

CS 5621 – Computer Architecture
Needles in a Gigastack Project
Group Tulum (Maria, Chris, Sumon, Matt)
Beta stage paper
Due: April 10, 2008

(Maria Sinn) Intro:

The “Needle in a Gigastack Problem” involves solving a counting problem. The goal is to count the number of distinct m to n word terms. An example of this would be, $m=2$, $n=3$, “The duck quacks loudly everyday”

The 2 word terms are:

“The duck”

“duck quacks”

“quacks loudly”

“loudly everyday”

The 3 word terms are:

“The duck quacks”

“duck quacks loudly”

“quacks loudly everyday”

The only difference is that in this project we want to count m to n word terms in a large corpus called the Gigaword Corpus. We want to keep track of distinct terms and the number of times that those terms occur. Meaning if the one word term “the” occurs 1million times we want to keep a count of the occurrence of that term. Then the goal is to be able to rank these terms from the most important to the least important. Importance can be described as the occurrence of the term within the corpus. This is done using the TF*IDF equation.

The “Needle in a Gigastack problem” relates very much to issues in the computer security field. When usernames and passwords are stored in a database each is typically stored in a very large file. It is essential that usernames are unique so that one user isn’t accidently mistaken for another. The “needle” could be compared to a username, and that user name is a one-word term. When a new user registers for an account for some organization they have to ensure that their username is unique from all other usernames in the database.

Uniqueness and randomness of passwords is a necessity for an organization to provide more security for their consumers. Since a password is linked to a username this combination is best if it is unique from another username/password

combination. When a new user figures out their password it is stored in a large file. It is typically stored in its encrypted form. So when a user signs in using their username and password this combination isn't confused for someone else's combination. The user probably doesn't care if their password or username is unique. If a username/password combination is unique a hacker will have much difficulty using a dictionary to attempt to steal the users credentials. Then if the hacker does manage to find a username, if the password is very unique, meaning it is very random they will have a difficult time cracking the encryption.

(Maria Sinn)References:

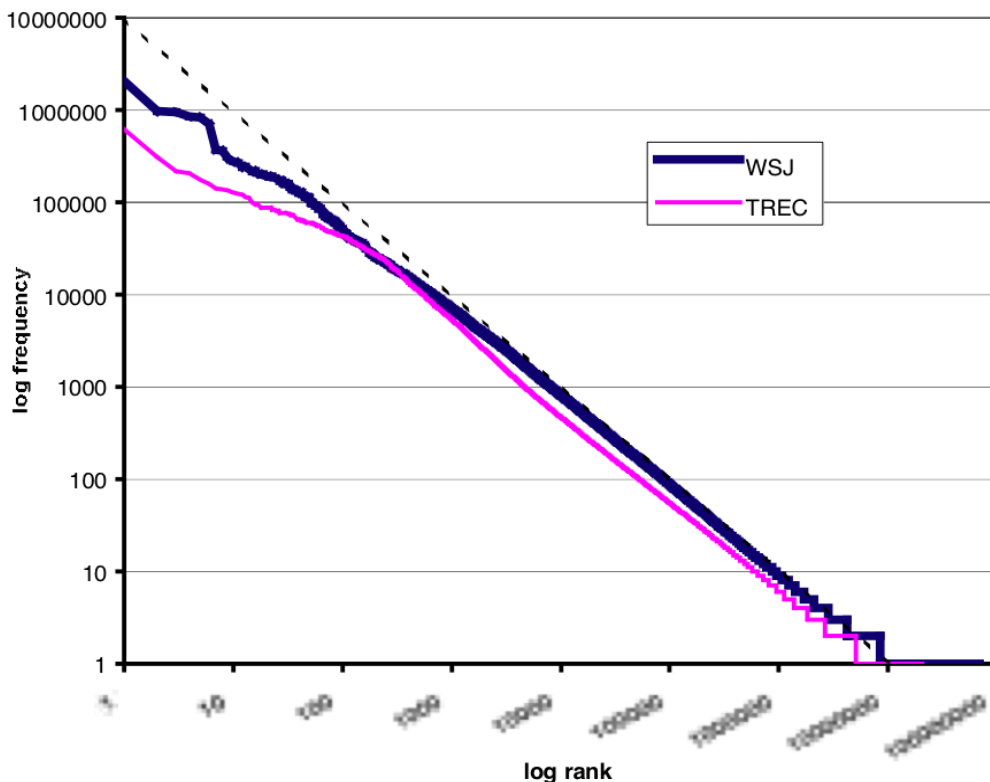
We have found two articles from the ACM Digital Library which relate to the project that we have been assigned: "Exploiting Parallelism in Pattern Matching: An information Retrieval Application," (Mak, Wing-kit) and "Extension of Zipf's Law to Words and Phrases. (Ha, Le)"

In the article: "Exploiting Parallelism in Pattern Matching: An information Retrieval Application" (Mak, Wing-Kit) a proposal is made to develop a "document searching architecture based on high-speed hardware pattern matching to increase the throughput of an information retrieval system." There is also a proposal made to develop a new parallel Very Large Scale Integration pattern-matching algorithm. The architecture is that of a single master that controls a set of slaves. In this article they also discuss the idea of the slaves continuing to work until they reach an "END_OF_DOC detected." (Mak, Wing-kit) If there is a match to the end of document condition then the search will end. Once their slave is to the end of a document it will send its findings to the manager. If there is another document to search the manager will send out a document otherwise the slaves is done. This is similar to what we have chosen to use for our project. We are using the Manger/Worker Paradigm and this exploits parallelism in the same fashion as this article explains, we have a single manager processor and a large set of worker processors. This situation is parallel since the Manager does minimal work and the workers finish their job and communicates their findings with the Manger. We are using an end of file tag to detect the end of a file. Once a worker reaches the end of the file they will send their count of articles and words to the manager, then the manger sends another file or a termination message to the worker dependent on the number of files left and the number of workers looking to count a file.

The second article that we are referencing involves Zipf's law. Zipf's law states that the frequency of word tokens (a word or string that is unique compared to the rest of the text in a corpus) in a large corpus of natural languages is inversely proportional to the rank (the frequency in which word tokens occur). Zipf ranked word tokens in decreasing frequency order. When a single word and a phrase or another word are combined together in one list and put in order of frequency the combined list follows Zipf's law accurately for all words and phrases. Zipf's law works with a few constants to calculate the frequency of words and phrases. The equation below is a combination of Zipf's discovery and a modification made by Mandelbrot, the modification allowed the equation to be accurate for large corpora.

$$f = k / (r + \alpha)^\beta$$

f is the frequency of a word in the corpus, and α and β are constants for the corpus that is being analyzed. In the 1980s Zipf's law was used to process large corpora with 1 million words or more. This article is trying to disprove Zipf's law, using two different languages English and Mandarin Chinese. They failed to disprove Zipf's law for large corpora; they tested Zipf's law with articles from the Wall Street Journal for 1987, 1988, and 1989, with sizes approximately 19 million, 16 million, and 6 million words respectively. The Mandarin corpora was obtained from the People's Daily Newspaper from 1991, 1993, 1994, and 1995 it had just over 19 million words and they also sourced documents that are similar to syllabi which contained around 2.1 million words. This article seems very similar to the Needle in a Gigastack problem that we are working on for this course. They used a pretty large corpus and we are using a corpus just as big if not bigger and the ultimate goal of this project is to find terms that are "important." Then we are to rank them in highest order of importance. The goal of this project is very similar to the findings of Zipf and Mandelbrot. The differences appear to be the equations that are being used to calculate the importance of a term, and possibly the size of the corpus. The graph below shows the difference between English and Mandarin and their rank of words and phrases from lowest importance to highest importance. The difference is fairly minimal. WSJ stands for Wall Street Journal and TREC, which stands for Text Retrieval Conference. The Peoples Daily Newspaper is part of the TREC.



Amdahl's Law (Matt Wronski):

Amdahl's law is based mainly on the amount of code that can be run in parallel, however, knowing the amount of code that is parallelizable or not depends a lot on the time that it takes for every specific function call to implement. This is why I believe that it would be most beneficial to use Amdahl's law with our run times in order to compute the amount of parallelizable code we have through using Amdahl's law. Then we would have a good estimation of the time it takes for our manager to send to workers which is non-parallel the first time that it sends and parallel the next time it is sending to one processor and no other processors are forced to wait for their turn to receive. You can see that the number of waiting processors would be equivalent to the percentage of parallelizable sending, because if all are waiting then it sends serial to one processor at a time and when all are working but one processor who is receiving, then the sending task is being done in parallel, or at the same time that the problem is being worked on by all other processors. So the amount of code that is running in parallel will depend on the number of processors and the size of the data that is being worked on. This will make Amdahl's law come in handy still to estimate speedup, as the number of processors is increased, once we have used some benchmark times to estimate with Amdahl's law the percentage of code that is running in parallel. We have not been able to get any test run times which would allow us to use Amdahl's law properly, because we were unable to run the program in time. We will, however, use Amdahl's law to predict the speedup per processors graph line once we obtain proper runtimes.

Design (Chris Saxton):

For this project we decided to implement the manager worker paradigm. We chose this paradigm because we have made it exploit parallelism with the use of multiple files containing large bodies of text.

The manager will be responsible for opening each file. The manager then sends a pointer to the worker, breaking down each file line by line. Parallelism is exploited here because workers are working on their specified part of the file. The worker counts the total number of words in a specific line and adds the total to its total count of words. The way the work is broken up allows the program to be independent from the number of files, since every worker works on every file. We will store the terms in a series of arrays. Each element in the array will represent the letters A to Z and contain pointers to another array representing the letters A to Z. At the end of each array there will be a count variable. The last letter of a term will add one to the current value of count. The goal of this data structure is to reduce search time for a specific term. For example if a term is 20 characters long the worker will only have to traverse at most 20 characters to see if that term has already been found. The idea of this data structure is to compress the data of the files when counting.

We struggled with this data structure so we ended up using more of a brute force approach. Our new method stores each and every distinct term found in the corpus. To help preserve distinctness across processes we assigned each process a range of letters to analyze. For example one process would receive all terms that begin with "TH", this way no two processes will count the same term. Each worker would calculate the TF*IDF value for all their terms, and send the top R terms to the manager. The manager would then find out the top R terms over all and output them to the user.

Implementation (Sumon Rahman):

The implementation of the project at the beta stage begun with the division of tasks to complete the project. As stated in the strict project guideline, all of us decided to take part in both in the development of the algorithm and coding, proof reading and writing the report. We discussed different options for the use of various data structures to be able to "remember" terms the processors have already seen. We also decided to change the way tasks were divided among the processors in the alpha stage of the project. The current version which is described in the design section of this report takes advantages of the added processors and also scalable to different number of files. Matt took responsibility of developing the data structure and the code to manipulate it, while Chris developed the manager process. Worker process was written by Maria and I developed the main function.

Works Cited

- Mak, Victor Wing-Kit, Kuo Chu Lee, Ophir Frieder.
"Exploiting Parallelism In Pattern Matching: An
Information Retrieval Application."ACM Transactions on
Information Systems 9.1 (1991): 52-74.
- Ha, Quan Le, E.I Sicilia-Garcia, Ji Ming, F.J.Smith.
"Extension of Zipf's Law to Words and Phrases."
Proceedings of the 19th International Conference on
Computational Linguistics volume 1 (2002): 1-6.