Perplexity Method on the N-gram Language Model Based on Hadoop Framework

Tahani Mahmoud Allam[1], Hatem Abdelkader[2] and Elsayed Sallam[3]
[1,3]Computers and Control Engineering Dept., Faculty of Engineering,Tanta University, Egypt
[2]Information System Dept., Faculty of Computers and Information- Menofia University, Egypt

Abstract: *The N-gram language model is used in statistical natural language processing like machine translation and speech recognition. The evaluation method of the N-gram probability needs a testing process. We use a distributed computing platform by using MapReduce algorithm and Hbase tables in Hadoop. Hadoop is an open source implementation of the MapReduce framework. The comparative query process is dependent on the NoSQL database. The NoSQL database is used to store the testing data sets in tables with different structures. The evaluation process uses a MapReduce algorithm on the testing process which acting as a decoder but distributed. This decoder can process multiple testing texts together. There are two ways to perform the MapReduce query on testing data. First one called forward query and the second is hiding query. We focus on the query response time on a single user runs of three different corpora in the N-gram model. The perplexity method is a correct way to estimate the performance of the language model. The perplexity of the testing set is compared with traditional language modeling package SRILM Toolkit. The result is discussed depending on the choice of the different Hbase tables. The results demonstrate that the proposed framework provide enhanced performance such less time cost, small memory size.*

Keywords: *Perplexity model, Distributed language models, N-gram model,MapReduce, Hadoop framework, Hbasetables, SRILM Toolkit.*

Received December 9, 2014; Accepted May 31, 2015

1. Introduction

A building block in natural language processing and information retrieval defined as N-gram [1]. It is a sequence of a string data like contiguous words or other tokens in text documents. N-grams are essential in the tasks which need to identify words in noisy, ambiguous input [5]. It is essential to have a language model which measures the probability of how much words may occur in some context [11]. Depending on the smoothing and back-off methods the N-gram language model deals with the problem of data sparseness.

The distributed computing framework called Hadoop can be used for language modeling, and Hbase is a distributed database which may store the data model as database tables and integrate with Hadoop platform. On the testing process we use MapReduce, acting as a decoder but distributed. This decoder can process multiple testing texts together. Based on the Katz Back-Off model we estimate a specific n gram, if not found, then we estimate n-1 gram until reaches to unigram. For different Hbase table which constructed on the training data on earlier work [1] we need to generate different row and column names only.By using MapReduce, we can store multiple testing texts into HDFS. Then process all of them to generate the word counts using MapReduce, just the same as we have done in the training process on earlier work [1]. Then for each N-gram with its counts we directly estimate the probability using back-off model, and multiply by the counts. Each different N-gram is processed only once, which speeds up the whole process especially for lower order N-grams. This method called forward Query because we query each N-gram from Hbase table directly. The more testing N-grams, the more time it will cost. The perplexity used to estimate the evaluation value for the language model.

Organization.Section II describesthe mathematical concept of N-gram model. Sections III introduce the definition of Hadoop MapReduce framework and Hbase distributed database. In Section IV gives the detail of perplexity methods and Back-Off method. In Section V shows the SRILM Toolkit. In Section VI shows the experiments and results. Finally, Section VII the conclusion.

2. Mathematical Concept of N-Gram

The ability to predict the next word is important for augmentative communication systems [8]. N-gram algorithm used to assign the probability of sentences. And also can assign a probability to the next word in an incomplete sentence, and vice versa

In speech recognition, it is traditional to use the term language model language model or (LM) for a statistical model of word sequences. Probabilities are based on counting things.

The simplest N-gram model of word sequences would make the word of the language follow any other word. So that, every word would have an equal probability of following every other word. It is not accurate measure so we should use a more complex N-gram model. To compute the probability of a complete string of words $w_1, w_2, \dots, w_n$. As an independent event, we might represent this probability as in Equation (1):

$$p(w_1, w_2, \dots, w_n) \quad (1)$$

We can use the chain rule in Equation (2) to analyze this probability:

$$p(w_1^n) = p(w_1)p(w_2|w_1)p(w_3|w_1^2) \dots . p(w_n|w_1^{n-1}) \quad (2)$$

A conditional probability is the suitable way to compute the probability. In the bigram model we can approximates the probability of a word given all the previous words. On Markov assumption, the prior local context affectson the next word [7].Equation (3) the probability function can be expressed by the frequency of words occurrence in a corpus using without smoothing:

$$p(w_n|w_1 \dots w_{n-1}) = \frac{f(w_1 \dots w_n)}{f(w_1 \dots w_{n-1})} \quad (3)$$

where f ($w_1,\dots,w_n$) is the counts of how many times we seen the sentence $w_1,\dots,w_n$in the corpus.

As in Equation (4), N-gram models can be trained by counting and normalizing. The normalizing means to divide the count of a specific word on total count. So the probability is always fall between 0 and 1. We take some training corpus, and from this corpus take the count of a particular bigram, and divide this count by the sum of all the bigrams that share the same first word:

$$p(w_n|w_{n-1}) = \frac{C(w_n \cap w_{n-1})}{C(w_{n-1})} \quad (4)$$

There are some techniques we can use to assign a non-zero probability. But if the probability is zero, we can reevaluate these probabilities to make them non-zero values. This technique is called smoothing. There are a lot of smoothing can be used. The add-ones smoothing, one of a simple way to do smoothing might be just to take our matrix of bigram counts, before we normalize them into probabilities, and add one to all the counts. But this type is not accurate. The adjusted count for add-one smoothing is defined in Equation (5):

$$C_i^* = (C_i + 1)\frac{N}{N+V} \quad (5)$$

Alternatively we can compute the probability directly from the count as follows in Equation (6):

$$P_i^* = \frac{C_i+1}{N+V} \quad (6)$$

The Good-Turing smoothing algorithm was described by Good (1953), who credits Turing with the original idea [7]. The basic idea of Good-Turing smoothing is to re-estimate the amount of probability mass to assign to N-grams with zero or low counts [5]. By examine N_c which is the number of N-grams that occur c times. We refer to the number of N-grams that occur c times as the frequency of frequency c. So by using this idea to smoothing the conditional probability of bigrams, N_0 is the number of bigrams of count 0, N_1 the number of bigrams with count 1, and so on.

The Good-Turing estimate gives a smoothed count c based on the set of N_c for all c, as in Equation (7):

$$C^* = (C + 1)\frac{N_{c+1}}{N_c} \quad (7)$$

If we have no examples of a particular trigram $w_{n-2}w_{n-1}w_n$ to help us compute $P(w_n|w_{n-1}w_{n-2})$, we can estimate its probability by using the bigram probability $P(w_n|w_{n-1})$. Similarly, if we don't have counts to compute $P(w_n|w_{n-1})$. We can look to the unigram $P(w_n)$.

$$p(w_n|w_1 \dots w_{n-1}) = \begin{cases} p(w_n|w_1 \dots w_{n-1}) \ if \ found \ (w_1 \dots w_n) \\ \partial(w_1 \dots w_{n-1}) * p(w_n|w_2 \dots w_{n-1}) \ otherwise \end{cases} \quad (8)$$

where$\partial(w_1 \dots w_{n-1})$is the back-off weight. Modern back-off smoothing techniques like Kneser-Ney smoothing [5] use more parameters to estimate each N-gram probability instead of a simple Maximum Likelihood Estimation.

3. Hadoop Framework and Hbase Tables

For coding and running distributed applications that process a massive amount of data we can use Hadoop. Hadoop is an open source implementation of MapReduce model[1].MapReduce based on Java and Hadoop Distributed File System (HDFS). The HDFS used to create a multiple blocks of data used. This makes a model more reliable. Also the HDFS separate the task to small blocks. According to this paper, Hadoop demonstrated on 3,000 nodes and designed to support 20,000 nodes on the clusters.

For input text files, each line is parsed as one string which is the *value*. For output files, the format is one *key/value* pair per one record, and thus if we want to reprocess the output files, the task is working at the record pair level [1]. The first step is to split the input files to small blocks which called FileSplits. The operation of the Map function is parallel working on one task per FileSplit. The input and output types of a MapReduce job should be in <key and value> pairs. The FileSplit input is treated as a key/value pair, and user specifies a Map function to process the key/value pair to generate a set of intermediate key/value pairs [6]. After Map function operation finished, all output

pairs which have the same key can be collected together. The intermediate pairs are generated from the Combine function. Then all intermediate values are merges on the Reduce function and writes to output files. Map and Reduce operations are working independently on small blocks of data [6].The final output will be one file per executed reduce task, and the output files stores by Hadoop in HDFS.

For language model training, which described on paper using MapReduce [1], our original inputs are text files. The N-gram/probability pairs will finally get from Hadoop. Theoretically we can only use Hadoop to build language model.

The distributed Hadoop database on top of HDFS is Hbase.The Hbase structure is very similar toGoogle's Bigtable model. Hbase is an open-source, distributed, versioned, column-oriented store modeled after Google's Bigtable [2]. Hbase Tables are designed to provide random, realtime read/write access to very large tables, billions of rows and millions of columns[1]. The time cost and computations will be increased if we used the Hadoop and HDFS only. The Hadoop MapReduce is mainly used to store the input/output files.

In the query process the mare suitable choice is the Hbase tables.Hadoop and HDFS is not suitable for query process. Because if we want to query probability for one N-gram, we have to load all the files into map function, parse all of them, and compare the key with N-gram to find the probability value. Basically it will cost quite a long time because we need to do this comparison for each N-gram in test texts. So we can make use of a database structure such as Hbase, to store the N-gram probabilities in database table instead of parsing files. The advantage of the Hbase databasestructure is designed to meet the needs of multiple queries.Hbase tables provide the scalable and reliable storage. On a language model, the model data is highly structured. The basic format is N-gram/probability pair, which can be easily constructed into more organized and compact structures [1]. The compressed structures are essential from both time and storage aspects because we get huge amount of data.

Hbase stores data in labeled tables. The table is designed to have a sparse structure. Data is stored in table rows, and each row has a unique key with arbitrary number of columns [1]. Figure 1 shows the relationship between Hadoop, Hbase and HDFS.

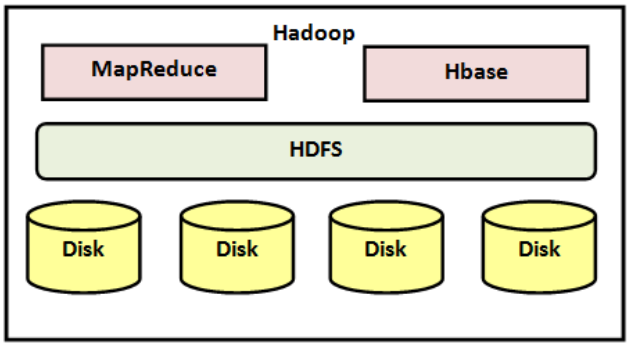

Figure 1.The relationship between Hadoop, Hbase and HDFS.

4. Perplexity Method

The correct way to evaluate the performance of a language model N-gram is to embed it in an application [5]. There are some methods to evaluate this performance. One of these methods called in vivo evaluation. But this type is very expensive on time which may take hours or even days. Perplexity (PP) is the most common evaluation metric for N-gram language models.

The intuition of the perplexity depends on two probabilistic models. The better model is the one that has tighter fit to the test data also predict better details on the test data. A better prediction depends on the probability. If the probability of test data is high the model is good. If the probability of the word sequence is high, then the perplexity is then low. Minimizing the perplexity is equal to maximizing the probability.

The mathematical concept of the perplexity of language model on a test set is function of probability normalized by the number of words. The perplexity of a test set $W=w_1w_2\ldots w_N$ is in Equation (9):

$$pp\ (w) = p(w_1 w_2 \ldots\ldots w_n)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{p(w_1 w_2 \ldots\ldots w_n)}} \quad (9)$$

Equation (10) shows perplexity of bigram model using chain rule:

$$pp\ (w) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{p(w_i|w_{i-1})}} \quad (10)$$

On the other hand, there is another way to think about perplexity. According to the weighted branching factor of language. The branching factor is the number of possible next words that can follow any word. As an example, if we want to compute the probability of each ten digit from zero to nine, the probability equal 1/10 and the branching factor for each digit will be 10.

Our job is to compare four N-gram models on a test set. We trained unigram, bigram, trigram and four gram on 40 million words [1]. So the results will shows later. In this paper we compute the perplexity of each of four N-gram models on testing data according to the equation (9).the perplexity of N-gram must be

constructed without any knowledge of testing data. Because any previous knowledge of test set make low perplexity.

To compute the perplexity on testing data we can follow two algorithms. The first algorithm called forward processing. And the second algorithm called hiding processing. The forward query depends on the Katz backoff model. For each testing N-gram we query the N-gram based on the backoff model. If the N-gram dose not found then we used N-1-gram until reach to the unigram. On the second algorithm, we compute the perplexity by considering the different N-gram that share the same N-1-gram. The hiding query has little speed than forward query.

Katz backoff N-gram model is used if N-gram we need has zero counts, so we estimate it by backing off to the N-1-gram. We apply backing off until we reach a history that has some counts. Figure 2 shows the flow chart of the testing process.

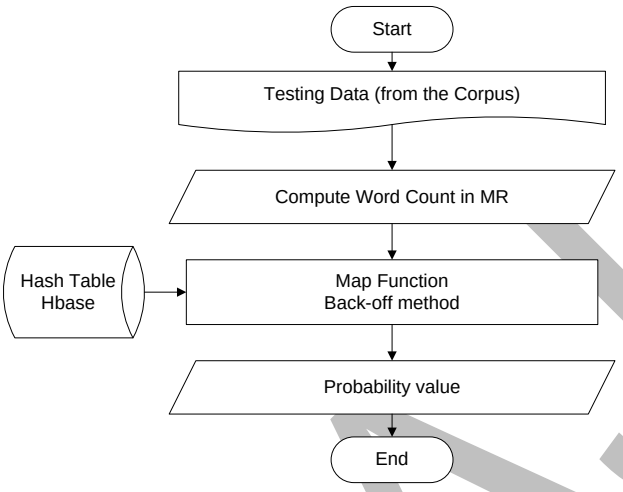

Figure 2.Flow chart of testing process.

5. SRILM Toolkit

There are two available toolkits using for building language models, The SRILM toolkit [9] and the Cambridge-CMU toolkit [4]. SRILM is a collection of C++ libraries, executable programs. It is used to allow the production and experimentation with statistical language models for speech recognition and other applications. SRILMis freely available for noncommercial purposes [5]. The toolkit supports creation and evaluation of language model types based on N-gram statistics.

Various software packages for statistical language modeling have been used for many years. The basic algorithms are simple enough that one can easily implement them with reasonable effort for research use. The CMU-Cambridge LMtoolkit [4], has been in wide use in the research community and has greatly facilitated the construction of language models (LMs)for many practitioners. SRILM toolkit takes a raw text file, one sentence per line with words separated by white space. And the output is a language model in ARPA format.

The main purpose of SRILM is to support language model estimation and evaluation. Estimation means the creation of a model from training data. Evaluation means computing the probabilityof a test corpus, conventionally expressed as the test set perplexity. Since most LMs in SRILM are based on N-gram statistics. A standard LM (trigram with Good-Turing discounting and Katz backoff for smoothing) would be created by*ngram-count -text TRAINDATA -lm LM*. The resulting LM may then be evaluated on a test corpus using *ngram -lm LM -ppl TESTDATA -debug 2*.The ngram – debug option controls the level of detail of diagnostic output. A value of 2 means that probabilities are to be reported at the word level, including the order of N-gram used, in addition to the standard log probabilities and perplexities.

SRILM treats everything between whitespace as a word by itself with no text conditioning. Normalization of text is highly corpus-dependent [10]. SRILM designed to ensure that enhancements by others find their way back into the user community. Licensing for commercial purposes is also available. Documentation and software are online athttp://www.speech.sri.com/projects/srilm/.

6. NoSQL Query Process in Hadoop

In this section, we estimate the quality of N-gram language model. By using perplexity (PP) for a testing set we evaluate the LM quality and compared with traditional language modeling tools SRILM. The model data size using SRILM is computed as a reference in the comparison with PP.

6.1.Forward NoSQL Query

The Katz back-off is performed in the forward query [7]. Based on the back-off model, for each testing N-gram, we need to query the raw counts of N-gram, if not found, then find (N-1)-gram, until we reach to the raw count of unigram [11]. For different table structures [1], we just need to generate different row and column names. The advantage of using MapReduce for the testing is that, we can put multiple testing texts into HDFS, and a MapReduce job can process all of them to generate the raw counts, just the same as we have done in the training process [1], then for each N-gram with its counts, we directly estimate the probability using back-off model. In such a method, each different N-gram is processed only once, which speeds up the whole process especially for lower order N-grams.

We call this method Forward Query because we query each N-gram directly from Hbase table, so the more testing N-grams we have, the more time it will cost. Also the perplexity of the estimation is computed and collected as an evaluation value for the language in

the next section. Figure 3 shows the flowchart of the forward NoSQL query.

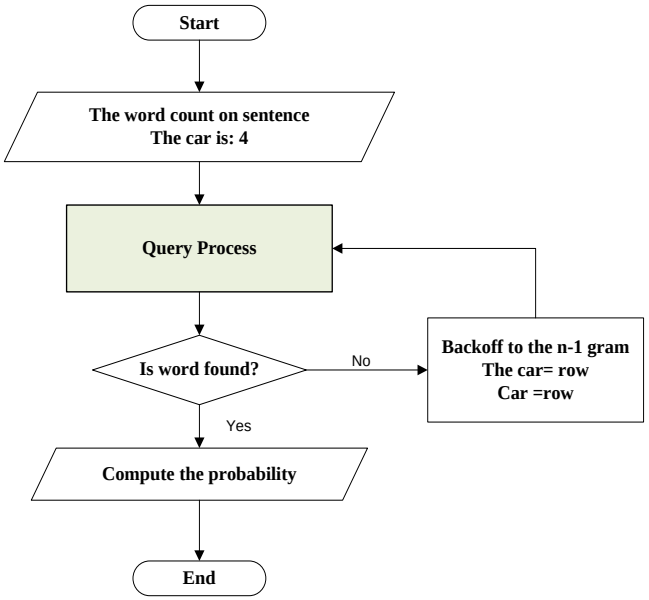

Figure 3.Flow chart of forward query.

The pseudo code of the forward query for the probability based structure [1] is:

Data Input
Compute the words count for each N-gram model

Start query process
Method map():
Words=line split

Go through all words in the corpus
Complete all words
Collect the output

Method reduce():
Collect the output
Collect the counts of each word in the corpus

Method.Estimate():
Column = "gt:prob"

6.2.Hiding NoSQL Query

The hiding query makes a simple modification in the time of the query process than the forward query. Considering queries for different N-gram that share the same (N-1)-gram, in a back-off model we query the N-gram first, then if not found, we move on to (N-1)-gram. Suppose we need to back-off for each N-gram, the (N-1)-gram will be requested for multiple times [11]. Here is where the hiding steps in. For each back-off step, we store the (N-1)-gram probability as HashMap in memory inside the working node. Every time when node comes to a new n-1 backoff query, it will first look up in the HashMap, if not found, then ask the Hbase table, and add the new (N-1)-gram into HashMap. Figure 4 is the flowchart of the hiding query process.

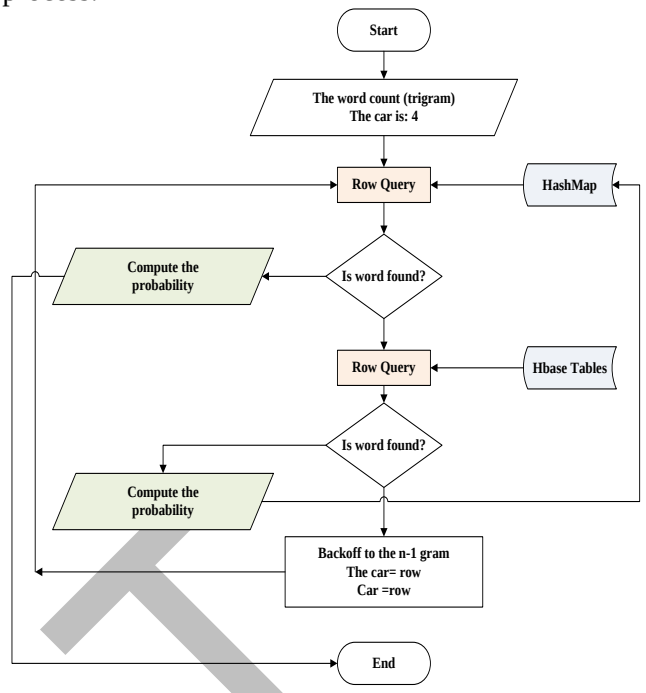

Figure 4.The flowchart of the hiding query.

We don't need to store probabilities for N-grams, only the (N-1)-grams. Also there is a maximum limit of the number of keys in the HashMap. We can't store all the (N-1)-grams into HashMap, otherwise it will become huge and eat up the entire working node's memory. So we only store up to maximum limit (N-1)-grams, and when counts are over the limit, the previous HashMap is dropped and filled in new items. It is like an updating process, and another alternative is to delete the first key in HashMap and push in new one. The pseudo code can be written as:

Data Input
Compute the words count for each N-gram model
Start query process

For k as the N-gram
Hide the HashMap
while not finished
prob = get table

ifprob != null
found probability, finished
else
let row be the n-1 gram from k
if exit cache then prob = get cache
else
prob = get table

ifprob !=null then found probability, finished
if number of cache.keys<maxlimit

7. Application Experiments and Results

The performance was measured on the same sets of data. We used three different corpora from three

different sources. The first data is about 200 million words from the British National Corpus (BNC). The second data is about 1.8 millions newspaper articles from the New York Times Annotated Corpus (NYT). The third data is about 50 million web documents in English language from the ClueWeb09-B (C90).

We trained the N-gram models in the previous work in paper [1]. The probability of the different N-gram models was stored in the Hbase tables in four different structures [1]. In the experimental results, we use HNM approach. It based on Hadoop framework related to the Hbase tables. The performance includes: the time cost in the query process, the space size needs, and the model evaluation using the perplexity in both approaches. We compare the perplexity with the traditional evaluation method SRILM.

7.1. The Experimental Setup

For the HNM the experiments are done in a cluster environment with 2 working nodes and 1 master server. Operating System is Ubuntu 10.04. 6 cores CPU, 64 GB main memory, 2 TB HDD. Use open source Hadoop 1.1.2 running on Oracle Java 1.6.02_26. The working nodes are running Hadoop, HDFS and Hbase slaves, and the master server controls all of them. For each experiment, we repeat five times and choose the average value as the result.

7.2. The Performance of Query Processes

According to paper [1] in the HNM, after training process complete, query process starts in different two ways in the HNM. Tables 1, 2, 3, 4, 5, and Table 6 show the time cost for testing process in the querying steps. We apply the query process on three different corpora.

Table 1. Forward query process time cost (min.) for BNC data.

Gram order / Hbase types	*Unigram*	*Bigram*	*Trigram*	*4-gram*
Type 1 (PBS)	6	35	45	90
Type 2 (UWBS)	14	55	110	185
Type 3 (UHBS)	6	25	56	98
Type 4 (CHBS)	6	25	58	97

Table 2. Hiding query process time cost (min.) for BNC data.

Gram order / Hbase types	*Unigram*	*Bigram*	*Trigram*	*4-gram*
Type 1 (PBS)	5	33	40	87
Type 2 (UWBS)	17	50	110	160
Type 3 (UHBS)	7	20	45	94
Type 4 (CHBS)	6	20	44	90

Table 3. Forward query process time cost (min.) for NYT data.

Gram order / Hbase types	*Unigram*	*Bigram*	*Trigram*	*4-gram*
Type 1 (PBS)	4	32	55	60
Type 2 (UWBS)	11	43	100	106
Type 3 (UHBS)	4	19	58	98
Type 4 (CHBS)	3	20	58	66

Table 4. Hiding query process time cost (min.) for NYT data.

Gram order / Hbase types	*Unigram*	*Bigram*	*Trigram*	*4-gram*
Type 1 (PBS)	4.5	32	56	63
Type 2 (UWBS)	12	43	104	106
Type 3 (UHBS)	5	20	59	99
Type 4 (CHBS)	3.8	20	58	67

Table 5. Forward query process time cost (min.) for C90 data.

Gram order / Hbase types	*Unigram*	*Bigram*	*Trigram*	*4-gram*
Type 1 (PBS)	7	40	66	68
Type 2 (UWBS)	12.6	49	112	176
Type 3 (UHBS)	5.8	28	59	106
Type 4 (CHBS)	3.9	25	59	70

Table 6. Hiding query process time cost (min.) for C90 data.

Gram order / Hbase types	*Unigram*	*Bigram*	*Trigram*	*4-gram*
Type 1 (PBS)	8.2	40.8	68	70
Type 2 (UWBS)	14	51	140	179
Type 3 (UHBS)	8	30	77	132
Type 4 (CHBS)	7	39	66	78

Figures 5, 6, and 7shows the time cost in two query processes. The query processes applied on three different corpora.

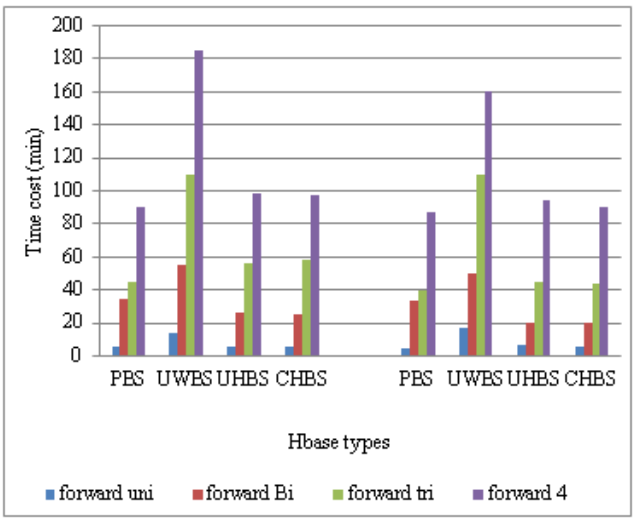

Figure 5. Time cost in forward & hiding query of BNC

.

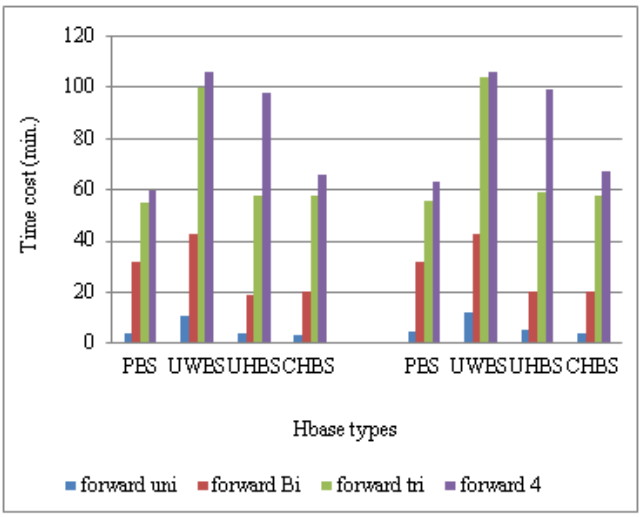

Figure 6. Time cost in forward & hiding query of NYT.

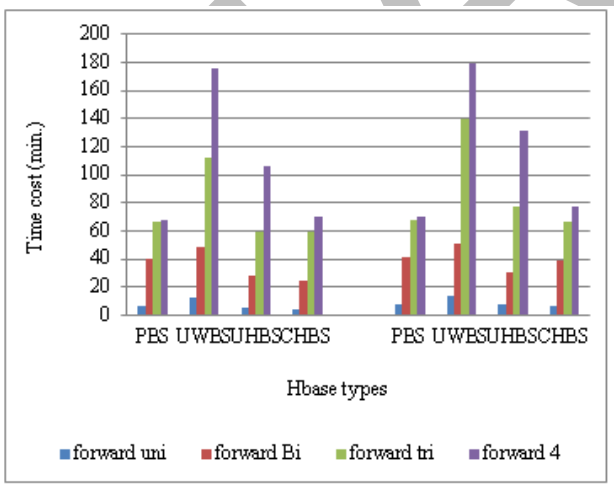

Figure 7. Time cost in forward & hiding query of C90.

The time consumed for forward and hiding query in UWBS is the bigger than other tables. There is some kind of redundancy. This is because the rows are less expensive than more columns. Also the data structure is still uncompressed. There is some kind of redundancy. PBS has a good time results. In general, for a big data the best time performance is found in CHBS. This is because CHBS aggregate UWBS and UWBS together. This makes a balance in row and column numbers. Hiding query has a less time cost than forward query. It stored all the history probability on hash table.

Figure 8explains the space size needs in two query processes. We take the average space value for each N-gram order.

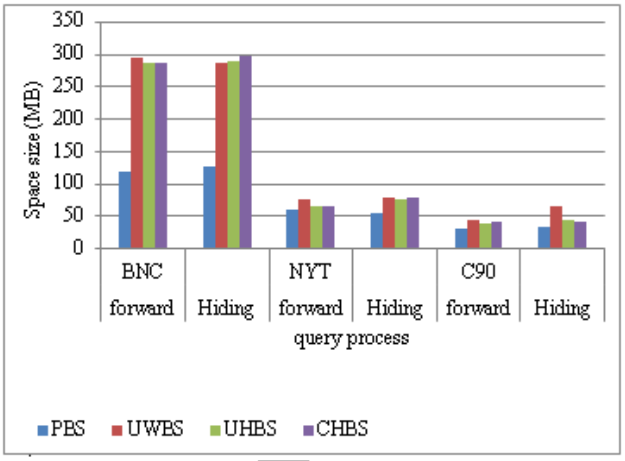

Figure 8.Space size in two query processes.

Perplexity is the model used to evaluate the N-gram model. The perplexity of testing processes compared with SRILM is shown in Table 7. The SRILM Toolkit and the perplexity are computed using default parameters for unigram, bigram, trigram and 4-gram models. In HNM, the perplexity is better than SRILM model especially in a low N-gram order. On Figure 9, we shows the perplexity on Hadoop and Hbase is more accurate than traditional SRILM model.

Table 7. Perplexity & SRILM

aluation method / N-gram order	*Hadoop perplexity*	*SRILM*
unigram	2271.958	3245.654
Bigram	470.549	588.187
Trigram	461.911	512.879
4-gram	209.325	348.876

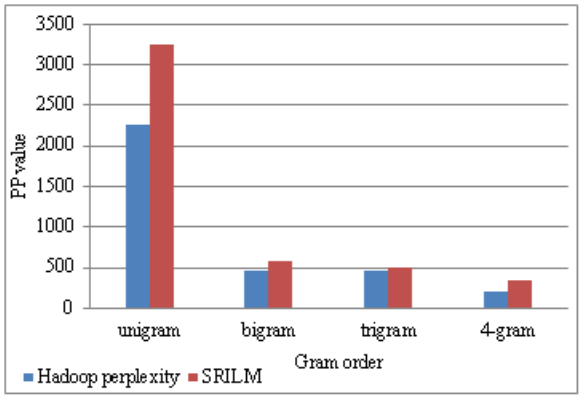

Figure 9. Relationship between Hadoop and SRILM perplexity.

As shown in Figure 9, the high perplexity found in lower order N-gram. Perplexity is high when dealing with high N-gram order. The reason for this variation is for the huge data founded in low N-gram order. But in the high order gram the frequencies of words are small.

The important factor on a comparison between the SRILM and perplexity is the size of the language model data. By default the size of files in SRILM is 780 MB without compression. But in Hbase we have used compression, we compress the file with gzip and the compressed size is only 175 MB, which is a 77.56% compression.

8. Conclusion

This bibliographical study concerned on how to measure N-gram algorithm quality of natural language process based on Hadoop framework. From our search result; a good choice for distributed language model using Hadoop and Hbase is CHBS. PBS is a flat Hbase structure which is fast and easy to manipulate locally. Also it is widely used in traditional language modeling tools like SRILM. For the space cost in the testing set, each run the table size is identical, which means the model is stable. The CHBS table size is also smaller compared with SRILM model data size. This makes the CHBS is the best choice. There is small improvement in time on the hiding query compared with the forward query. This means in a MapReduce job it is not efficient way to store a cache in memory and check both the memory and the Hbasetable. Because there is a lot of time wasted until node search on the received whole data and to the stored data. Our model shows a better result than SRILM in the perplexity comparison. SRILM uses a more complex way to compute the probability for unseen unigrams. Of course needs more steps using MapReduce. If there are not so many unseen unigrams this algorithm is good.

References

[1] Allam, T., Abdelkader, H., and Sallam, E., "Distributed Data Storage of the Language Model Based on Hadoop Framework", submitted to Computer Speech and Language, CSL 15-75, Elsevier, 2015.

[2] Apache HBase project: http://hbase.apache.org

[3] Church, K.W., and Gale, W. A., "A comparison of the enhanced Good-Turing and deleted estimation methods for estimating probabilities of English bigrams". Computer Speech and Language, 1991, 5, 19–54.

[4] Clarkson, P., and Rosenfeld, R., "Statistical language modeling using the CMU-Cambridge toolkit", in G. Kokkinakis, N. Fakotakis, and E. Dermatas, editors, Proc. EUROSPEECH, vol. 1, pp. 2707–2710, Rhodes, Greece, Sep. 1997.

[5] Daniel, J., and Martin, H., "Speech and Language Processing", An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition, Prentice Hall, Englewood Cliffs, New Jersey 07632, September 28, 1999.

[6] Dean, J., and Ghemawat, S., "Mapreduce: Simplified data processing on large clusters", In OSDI '04, pages 137–150, 2004.

[7] Kneser, R., and Ney, H., "Improved backing-off for m-gram language modeling", Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on, 1:181–184 vol.1, 9-12 May 1995.

[8] Newell, A., Langer, S., and Hickey, M., "The role of natural language processing in alternative and augmentative communication", Natural Language Engineering, 1998, 4(1), 1–16.

[9] Stolck, A., "SRILM- an Extensible Language Modeling Toolkit",Speech Technology and Research Laboratory SRI International, Menlo Park, CA, U.S.A. 2002.

[10] Wang, W., Liu, Y., and Harper, M. P., "Rescoring effectiveness of language models using different levels of knowledge and their integration", in Proc. ICASSP, Orlando, FL, May 2002.

[11] Xiaoyang, Y., "Estimating language model using hadoop and Hbase", Master of Artificial Intelligence, University of Edinburgh, 2008.

Tahani Allam is a Ph. D. student and an assistant lecturer at Computers and Control Engineering Dept. Faculty of Engineering, Tanta University, Tanta, Egypt. She was born in Kuwait at 1980, her B.Sc. and M.Sc. degrees taken from Computers and Control Engineering Department - Faculty of Engineering - Tanta University at 2002 and 2009, respectively. Her search interests include Data Base, Cloud Computing, Hadoop framework. She has a paper published in the International conference on Computing Technology and Information Management, (SDIWC), pp 455-460, Dubai, 9-11 April, 2014. The second paper published in the 9th International Conference on Informatics and Systems 2014 (INFOS), Cairo University, Egypt pp., 15-17 December 2014. The third paper was submitted on Computer Speech and Language, CSL 15-75 on to the Elsevier Editorial System, 2015.Eng.Tahani Works as a consultant Engineer Management Information Systems (MIS) Project – Tanta University, Egypt, from August 2008 until Now.

Hatem Abdelkader obtained his B.Sc. and M.Sc. (by research) both in Electrical Engineering from the Alexandria University, Faculty of Engineering, Egypt in 1990 and 1995 respectively. He obtained his Ph.D. degree in Electrical Engineering from the Alexandria University, Faculty of Engineering, Egypt in 2001 specializing in neural networks and applications. He is currently a professor in Information systems department, Faculty of Computers and Information, Menofia University, Egypt since 2004. He has worked on a number of organizations. He has contributed more than +30 technical papers in the areas of neural networks, Database applications, Information security and Internet applications.

ElsayedSallam, Emeritus Professor at Computers and Control Engineering Department-Faculty of Engineering-Tanta University, Egypt. His B. Sc. degree taken from Faculty of Engineering, Menofia University, Egypt at 1977.M.Sc. and Ph.D. degrees taken from University of Bremen, German at 1983, and 1987, respectively. Dr. Elsayed was head of the Computers and Control Engineering Department-Faculty of Engineering-Tanta University from 2008 till 2014. He works as a Manager at Management Information Systems (MIS) Project– Tanta University, Egypt, from August 2008 until December 2011. Then as a CIO - Tanta University, Egypt, from December 2011until November 2014.