Managing the Google Web 1T 5-gram with Relational Database

Yan Chi LAM
Faculty of the Graduate School of Global Studies
Tokyo University of Foreign Studies
Tokyo, Japan

ABSTRACT

On Sep 19 2006, Google released *Web 1T 5-gram*, an n-gram corpus generated from a source of approximately 1 trillion words. It provides a valuable reference of English usage since there is no other comparable corpus of this data size. However, it has not been widely used in language education due to the difficulty in managing the huge data size. In this paper, a practical approach of using relational database to store, index and search the corpus is described and implemented with commodity hardware. Basic search queries are also designed for performance testing. Sample performance results are recorded which show acceptable data processing and search response times. It is shown that the 5-gram corpus can be managed using relational database and commodity hardware. Further search queries can be designed and implemented to make better use of the corpus in language education.

Keywords: Google Web 1T, 5-gram, N-gram, Mysql, Corpus, Relational Database, Language education

1 INTRODUCTION

The use of corpora in language education has been widely discussed in publication such as *Rethinking language pedagogy from a corpus perspective* [1]. As mentioned in one of the paper in [1] by Aston [2], the use of corpora in teaching languages take into account the frequencies and characteristics of language usage by native speakers which are ignored by traditional syllabus and teaching materials. In that sense, the bigger the corpus size, the better the representativeness of the language usage. Recent technology has already allowed researchers to harness the resources of corpora with notable sizes such as *The British National Corpus (BNC)* [3] containing 100 million words and *The Corpus of Contemporary American English (COCA)* [5] containing more than 385 million words.

On Sep 19 2006, Google released an English corpus, *Web 1T 5-gram Version 1* [6]. It contains English word n-grams and their observed frequency counts. The length of the n-grams ranges from unigrams (single words) to five-grams. The n-gram counts were generated from approximately 1 trillion word tokens of text from publicly accessible Web pages. Its data size is about 10000 times bigger than BNC and about 2500 times bigger than COCA. It provides a unique reference of global English language usage since there is no other comparable corpus of this data size. Here is an overview of its data sizes:

Number of tokens:	1,024,908,267,229
Number of sentences:	95,119,665,584
Number of unigrams:	13,588,391
Number of bigrams:	314,843,401
Number of trigrams:	977,069,902
Number of fourgrams:	1,313,818,354
Number of fivegrams:	1,176,470,663

Physically the data are distributed in 6 DVDs, as gzip'ed text files. Each gzip'ed text files contains exactly 1,000,000 grams or less and their frequency counts except for the unigram file which contains all of the unigrams. All the raw data amount to around 25GB in gzip'ed format.

In this paper it will be described in details how the Google 5-gram corpus can be stored and organized using relational database (RDB) with common commodity machine hardware. Two kinds of search queries are implemented to demonstrate the feasibility in running searches on top of RDB. Results and performance will be discussed.

2 RELATED WORK

There are a few researches related to managing and extracting data from the Google N-gram corpus that are found for references. Their main purposes of using the corpus are for NLP tasks. Here is a summary of

their approaches in handling the corpus:

Research	Strategies
Hawker etc. [7]	- hash-based strategy that pre-process queries and/or data - reducing the resolution of the data to give only approximate frequency counts and sometimes false positive counts - data compressing
Islam etc. [8]	- only 5-gram data are processed - reducing the size of the data set by deletion and substitution of grams - sorting data into different files based on query word as indexing strategies
Sekine [9]	- customized trie indexing - index all of the 5-grams using a index file 277GB of size

NLP tasks involve numerous statistical queries on the data. It may justify the approaches of designing complex indexing methods and softwares, and sacrificing the accuracies of the data as they aim to return query results within a faction of a second.

However, for usage such as language education, such approaches can be redundant as time factor is not as essential and priority should be put in the ease of setting up the system, the flexibility of designing queries, and the ability to browse accurate data. Under such conditions, it justifies more to use existing RDB softwares in handling the corpus for language education as they have readily available internal storage and indexing functionalities that can be leveraged. This paper will explore the practicability and feasibility of such means.

3 PROPOSED APPROACH

This section will propose in abstract terms how the corpus can be processed and organized into a RDB and afterwards be indexed by it.

3.1 Data Modeling

In order to efficiently store and index all the n-grams data into a RDB, each of the unique English words in the corpus is given a numeric *word_id* since storage and indexing of *integers* require less space and execute faster compared with *strings* data type. The following relational data models are proposed:

Unigrams Table

Field Name	Data Type	Description
word_id	*integer*	A unique id ranging from 1 to 1,024,908,267,229 identifying the English word
word	*string*	The English word
frequency	*integer*	The frequency count of the English word

**All columns are to be indexed by the RDB*

Bigrams, Trigrams, 4-grams, 5-grams Tables

Field Name	Data Type	Description
gram_id	*integer*	A unique id identifying the gram instance
word1_id	*integer*	The corresponding *word_id* of the first word in the gram according to the Unigrams table
...	...	...
word(n)_id	*integer*	The corresponding *word_id* of the *n*th (up to 5) word in the gram according to the Unigrams table
frequency	*integer*	The frequency count of the gram

**All columns are to be indexed by the RDB*

The assignment of *word_id*s should be done when creating the Unigrams table. Considering the large scale of the data, the following problems may arise if each sets of the two to five grams is stored into one single table:

- The actual file used by the RDB software to store the table may exceed the maximum file size of the underlying operating system
- The number of entries in a set of grams (e.g. 4-grams has 1,313,818,354 entries) may exceed the limit of the maximum number of rows in a single table of the RDB software
- If the index size of a single table is too big, the index may not load or effectively load into the RAM, affecting search speed

Thus, each of the two to five grams tables is split into smaller tables to avoid the mentioned problems. The optimal way to split the tables depend largely

on the architectures of the hardware, the operating system and the RDB software. Since the aim of this paper is to explore the feasibility of using RDB to handle the data rather than how to use RDB to handle the data optimally, a naive splitting method is proposed here. Each sets of the two to five grams is split into the same number of tables as the number of raw text files containing the whole set. E.g. The set of 4-grams come in a total of 132 text files so the set of 4-grams will be split into 132 tables accordingly with each table holding the data of one of the text files.

3.2 Search Queries

Two kinds of queries are proposed here to serve the purpose of demonstrating the feasibility of searching the corpus processed into the proposed data models.

1. Exact Query
Two to five words or the special wildcard character * are to be input. The number of words and wildcards together are taken as the grams to be searched. All matching instances are returned sorted in descending frequency order. E.g. If "Apple *" is the input, all bigrams will be searched and all instances with the first word matching "Apple" (case sensitive) and the second word matching anything (wildcard) will be returned sorted in descending frequency order.

2. Keyword Query
Two to five words and the number of grams to search are to be input. Then any instances in the specified grams to be searched containing all of the keywords are returned in descending frequency order. E.g. If "apple tree" is the query and the search is specified to 5-grams, then all matching instances of 5-grams containing both the word "apple" and "tree" (case sensitive) will be returned in descending frequency order. Moreover, another optional wildcard * can be used in between words. E.g. If "apple * tree" is the query and the search is specified to 5-grams, then all matching instances of 5-grams containg "apple" as the first and "tree" as the last word with be returned.

These two queries are for demonstrating possible usages of the data and are not designed for any specific purposes. Many other possible queries can be further designed and implemented to extract data from the 5-gram corpus for specific purposes in language education but they are out of the scope of this paper.

4 IMPLEMENTATION

4.1 System Setup

Hardware and OS
In this research two machines are used. Their specifications are as follow:

	Development Machine	Server Machine
CPU	Intel Core(TM)2 Duo CPU E8400 3.00GHz	Intel Xeon Quad-Core E5506 2.13GHz
Memory	4GB	8GB
Harddisk	200GB	1TB
OS	Ubuntu 9.10 64-bit Server	Ubuntu 9.10 64-bit Server

The development machine's specification is common to most desktop machines. It is used for developing the scripts and codes before deployment and for comparison of speed with the server machine. The server machine is for final deployment and physically holds the database that contains all the data in the 5-gram corpus.

RDB and Programming Language
Mysql [10] is a free, open source, popular, easy to set up, and stable RDB software. Mysql version 5.0 is used in this research. Python [11] is an expressive interpreted programming language which provides good balance between coding time and execution speed. Python version 2.6 is used in this research.

4.2 Data Processing

First, the Unigrams table is created according to the data model described, assigning a *word_id* to each of the English words. Then, algorithm 1 is used for reading each n-gram raw text files and inputting them into Mysql.

The mapping of the English words to their *word_id*s and the insertion of data into the Mysql table are the heaviest tasks in this process. The mapping is done using an on memory cache of Python data structure dictionary to make it fast. The cache holding the mappings of all words implemented by Python dictionary takes up about 1.7GB of RAM.

It is essential that the insertion into Mysql table is done in a batch to minimize the overhead of each insertion calls to Mysql. The *INSERT* statement in Mysql supports multiple rows insert in one SQL command. Batch size of 10000 table rows is used in this implementation.

Indexes are to be created after all the insertion of one file instead of during insertion or it will slow the process down. Locking the table during insertion gives

a better performance.

Algorithm 1 Processing a two to five grams text file into a table in Mysql

Require: Unigram file, one of the n-gram files, Mysql connection

1: Create an empty Python dictionary data structure *cache*
2: $i \leftarrow 0$
3: **for** each English word in the unigram file **do**
4: *cahce*[*word*] $\leftarrow i$ {Assigning an ID to the word. Same assignment is used in creating the Unigrams table.}
5: $i \leftarrow i + 1$
6: **end for**
7: Create a Mysql table to hold the data according to the data model
8: Lock the table for faster insertion
9: **while** Lines can be read from the n-gram file **do**
10: *batch* ← Create an empty data structure (e.g. array or list) for temporary storage
11: *lines* ← Read as many as 10000 lines from the file
12: **for** each *line* in *lines* **do**
13: Split *line* to get individual words in the gram and its corresponding frequency
14: Use the *cache* dictionary to get the *word_id*s for each words in the gram
15: Save all the *word_id*s and the frequency count into *batch*
16: **end for**
17: Insert all data in *batch* in a single batch into the Mysql table
18: **end while**
19: Unlock the Mysql table
20: Create index on each columns in the Mysql table

After processing all the raw n-gram text files, the Mysql database contains the following tables:

	No. of Tables	Physical Size
Unigram	1	1.3GB (data: 463MB, index: 878MB)
Bigram	32	19.7GB (Each tables - data: 201MB, index: 430MB)
Trigram	98	73.3GB(Each tables - data: 239MB, index: 527MB)
4-gram	132	116.1GB (Each tables - data: 277MB, index: 624MB)
5-gram	118	119.4GB (Each tables - data: 315MB, index: 721MB)
Total	381	329.8GB

4.3 Search Queries

Algorithm 2 and 3 describe how the exact and keyword searches are implemented respectively.

Algorithm 2 Exact Search

1: *query* ← Get user input
2: Parse *query* to get individual words and wildcards
3: $n \leftarrow$ the total number of words and wildcards
4: Query the Unigrams table to get the *word_id*s for all the words in the query
5: *table_stacks* ← Create an empty data structure (e.g. array or list) for holding temporary Mysql table data
6: **for** each n-gram tables **do**
7: Execute an SQL query to return only the first instance in descending frequency order matching all the *word_id*s in the right word positions
8: **if** result are returned **then**
9: Append the result, frequency count, row offset (which is 1 now) and table name in *table_stacks*
10: **end if**
11: **end for**
12: Sort *table_stacks* with descending frequency count
13: *cache* ← Create an empty Python dictionary to cache *word_id*s mappings
14: *result_set* ← Create an empty data structure to store results (grams and frequency sets)
15: **while** *table_stacks* is not empty **do**
16: *top_table* ← Pop the top table (highest frequency count), its cached offset and cached result from *table_stacks*
17: Replace the word ids in the cached result in *top_table* with actual words using *cache*, if the mappings are not found in *cache*, query the unigram table and cached them in *cached* for later use
18: Append the gram and frequency in *top_table* to *result_set*
19: Try to fetch a new row from *top_table* with the same matching condition
20: If fetched then, append the result, frequency count, row offset and table name in *table_stacks* and sort *table_stacks* by descending frequency count
21: **end while**
22: return *result_set*

Algorithm 3 Keyword Search

1: *query*, *ngrams* ← Get user input for keywords and grams to search
2: Parse *query* to get individual words and get their corresponding *word_id*s by querying the Unigrams table
3: *table_stacks* ← Create an empty data structure (e.g. array or list) for holding temporary Mysql table data
4: **for** each *n*-gram tables of *ngrams* **do**
5: Execute an SQL query to return only the first instances in descending frequency order matching all the *word_id*s in any word positions or positions that match with the wildcard criteria
6: **if** result are returned **then**
7: Append the result, frequency count, row offset (which is 1 now) and table name in *table_stacks*
8: **end if**
9: **end for**
10: Sort *table_stacks* with descending frequency count
11: Follow step 12 to 22 described in the Exact Search algorithm

5 PERFORMANCE

5.1 Data Processing

Development Machine

It takes around 150 seconds to insert and index a bigram text file into a Mysql table while it takes around 230 seconds for a 5-gram text file. Trigram and 4-gram files take more time than bigram but less time than 5-gram. Let us generously assume that the time to process one text file (there are totally 381) is 4 minutes, it would take 1524 minutes, 25.4 hours, only a little bit over a day to process the whole Google 5-gram corpus into Mysql and index them, with a commonly available desktop machine specification.

Server Machine

It takes around 210 seconds to insert and index a bigram text file into a Mysql table while it takes around 340 seconds for a 5-gram text file. Trigram and 4-gram files take more time than bigram but less time than 5-gram. The process takes longer in the server machine than the development machine probably due to the bigger overhead in utilizing a bigger RAM size and a bigger harddisk size. However, the performance can be largely compensated by running multiple processes in parallel to process several text files at the same time. In this research, up to four processes are running in parallel processing 4 different text files at the same time. Again, for easy calculations, let us generously assume that three parallel processes are run and the time to process one text file (there are totally 381) is 6 minutes, thus, the average time to process one file becomes 2 minutes. It would then take 762 minutes, 12.7 hours, only a little bit over half a day to process the whole Google 5-gram corpus into Mysql and index them.

5.2 Search Queries

Exact Search

The following table gives some examples of execution times of the wildcard search implemented. All of the query return within one minute which is very acceptable in querying a corpus of this data size. Second runs are much faster due to the caching mechanism of Mysql.

Query	Time taken to return the first 100 results (in seconds)	
	1st run	2nd run
"banana *"	0.4	0.1
"* banana"	11.9	0.5
"cake * * * *"	4	0.2
"* * cake * *"	33.6	1.6
"* * * * cake"	47.3	1.7
"day dream * *"	3.6	0.2
"day * * dream"	3.7	0.2
"* day dream *"	31.7	0.5
"* * day dream"	54.1	0.5

Keyword Search

The following table gives some examples of execution times of the keyword search implemented. Some queries take up to 6-7 minutes to return. Second runs are much faster due to the caching mechanism of Mysql. The search is now running in a sequential manner, querying Mysql tables one by one and does not take any advantage of the possibility of distributed computing. The way how the data models are proposed, the data can actually be stored across several servers in the same network running Mysql. By running the part from line 6-11 described in algorithm 3 in parallel across for example n machines, the speed would be shortened by close to n times theoretically.

Query	N-gram	Time taken to return the first 100 results (in seconds)	
		1st run	2nd run
"love"	2	45.2	0.5
"love"	3	209.3	1.7
"love"	4	371.6	2.3
"love"	5	394.8	2
"book library"	3	191.1	1
"book library"	4	314.4	1.5
"book library"	5	321.6	1.5

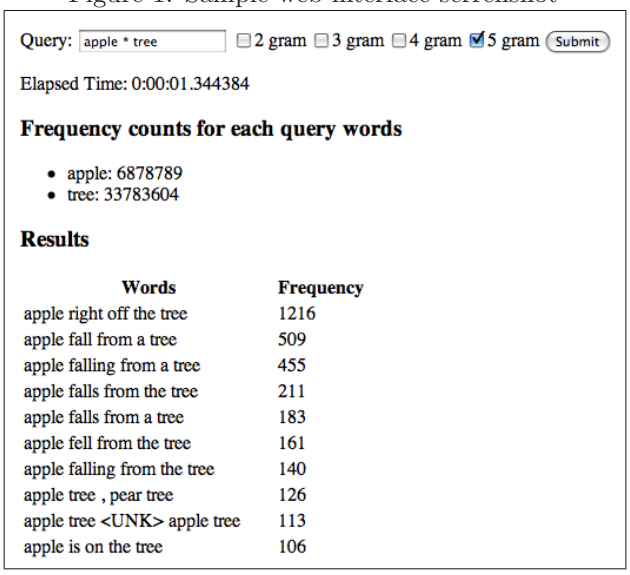
6 FUTURE WORK

The Google 5-gram corpus can serve as a valuable resource in language education. It is shown and documented in this paper how the Google English 5-gram corpus can be handled by using commodity machines leveraging the power of readily available relational database softwares. Furthermore, search queries are also implemented on top of the proposed data models to demonstrate the feasibility of designing useful searches. With this knowledge, the Google 5-gram corpus can now be set up easily and be examined, browsed and considered for use in language education.

Currently, web interface has been setup to allow teachers and students on campus to use the implemented search functions. Figure 1 shows a sample screenshot of the web interface. After more testing and usage data collection, more meaningful searches tailored to language education can be developed and search performance can be optimized according to actual needs.

Finally, as Google has also released n-grams corpora in Japanese and other European languages, the same way of handling data can be extended to those corpora and thus can benefit language education research in those languages.

Figure 1: Sample web interface scrrenshot



References

[1] Burnard, L., & McEnery, T. (Eds.). (2000). *Rethinking language pedagogy from a corpus perspective: Papers from the Third International Conference on Teaching and Language Corpora*. Frankfurt: Peter Lang.

[2] Aston, Guy (2000): *Corpora and language teaching*. In: Burnard, Lou & McEnery, Tony (eds), 7-17.

[3] *The British National Corpus, version 3 (BNC XML Edition)*. 2007. Distributed by Oxford University Computing Services on behalf of the BNC Consortium. URL: http://www.natcorp.ox.ac.uk/

[4] *What makes an Oxford Dictionary?* AskOxford.com. Oxford University Press. URL: http://www.askoxford.com/oec/mainpage/ Retrieved 13 Feb, 2010.

[5] Davies, Mark (2009), *The 385+ Million Word Corpus of Contemporary American English (1990-present)*. International Journal of Corpus Linguistics.

[6] Thorsten Brants, Alex Franz. 2006. *Web 1T 5-gram Version 1*. Linguistic Data Consortium, Philadelphia.

[7] Tobias Hawker, Mary Gardiner and Andrew Bennetts (2007). *Practical Queries of a Massive n-gram Database*. Proceedings of the Australasian Language Technology Workshop 2007. Melbourne, Australia, 10th–11th September, 2007, pages 40–48.

[8] Aminul Islam, Diana Inkpen. *Managing the Google Web 1T 5-gram Data Set*. Proceedings of the IEEE International Conference on Natural Language Processing and Knowledge Engineering (IEEE NLP-KE'09). Dalian, China, September, 2009.

[9] Sekine, Satoshi (2008). *A Linguistic Knowledge Discovery Tool: Very Large Ngram Database Search with Arbitrary Wildcards*. Proceedings of Coling 2008: Companion volume: Demonstrations. Coling 2008 Organizing Committee. Manchester, UK. Pages 181-184

[10] http://www.mysql.com/

[11] http://www.python.org/