize io latency. But one of the issue that none of the literature has addressed is the request tagging. The existing prefetching would work better when all the applications and data are stored in a single machine, which is not true for distributed storage. For distributed storage, prefetching is the next step, while tagging is the first step to make it work. That is why I took the challenge, so that I can initiate it and could improve it later as a part of my research work. Though there is not much work on storage, there are lot of work (still ongoing) on iot devices [3]. Like in [3], they have used n-order markov model for the n-gram clustering and they have pre-processed the data so that it can be fit in a language model. And what I found in most of iot works, they try to solve the problem by formulating their own problem in terms of NLP (natural language processing). I have followed the same approach in my project and that is what I am going to discuss in the following sections.

*Produces the permission block, and copyright information

Machine Learning, 2018

DOI: 10.475/123_4

2 DATASET AND PROBLEM FORMULATION

For the project I have used SNIA distributed storage dataset (http://iotta.snia.org/tracetypes/3). Each of the entry in the dataset D the features D_i =(timestamp, logical_block_address, I/O type, process_id, major_number, minor_number, md5 hash, process). D_i refers to one data access request.

- **timestamp**- the time of the request
- **logical_block_address** - logical block address that has been requested to access
- **major_number and minor number** - major and minor number of the device
- **I/O type** - Read/Write operation
- **process** - Application that made the request

Problem formulation: Given a request D_i, classify the request to any one of the given application set $A = A_1, A_2, ..., A_p$, where A is the set of unique elements of process column in D. Details of the problem formulation has been discussed in the subsequent sections.

3 PREPROCESSING

This was the main challenge I faced while working on the project.

Issue 1: At the beginning I thought of using naive bayes classifier for the classification. My thinking was something like: find the conditional probability of happening A (list of distinct applications/labels), with respect to the features (mainly logical block address). Then in the testing step, I would calculate the probability of being any application, given the features. But unfortunately later I found that the logical block address does not make any sense in distributed system (it is kind of relative to each of the instances). The feature that would be useful is the I/O type only, which is not enough for the prediction. But the logical block address can be used to find out the sequentality or randomness of any data access. Say for example application A_k has the following of logical block access [200, 208, 216,1000,1008,1012..]. Here 200, 208, 216 and 1000,1008,1012 these access are sequential and there is a event of random access between these two sequential access(216->1000). The difference 8 represents the sequentiality, because this is a byte addressable machine. To

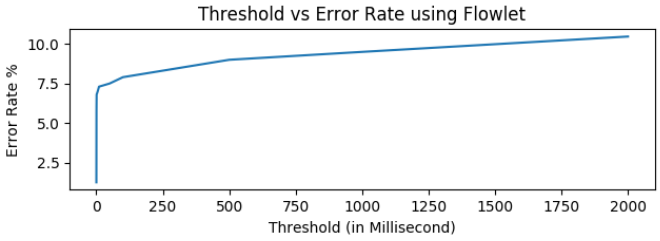

Figure 1: Threshold vs Error Rate

formulate my project as NLP sequence labeling problem, I have preprocessed the data as following:

(1) Split the entries based on timestamp using the concept of Flowlet (discussed in next para - Issue 2), all these entries corresponding to one application and work as a feature while training, validation and testing.
(2) In each of the subsets, for each of the entry
    - if the access is sequential,
        - put a initial character, say "a".
        - If this access is a read/write, then I put "R" or "W" respectively after that initial character, For example "aR" or "aW"
    - If the access is random,
        - put a initial character, say "b".
        - If this access is a read/write, then I put "R" or "W" respectively after that initial character, For example "bR" or "bW"
(3) After the whole operation, the subset is transformed to one feature, any the feature content is just a sequence of aR,aW,bR and bW, separated by space.

Thus the corpus of my NLP model has only 4 words, aR,aW,bR,bW. This is my initial thought, in the future enhancement I would add more words based on other attributes (if I find any useful attribute later). Labeling of the generated feature is discussed in the *Issue2* section.

Issue 2: This issue relates more to practical, rather than theoretical. As stated earlier, I have created subsets using Flowlet concept. This is well established concept used in networking to identify the flows from a stream of data packets. Whenever there is a significant amount of inter packet arrival gap, then the window is considered as a flow. Analyzing the dataset, I found there are many interleaved application access within a timestamp range in the dataset. This is pretty usual because this is streaming dataset. So it is really difficult to detect a application access sequence. So I used the same flowlet concept here to generate a feature, and assign that feature to that application which has higher access within this subset (e,g: feature). By doing this, we are losing some data, because some access are mislabeled. But choosing a good threshold(flowlet interval) value, the mislabeled rate

Application	Access_Pattern
kjournald	bR bW aW bR aW aW aW bR
pdflush	bW bW aW bW bW bW
kjournald	aW aW aW aW aW aW
.....	.....

Figure 2: Sample Preprocessed Data Set

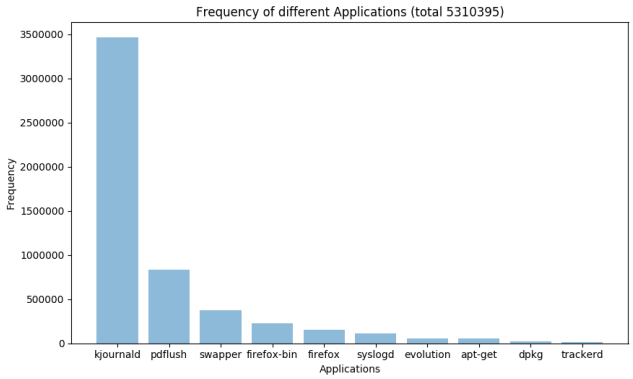

Figure 3: Application Frequency

can be reduced significantly. Fig 1 shows the error rate, based on different threshold.

With 2000 microsecond, error is 10.47% and with 0.1 microsecond,it has been reduced to 1.25%, which is not so much. And the trend obviously shows that lower the threshold (flowlet gap) better the error rate. But there is a trade off between accuracy and this error rate, which has been explained in evaluation section.Sample dataset is shown in Fig 2.

4 LEARNING ALGORITHMS, EDA, AND EVALUATION

I used three different learning algorithm on the preprocessed dataset, K-nearest neighbours, LSTM and naive bayes algorithm. K-nearest works worst in all the cases. LSTM outperforms naive bayes all the time. On a small data set (approx. 15% of the real dataset), the naive bayes has almost the same accuracy as LSTM, but on the real data set, the difference becomes huge. The frequency of the applications in the real data set and the preprocessed data set are given in Fig 3: The highest frequency belongs to kjournald (61.2%). So as rule of thumb, I expect my learning algorithms accuracy better than 61.2%.

Fig 4 and Fig 5 shows the write and read frequency of different applications. The insight from the EDA is that, the frequency of write operation is much greater than the Read operations (almost 10 times). And one interesting thing to

notice that, when there is a read operation, then it can be easily be predicted because since the frequency of access by one application named "swapper" is almost same as the read frequency. Bayes and KNN classifier would perform much better job in this case. The main challenge of prediction lies with the write operations. Because those applications that are much popular in access frequency, shows the same trend in write access (for example application "kjournald" and "pdflush"). So in that case, I assume generative sequence prediction algorithms like LSTM works better than the discriminative algorithms like Bayes classifier and KNN. Because random and sequential data access pattern plays a vital role for the classification.

Model Design Details:

For the KNN evaluation I have used the following distance function:

$distance_i = (num(aW) - num_i(aW))^2 + (num(aR) - num_i(aR))^2 + (num(bW) - num_i(bW))^2 + (num(bR) - num_i(bR))^2$. For the calculation of $num(String)$, I normalized the data entry. The values of k I tried with are 3, 5, 11, 17, 23, 35, 49 and got the best accuracy for $k = 23$, which was 72.3%. Most of the cases I got accuracy slightly better than 61.2%. In the evaluation section I report only the results for bayes classifier and LSTM.

For both the KNN, LSTM and Naive bayes, the ratio of Train and Test data is 80:20. For the naive data, I don't need to set any parameter/hyper parameter, since this is totally statistic driven classifier. For the LSTM, I defined:

- **input_dim**- the size of my dictionary, which is actually 4 as I stated in section 3. This would be higher, when I could figure out important features of my data-set.
- **output_dim** - The number of classes appears in the dataset, which is 50.
- **input_length** - The maximum length of the flowlet generated sentence, which I found 147.
- **Hidden Layers** - LSTM (64 units)-> Dense (256 dimension)-> Activation(RELU) -> Dense (50 dimesion) -> Relu(sigmoid)

5 EVALUATION

While making prediction of the label (in my case it is application) from the generated feature, I tried to evaluate it on two base:

- The maximum probability of occurrence of target/given strings in the training examples. Maximum sequence matching is not the goal here. Only the frequency of the individual word in the target/given string matters.
- The maximum likeliness of the target string pattern in the training examples.

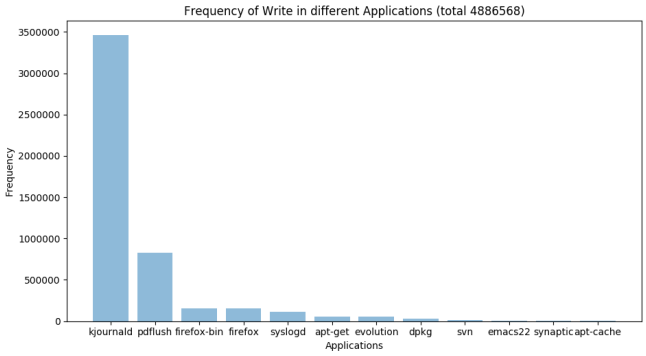

Figure 4: Application vs Write Frequency

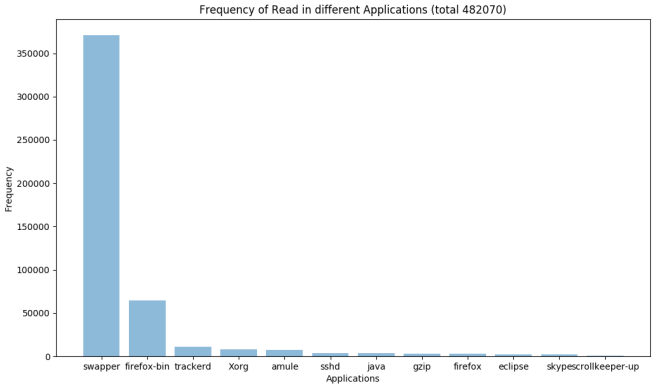

Figure 5: Application vs Read Frequency

These are the two reasons why I choose KNN, Naive Bayes and LSTM for the prediction. I tried to find out which one of them (frequency or sequence) actually makes more sense for the classification. In case frequency based prediction gives better accuracy, then degree of randomness is the better metric for the prediction. For the frequency based classification, I used KNN and Naive Bayes. Different sources claim that LSTM is a good choice for the time series based prediction, so I used LSTM for the pattern matching based classification. There might be other learning algorithms for the sequence based classification, which I would explore later.

The comparison between Naive Bayes and LSTM accuracy on small, medium and large(real) is shown in Fig 6. With a smaller dataset and medium dataset, the difference is not so much. But the difference is noticeable with the whole dataset. The accuracy of LSTM and Naive Bayes with optimal flowlet interval (discussed later) are 81.5% and 75.2% respectively in the real dataset. On the other hand with smaller dataset the difference in accuracy is trivial (89.17 % and 88.9 %). My understanding is, when the dataset size was small, there were considerable amount of variation in the dataset attributes in terms of randomness. When the dataset grows, the variations

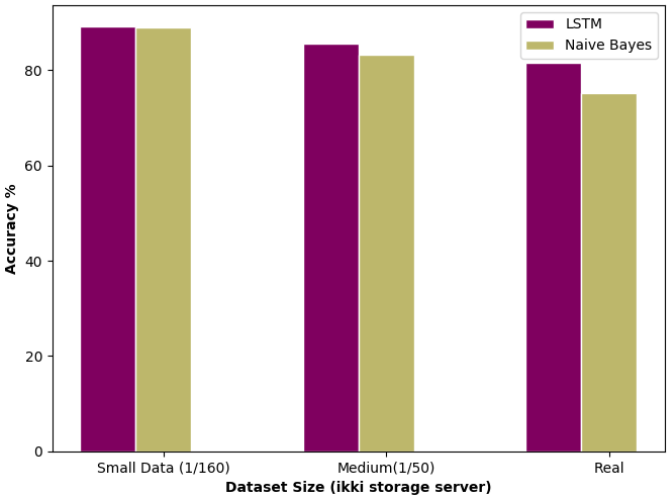

Figure 6: Accuracy comparison

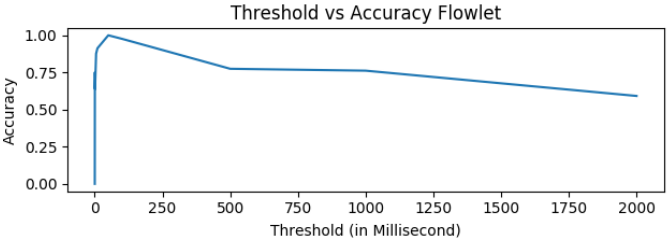

Figure 7: Accuracy and Flowlet Duration Trade off

becomes less and the sequence becomes more import aspect for the classification, as the number of word (corpus) in the dictionary is very less ("aW", "aR", "bW" and "bR"). Thus naive bayes gives less accuracy compared to the LSTM. **I was not able to run KNN on the medium and larger dataset, due to resource and time constraint**.

Comparison with respect to Flowlet duration: As stated in section 3, the error in mislabeling the access while doing Flowlet separation, is inversely proportional to the Flowlet duration. But if we continue to reduce the flowlet duration, after certain point, the error in mislabeled data might be less, but the accuracy downgrades severely. Because in that case there would be many splits and those splits can not reflect the real behavior of the pattern. This tradeoff relation has been shown in Fig 7. The accuracy increase until certain Flowlet duration, after that the accuracy begins to drop sharply and continues. **Note that, I run this experiment on medium size dataset due to time constraint and I found the optimal threshold interval is 50 microseconds. I used this interval for the large (real) data set too, though this might not be optimal for the real data set.**

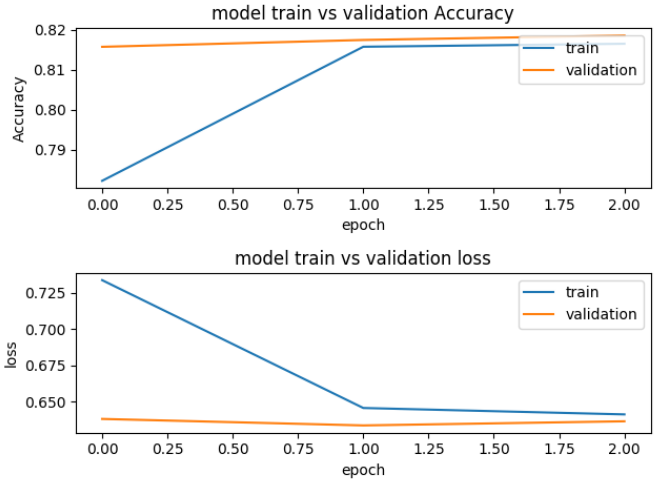

Figure 8: Train and test data accuracy and loss over different epoch

And lastly the change in loss and accuracy over the time (epoch), for the train and test data is shown in fig 8. The trend in the loss and accuracy over the test data implies no notion of overfitting. I used early stopping method as a remedy of overfitting.

6 FUTURE ENHANCEMENT

This work can be improved in several ways as I thought.

- Generating more useful features from the given limited attributes. The accuracy I got using LSTM is 81.5%. I am not satisfied with that because one application has 61.2 % (kjournald) of the data set, and one of the application named "swapper" contains all the read access. If those kind of insights are taken into considerations while preprocessing and training the model, then the accuracy might be better.
- Use maximum likelihood approach to understand the data(request) generation process and that can be used for prefetching in distributed system. Each application would act as a random variable of multivariate distribution in the estimation.
- Use this model for different data set from different distributed solution providers.

7 ACKNOWLEDGEMENT

Thanks to Golam Kayas (golamkayas@temple.edu) for the useful direction regarding the machine learning approaches in IOT systems. Thanks to Tanaya Roy (tanaya.roy@temple.edu) for the Trace file.

REFERENCES

[1] M. Chen, M. Mozaffari, W. Saad, C. Yin, M. Debbah, and C. S. Hong. Caching in the sky: Proactive deployment of cache-enabled unmanned aerial vehicles for optimized quality-of-experience. *IEEE Journal on Selected Areas in Communications*, pages 1046–1061, May 2017.

[2] L. Peled, S. Mannor, U. Weiser, and Y. Etsion. Semantic locality and context-based prefetching using reinforcement learning. pages 285–297, June 2015.

[3] Jiang Zhu, Pang Wu, Xiao Wang, and J. Zhang. Sensec: Mobile security through passive sensing. In *2013 International Conference on Computing, Networking and Communications (ICNC 2013)(ICNC)*, volume 00, pages 1128–1133, Jan. 2013.