PageRank by Distributed Computing: An Empirical Analysis

Kirthi Venkatraman[*]

ramank@quotient-inc.com

ABSTRACT

The World Wide Web since its inception has grown so rapidly that it poses unprecedented challenges in computing the *PageRank* for what has been called "the worlds largest matrix computation" [18]. There are some novel algorithms for fast computation of *PageRank* calculation that is already available, but the size of the problem domain today challenges us to think of a distributed model to solve this problem with very minimal memory constraint. Furthermore, modern distributed computing has come a long way and today we are equipped with better technology and therefore a distributed approach seem plausible.

In this paper, we present a distributed model that attempts to solve the *PageRank* computation and subsequently other ranking methods for sufficiently large number of pages. We analyze empirically how the iterative values converge faster, and with the results of this analysis we report on the use of a distributed approach which can be further enhanced and extended on a larger scale. The preliminary experiments on both simulated data and real data set demonstrate that the system achieves encouraging results that are reasonably accurate. Comparison for various data samples are discussed and variations for this ranking are elaborated as well.

Categories and Subject Descriptors

G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra; H.2.8 [**Database Management**]: Database Applications—*Data Mining*; H.3.5 [**Information Storage and Retrieval**]: Online Information Services —*Web-based services*

General Terms

Algorithms, Performance, Experimentation

[*]Kirthi currently coordinates Quotient's Information Retrieval Research Studies and specializes in Cryptography and Information Retrieval. He has an MS in Mathematics and MTech in Computer Science from the Indian Institute of Technology, Delh. Kirthi currently works at National Library of Medicine. (May 2005)

Keywords

Distributed Processing, Google, Link Analysis, PageRank, Distributed Computing, Empirical Analysis, World Wide Web

1. INTRODUCTION

The Web, as it grows today, has allowed global information exchange on a scale unprecedented in human history. Estimated Web pages rose from *500 million* in the year 2001 to a staggering *11.5 billion* [9], including content that shares research on medicines, diseases and their cures (if any), emotional experiences, political ideas, cultural customs, musical idioms, business advice, artwork, photographs, and literature. Due to this explosion in size, web search engines are becoming increasingly important for locating information. Given the vast amount of information on the World Wide Web today, on the order of over billions of pages, a typical short query such as *Text Mining* submitted to the *Google* search engine yields **53,200,000** web pages. Despite the fact that search engines like Google only crawl and index 35% of the entire World Wide Web, computing *PageRank* on such a volume today, we face an arduous task and it's complexity will increase much more in the future as the volume of the content grows at this rate.

Most current engines perform ranking through a combination of term-based techniques, link-based techniques, and user feedback or popularity rank. The ranking factors in web search can be divided into two categories. The first is the content-based relevance from traditional information retrieval, and the second is link-based authority and a significant amount of research has been focused on link-based ranking algorithms, i.e., techniques that use the graph structure of the web to identify interesting relationships between the pages. Among the most popular algorithms are *PageRank* [5] which has a record number of 596 citations, Topic-Sensitive PageRank [13] and their variations, HITS [16], and SALSA [17], which have been demonstrated to be successful in many web information retrieval applications.

The accuracy of ranking algorithm is also important besides the complexity, because ranking web pages such that useful and relevant ones appear in the reasonably meaningful order of their rank is crucial in the Web Information Retrieval today.

There are several issues in the ranking algorithm and we address a few of them which are known and list some resolutions that have been made by several sources.

Search engines can be '*spammed*' by websites faking high relevance with respect to some topics for the purpose of attracting visitors. Common early techniques that were used to deviate the ranking included the use of inappropriate site titles or description, inclusion of META keywords in the HTML code, or in some cases even the use of extra text invisible to human surfers. Such an activity caught the attention of search engines such as *Yahoo* and *Google*, and it became an important issue to resolve, and soon there was a term called *spamdexing* in Wikipedia, which also meant *search engine spamming*. Combating Web Spam has been addressed by TrustRank [11, 10], a new technique proposed by Zoltn Gyngyi and Hector Garcia-Molina of Stanford University and Jan Pedersen of Yahoo! to semi-automatically separate reputable, good pages from spam. SpamRank [1] is another solution that is based on fully automatic link spam detection by Mate Uher et al. at Computer and Automation Research Institute, Hungary.

Identifying content replication and knowing the original author or the source of the content was a problem, because there were too many replication of contents, and co-location sites. National Information Standards Organization has proposed a syntax to create web-transportable packages of metadata and/or identifiers about the content and their authors[6]. Previous studies [12, 8] that describe efficient methods to construct a similarity index out of chosen strategies is a good solution and yet an expensive task.

In fact, the ranking also depends on how the crawling is done. There are several limitations of a general purpose web crawler and therefore in this study, we focus on a combination of selective, focused [22, 14] and distributed crawling, by choosing a pre-defined set of URL's as seed and using some scoring function with respect to some relevance criteria. We specifically narrowed down to focus in the context of medicine, diseases, biotechnology and biomedicine. There were some challenges and many interesting results that were observed during our crawling process, but we reserve those discussions for future work.

In this study, we choose an incremental approach with two steps. First we chose relatively small sample of URL's upto 65,536 web pages on a multi-threaded single process model and studied the results. Second, we chose a much larger sample of about 600,000 web pages on a distributed environment using a Sun Fire 280R with 2GB of memory, a Sun Ultra 10 with 512 MB memory and compared the results in both these approaches.

1.1 Motivation

The most interesting trend in search engines is the growing sense of the size and their limits, and simultaneously there are numerous smaller and more focused search engines that have been evolving [3]. The reasons for existence of such variety of choices today has been very well explained by Naor et al. [7], but the first motivation for this study was to seek a User-Centric Search Paradigm in a restricted context of medicine, biotechnology, and diseases. Even after narrowing down to these specific areas, the volume of content still remains very high and ranking these content is not straightforward. For the present, link analysis [5, 16] and many variations of these have been some popular algorithms. However, the fact that the web is evolving computing *PageRank* is too massive, improvement in this computation is imperative.

As a first step, our focus was to study and evaluate a ranking mechanism that is topic-sensitive [13], and also utilizes the user feedback and several such parameters, but before that we set to study the pagerank algorithm and evaluate possible distributed model that is not expensive in terms of both CPU and Disk Usage. We chose to use compression/decompression techniques to save disk space and extrapolation methods [15] to efficiently accelerate the computations, which will be elaborated more in the later sections.

The remainder of the paper is structured as follows. In Section 2 review the *PageRank* algorithm and iterative methods to compute them and we explain the method that is better approximation. In Section 3 describes the data set that we worked on for our experimental results. In Section 4 we describe a models to solve this computation in a distributed model. Section 5 elaborates the numerical experiments that were conducted and their results. Section 6 summarizes the conclusion and discusses related theories and future work.

2. REVIEW OF PAGERANK

In this section we summarize the definition of *PageRank* [5] and analyze how we will use the iterative nature of this algorithm and propose various distributed models to compute *PageRank*.

Google users *PageRank*, to measure the importance of web pages, which is based on the linking structure of the Web, in which it is considered as a model of user behavior, the "Random Surfer Model", where a random surfer visits a web page with a certain probability which is derived from the sum of the ranks of the parent page. In other words, if $q_1, q_2 \cdots q_n$ are all the pages that link to a web page p, and if $d(q_i)$ is the number of outgoing links of a page q_i,

$$r(p) = (1-d) + d\left(\frac{r(q_1)}{d(q_1)} + \cdots + \frac{r(q_n)}{d(q_n)}\right) \qquad (1)$$

where d is usually chosen to be 0.85 or 0.9. The random surfer browses the web by going from one page to the next. Given the current location of the surfer, the successor location is a page that can be reached by following the link from the current page uniformly at random. In addition, if the surfer reaches a page without any out-going links, there is no place to click and therefore the user jumps at random to any other page.

The above equation can be re-written in Jacobi iteration form as shown below requiring us to compute at iteration k

$$r_i{}^{(k)} = (1-d) + d \sum_{(i,j)\in E} \frac{r_j{}^{(k-1)}}{d_j} \ \forall\, I \qquad (2)$$

where $r_i{}^0 = \frac{1}{n} \quad \forall\, 1 \le I \le n$.

A simple and intuitive remedy for dangling node (nodes that has an outdegree of zero) is to modify the model by picking up a random page for the next state [24], which then leads to the following modified form

$$r_i{}^{(k)} = (1-d)\frac{R^0}{n} + d \sum_{(i,j)\in E} \frac{r_j{}^{k-1}}{d_j} \ \forall\, I \qquad (3)$$

where n is the total number of pages and $R^0 = \sum_p r^0(p)$ is the amount of rank initially inserted and d_j is the out-degree of the page j. Gauss-Seidel iteration (4) on the other hand is guaranteed to be faster and saves at least 40% of iterations.

$$r_i{}^{(k)} = (1-d)\frac{R^0}{n} + d\sum_{j<i}\frac{r_j{}^k}{d_j} + d\sum_{j>i}\frac{r_j{}^{k-1}}{d_j}\ \forall\ I \quad (4)$$

However, it has an overhead of sorting the adjacency-list representation of the Web graph, so that back-links for pages, rather than forward-links are stored consecutively. Hence this method is very difficult to parallelize.

2.1 Extensions of PageRank

Interestingly enough there are some extended ideas [21] that enhances PageRank to the extent that the title [4] *Back-Rank: an alternative for PageRank?* is worth exploring. This extended idea is based on the model for *Back* button, which improves the random surfer model.

In this study, we have not attempted to experiment with these results. However, it will be interesting to see the future improvements of BackRank.

2.2 Extrapolation Methods

Extrapolation methods are based on the expansion of iterates $r^{(k)}$ in a series

$$r^{(k)} = \sum_I \lambda_i^k u_i \quad (5)$$

where terms u_m are eigenvectors and $\lambda_1 = 1$ (The first eigenvalue in a Markov chain is 1). These extrapolation methods try to improve approximation of the solution u_1 closer and closer to $r^{(k)}$ through suppression of the higher terms. Extrapolation methods are probably the most efficient and powerful methods for solving power iterations and the resists are more accurate based on the assumptions, and specifically in the above case it depends on the amount of truncation.

Mathematically, the quality of extrapolation methods are limited by the assumptions that are made. In these extrapolation methods the assumptions are that $r^{(k-2)}$ can be expressed as a linear combination of u_1 and u_2 and $1 < \lambda_2 \le \lambda_3 \le \cdots \le \lambda_m$ and so on.

With these assumptions, while computing the iterations, we can construct a linear combination of several iterates that exactly eliminates one or more harmonics after the very first term and regard the remaining tail as relatively small approximations that can be ignored. The Aitken's method [15] aims at eliminating the second term. By truncating the tail after two terms, we get

$$r^{(k-2)} = u_1 + u_2 + \cdots \quad (6)$$

$$r^{(k-1)} = u_1 + \lambda_2 u_2 + \cdots \quad (7)$$

$$r^{(k)} = u_1 + \lambda_2^2 u_2 + \cdots \quad (8)$$

These are three equations with three unknowns, and we can therefore eliminate two of them, u_2 and λ_2. If we define [15] g_i and h_i as

$$g_i = (r_i{}^{(k-1)} - r_i{}^{(k-2)})^2 \quad (9)$$

$$h_i = r_i{}^{(k)} - 2r_i{}^{(k-1)} + r_i{}^{(k-2)} \quad (10)$$

Then by simple algebra and solving for u_1, we get

$$u_1 = r^{(k-2)} - \frac{g_i}{h_i} \quad (11)$$

where $|h_i| > 0$. In other words, with the three iterates $r^{(k-2)}$, $r^{(k-1)}$ and r^k, an approximation to the new iterate is evacuated as shown in (11).

Quadratic extrapolation is a generalization of Aitken's method that is based on a similar set of equations that is truncated after three terms.

$$r^{(k-3)} = u_1 + u_2 + u_3 + \cdots \quad (12)$$

$$r^{(k-2)} = u_1 + \lambda_2 u_2 + \lambda_3 u_3 + \cdots \quad (13)$$

$$r^{(k-1)} = u_1 + \lambda_2^2 u_2 + \lambda_3^2 u_3 + \cdots \quad (14)$$

$$r^{(k)} = u_1 + \lambda_2^3 u_2 + \lambda_3^3 u_3 + \cdots \quad (15)$$

Remember that these results are only best approximations and since we are dealing with massive graph size, any such approximation would be a great help. Sepandar et al. has elaborated [15] thorough details about these methods and their convergence.

From our empirical results we found that this extrapolation works well as predicted with larger graphs when compared among various data sizes.

2.3 Stopping Criteria

There are various methods to measure and determine a stopping criteria [5, 15], and empirically we found that the the L_1 norm of the residual vector, i.e.,

$$||r^{(k+1)} - r^k||_1 < \epsilon \quad (16)$$

is the simplest stopping criterion for iterations, although more relevant approach would be based on an estimation of how the resulting ranking is affected.

The ranking order defines a permutation σ over the pages. In order to evaluate between the ranking order between two iterations, one has to measure the correlation, and Kendall's tau K and Spearman's rank correlation methods are a few well known methods. If we assume that $\sigma(I)$ be the rank of page I, Kendall's tau K distance measure is defined as

$$K(\sigma_1, \sigma_2) = \#\{(i<j) \mid sgn(\frac{\sigma_1(i) - \sigma_1(j)}{\sigma_2(i) - \sigma_2(j)}) = -1\} \quad (17)$$

This is also known as bubble-sort distance, because it counts the number of swaps bubble-sort would make transforming σ_1 to σ_2.

Spearman's footrule F method measures the similarity of permutations as

$$F(\sigma_1, \sigma_2) = \sum_i |\sigma_1(i) - \sigma_2(i))| \quad (18)$$

The maximum values of F and K can be computed[2] and therefore these distances can be normalized. Kendall's tau K distance measure and Spearman's footrule [7] are classic dissimilarity measures known, but yet are more expensive than the distance measure.

In our analysis and experiment, we use a combination of iteration without approximation in first few iterations with a tolerance of $\epsilon = 10^{-4}$ and Aitken's extrapolation method

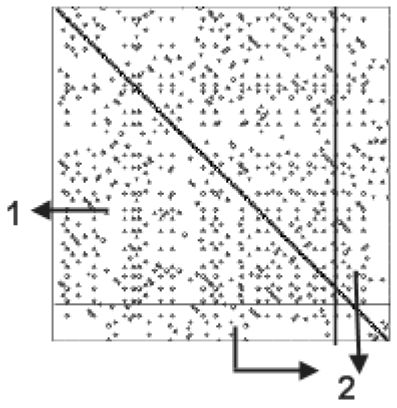

Figure 1: A sample subgraph with 8192 nodes.

for the rest of the iterations with a tolerance of $\epsilon = 10^{-7}$ as shown in (11) and (16).

3. DISTRIBUTED MODEL

3.1 Experimental Analysis

There are several good reasons for seeking various distributed model to compute PageRank. First, experience has shown specially in the past few years that the growth of content is growing exponentially. Centralized search paradigm such as *Google* and *Yahoo* are popular today, and even if a paradigm shift happens any time soon, it is imperative that we need to analyze and explore various distributed approach for ranking.

With an anticipation of the growth of content to increase at much higher rate, a distributed method that is based on scalable architecture is much needed. Our distributed approach depends upon various experiments that were conducted and therefore we describe some of out experiments here in the following section with examples first.

In order to analyze the results of pagerank computation and their convergence, it is easier to begin with a small data set. We therefore started our experiment on sample size $N = 8192, 16384$ and 65536 of web pages. Since it is known that the world wide web is not completely connected, there has to be some boundaries that separate these set of subgraphs that are connected, which as a union forms the world wide web. With this idea, we attempted to partition the web pages into two parts (to keep it simple). The first part (1) being more dense compared to the second part (2). One such example is shown in **Figure 1**.

3.1.1 Incremental PageRank Computation

A preliminary round of analysis conducted on some random graphs and some real graphs generated from breadth-first based crawler, produced some interesting results. A random graph with 8192 nodes that is fairly dense was used to run ranking algorithm first. Then this graph was extended to approximately $1/8th$ the size, by adding 1024 more nodes and the ranking algorithm was re-run. This second attempt used the ranks that were generated in the first run as initial values for those 8192 nodes and for the remaining new nodes that were added, an initial rank value of $d\sqrt{\frac{outdegree_i}{N}}$, where $d = 0.85$. The top 10 ranks are shown in **Table 1** for comparison (these ranks are normalized value by an approximate scale).

Table 1: Comparison of Selected 10 Ranks

Extended Run	Single Run	Difference
0.9965388850	0.9965388850	0.0
0.9965999877	0.9965999877	1.11 x 10^{-16}
0.9966465163	0.9966465163	2.22 x 10^{-16}
0.9965388850	0.9965388850	0.0
0.9965388850	0.9965388850	0.0
0.9965388850	0.9965388850	0.0
0.9965388850	0.9965388850	0.0
0.9966465163	0.9966465163	2.22 x 10^{-16}
0.9965388850	0.9965388850	0.0
0.9965388850	0.9965388850	0.0

While crawling by either *Breadth-First Search* or *Backlink Count*, if we adopt an efficient mechanism to keep track of the length of the backlinks, and dangling links, at the end of crawling we can reorder the graph such that those pages with higher backlinks appear first and dangling links appear last. In our analysis, we applied a correction to dangling nodes by creating a link to itself. In the following algorithm, G is the graph with nodes idexed 1 through n and 5 is the tolerance.

```
procedure $IncrementalRank(G, 1, n, t)$
1: $partition(G, G_1, G_2)$;where $G_1$ is dense
2: $default_i \leftarrow \sqrt{\frac{outdeg_i}{N}}$
3: $initRanks(G_{,1}\, default_i)\ \ \forall\ i$
4: $ranks(1, n_1) \leftarrow PageRank(G_1, t, default)$
5: $t_1 \leftarrow t$ x $10^{-3}$
6: $ranks(1, n) \leftarrow PageRank(G, t_1, ranks(1, n_1))$
```

3.1.2 Seeded PageRank Computation

Another interesting round of analysis was conducted on real data, by taking some chosen results from a query **'Text Mining'** on *Google* as seed and conducting Breadth-First Search crawling. We limited our crawling to a depth of 4 levels and 8435 web pages. In addition to these seeds, another set of seed URL were obtained from a query **'Coal Mining'** and **'Mining'** on *Google*. This choice of two sets of seeds were deliberate because we were attempting to test our theory that could prove useful to devise a distributed algorithm.

The PageRank computation for the two sets are assumed to follow *incrementalRank* approach. Consider the algorithm for merging the two graphs and comparing the ranks. In this algorithm, $PageRank(G, t)$ uses Jacobi iteration as described in Section 2. By using the merged ranks as initial values, the convergence is much faster even for a tolerance of 10^{-8}. The results of $MegedRank$ was compared with $PageRank$ and there were some interesting results. The top 10 ranks compare between MergedRank and a single run is shown in Table 2.

```
procedure $MergedRank(G_1, G_2)$
1: $t_1 \leftarrow 10^{-5}$
2: $default_i \leftarrow \sqrt{\frac{outdeg_i}{N}}$
3: $initRanks(G_1, default_i)\ \ \forall\ i$
```

Table 2: Seeded Computation: First Set

Seed URL
www.sims.berkeley.edu/ ∼ hearst/text − mining.html
www.cs.waikato.ac.nz/ ∼ nzdl/textmining/
www.cs.cmu.edu/ dunja/WshKDD2000.html
www.cs.utk.edu/tmw06/
www.cs.utk.edu/tmw03/
www.ima.umn.edu/reactive/spring/tm.html
www.nactem.ac.uk/

Table 3: Seeded Computation: Second Set

Seed URL
www.rootsweb.com/ ∼ wvcoal/
www.umwa.org/mining/colminrs.shtml
www.ncm.org.uk/
history.osu.edu/projects/gilded$_a$ge/
www.welshcoalmines.co.uk/
en.wikipedia.org/wiki/Coal$_m$ining
www.kingdomcome.org/museum/
www.beckleymine.com/

```
 4: $ranks(1, n_1) \leftarrow PageRank(G_1, t_1)$
 5: $initRanks(G_2, default_i) \ \forall i$
 6: $ranks(1, n_2) \leftarrow PageRank(G_2, t_1)$
 7: $ranks(1, n_1{+}n_2) \leftarrow merged(ranks(1, n_1), ranks(1, n_2))$
 8: $G(1, n_1 + n_2) \leftarrow merged(G_1, G_2)$
 9: $t_2 \leftarrow 10^{-8}$
10: $initRanks(G, ranks(1, n_1 + n_2))$
11: $ranks(1, n_1 + n_2) \leftarrow PageRank(G(1, n_1 + n_2), t_2)$
```

Interestingly enough for a query term **Mining the** on *Google*, resulted in a different response, for obvious reasons. Google's ranking currently has many parameters and a trivial set of them are title and body of the web page. A logical question here is 'what happens when a query term *mining* is used over the collection of the two sets here?', and since we encountered more web pages in the first set than the second, the PageRanks from these two sets can be used to apply some weighing criteria on the terms used in query and the search results.

There are two novel ideas here. First, Incremental PageRank Computation can help in getting the popular web pages to be ranked first. Using the results of this computation, an incremental approach yields faster convergence. Secondly, Seeded PageRank Computation concentrates on the topic domain. For instance, if all **Health** and **Medicine** related domain needs to be considered, one can choose a finite set of seeds for Health and another set for Drugs or Medicine. These two sets (as a graph) may not necessarily be connected. More real world examples can be seen at *www.google.com/intl/en/options/*. Using these ideas, we propose a distributed algorithm in the following section.

3.2 Empirical Analysis of Model

We have performed a set of experiments on some simulated graphs and real web pages with the goal to find the best scalable method to compute PageRank. In order to achieve this goal, the strategy we use is a common sense approach. If we were to distribute the workload into N different processes or machine, we need to identify a common group where there is a sense of strong connectivity.

Table 4: Top 10 Ranks: MergedRank

MergedRank	Error(Approx)
0.9924762290916748	2.22 x 10^{-16}
0.9924952557627331	0.0
0.9925162958461424	0.0
0.992567702701218	0.0
0.9925805241411206	0.0
0.9926211479053078	1.32 x 10^{-16}
0.9926356168545798	1.94 x 10^{-16}
0.9926586140180446	2.22 x 10^{-16}
0.9926689754168181	0.0
0.9927664432290569	0.0
0.9927877720089899	0.0
0.9928270460995271	0.0
0.9928484397442923	0.15 x 10^{-16}
0.992903741535735	0.0
0.9930098466583472	0.0
0.9930472623710699	0.0
0.9931250687148628	2.22 x 10^{-16}
0.9931317812331083	0.0
0.993224889176337	0.36 x 10^{-16}

We use randomly simulated graphs for a good reason. We had to conduct some of our experiments many times on different set of data, and mostly such analysis helps in using a combination of real world data and random data. In the past Boldi et al. [19] used simulation on subsets of the Web of 40 million pages from the .it domain and 100 million pages from the WebBase crawl, testing breadth-first against random ordering and came with good results.

The concept of *ServerRank* proposed earlier [23] seems interesting, but an example such as **wikipedia** will probably produce information in so many different context that it could produce a largely aggregated graph. On the contrary, we are seeking a divide and conquer approach and by separating the crawled spaces intelligently into several dense graphs, and merging ranks with a combination of iterative and extrapolation methods, we achieve the required goal. We are therefore interested in a distributed algorithm that works well with the assumption that we have a finite set of subgraphs that are strongly connected by themselves and can be aggregated into one large graph.

Storage and computational speed is mandatory in computations such as PageRank, and this will remain areas of future work for some time. As the web continues its growth, the need for smarter storage schemes and faster numerical methods will become more evident. Various approaches in representing web graphs has been already studied [20]. In this analysis, we experimented with compressed storage of adjacency list and if we could represent a compressed bitset then finding similar adjacency list is easier by using efficient bit twiddling mechanism, but this approach is still under progress.

We analyze two models here, and in both these models we assume that we have an efficient mechanism to retrieve and store backlinks and outdegree corresponding to each node. In the first model, for simplicity we consider a distributed

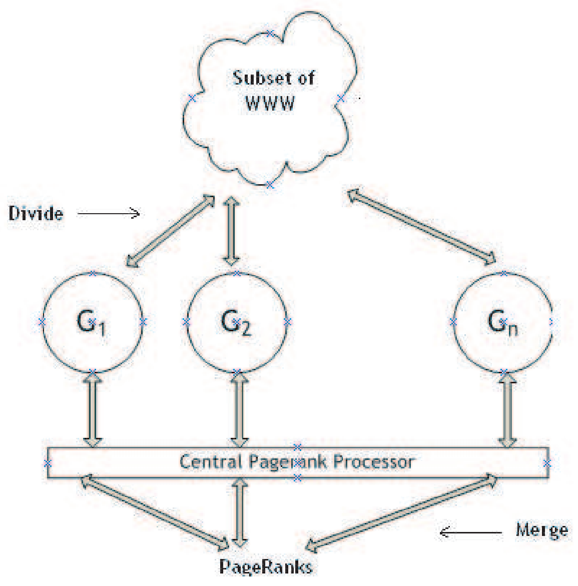

Figure 2: Distributed PageRank using MergeRank

model over 3 systems that can be extended to a higher scale. Each processor using their own localranks work on producing PageRank on their local graph. The only time they communicate with any other processor is when there is a merging of ranks required. There is certain amount of communication and merging overhead. The advantage here is that localranks are much faster for their corresponding local computation.

In the second model, we assume a central pagerank processor that handles all pagerank updates and synchronization. Each processor can have a cached version of localranks and at frequent time (not necessarily for every rank updates, but after every iteration), should communicate with the central pagerank processor. In this model, the processors dont need to keep track of where all the nonlocal ranks are stored. These processors only need to communicate with the central pagerank processor. This central pagerank processor should be able to communicate concurrently in an efficient manner.

The proposed algorithm here in this model is that each of the processor communicate only for fewer iterations, depending on the graph size. Let us assume that we decide to have 30 iterations for a million nodes. At the end of 30 iterations, the central rank processor should not only have pageranks, but also pageranks from previous two iterations. Using Extrapolation methods, the central rank processor iterates through until the stopping criteria is achieved. The savings on iterations were recorded as shown in **Figure 3**. In addition to these savings, the advantage here is that pagerank computation over a larger collection is divided into a smaller such computation. One overhead is that the central pagerank processor needs to keep track of pagerank values of 3 most recent iterations. However, this can be accomplished by implementing an efficient I/O based mechanism to keep track of the iterative values.

The reason for savings on iterations is partly because we use Extrapolation methods and the fact that we use the initial values of pagerank differently.

Table 5: Basic statistics for the data sets used in the analysis

Name	Nodes	Links	Storage Size
edu	$0.25M$	$1.7M$	$25MB$
med	$0.45M$	$2.8M$	$62MB$
random1	$0.2M$	$3M$	$25MB$
random2	$0.6M$	$8M$	$85MB$
random3	$0.8M$	$10M$	$125MB$

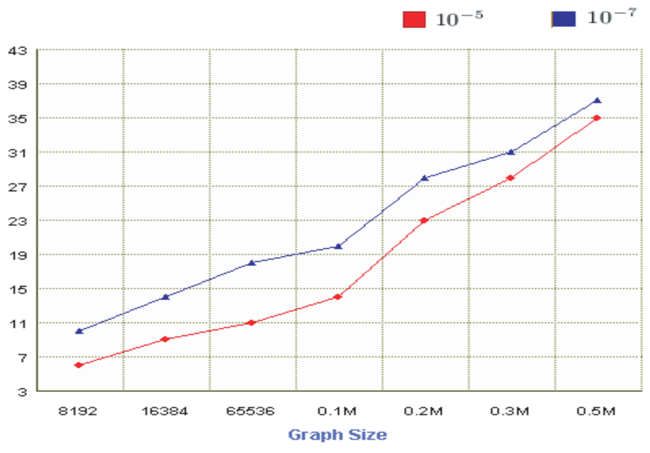

Figure 3: Savings on iterations

3.2.1 *Experimental Setup*

The entire analysis were conducted on Sun Fire 280R with 2GB of memory and 72GB disk space, a Sun Ultra 10 with 512 MB with 200GB IDE. Due to this limited resources, we use more I/O based implementation. For future study and analysis we have interest in attempting various compression methods and do a comparison study.

3.2.2 *Experimental Data*

In out experiments we use two set of Web related directed graphs. We call the first as "edu" graph visiting some chosen set of educational web pages from google */options/universities.html*. The second set we call "med" graph not to be mixed up with *medline*. The "med" graphs were generated from a chosen set of seeds that consisted of *www.nlm.nih.gov*, *www.webmd.com* and *www.hhmi.org* so that we stay focused in that domain. Basic statistics for these set of graphs is shown in the **Table 5**.

4. CONCLUSIONS

We presented an efficient method to perform the PageRank calculation in parallel over arbitrary large web graphs. We accomplished this by introducing two novel approach using incremental and divide and conquer methods as well as by adapting the extrapolation methods for solving linear systems to be used with PageRank.

Our next goal is to experiment with several other PageRank specific enhancements, such as better compression methods for storing adjacency lists and more efficient mechanism for merging ranks. We use Aitken's Extrapolation as it is simpler compared to Quadratic Extrapolation. We shall be testing Quadratic Extrapolation and compare the results with our current analysis.

5. ACKNOWLEDGMENTS

We would like thank Vincent Chan for suggestions to optimize storage, Roske-Shelton Renate for her valuable suggestions and several anonymous reviewers who were very encouraging.

6. REFERENCES

[1] A. Benczur, K. Csalogany, T. Sarlos, and M. Uher. Spamrank - fully automatic link spam detection. *First International Workshop on Adversarial Information Retrieval on the Web*, pages 1–14, 2005.

[2] P. Berkhin. A survey on pagerank computing. *Internet Mathematics*, 2(1):73–120, 2005.

[3] W. Boswell. 100 search engines in 100 days. *http://websearch.about.com/*, 2005.

[4] M. Bouklit and F. Mathieu. Backrank: an alternative for pagerank. *In the Proceedings of the 14th international World Wide Web conference on Alternate track papers and posters*, pages 1122–1123, 2005.

[5] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, 1998.

[6] E. F. V. de Velde. *The OpenURL Framework for Context-Sensitive Services*. http://www.niso.org/, National Information Standards Organization, 2004.

[7] C. Dwork, S. R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. *In Proceedings of the 10 th International World Wide Web Conference, Hong Kong, May*, pages 613–622, 2001.

[8] D. Fogaras and B. Racz. Scaling link-based similarity search. *Technical report, MTA SZTAKI*, 2004.

[9] A. Gulli and A. Signorini. The indexable web is more than 11.5 billion pages. *ACM*, pages 902–903, May 2005.

[10] Z. Gyongyi and H. Garcia-Molina. Web spam taxonomy. *First International Workshop on Adversarial Information Retrieval on the Web*, pages 1–9, 2005.

[11] Z. Gyongyi, H. Garcia-Molina, and J. Pedersen. Combating web spam with trustrank. *Proceeedings of the 30th VLDB Conference*, pages 576–587, 2004.

[12] T. Haveliwala, A. Gionis, D. Klein, and P. Indyk. Evaluating strategies for similarity search on the web. *Proc. of the Eleventh International World Wide Web Conference*, 2002.

[13] T. H. Haveliwala. *Topic-sensitive PageRank*. Proceedings of the Eleventh International World Wide Web Conference, Honolulu, Hawaii, 2002.

[14] H. L. E. M. J. Janssen. Probabilistic models for focused web crawling. *ACM*, pages 16–22, November 2004.

[15] S. Kamvar, T. Haveliwala, C. Manning, and G. Golubr. Extrapolation methods for accelerating pagerank computations. *In Proceedings of the Twelfth International World Wide Web Conference*, 2003.

[16] J. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604–632, 1999.

[17] R. Lempel and S. Moran. The stochastic approach for link-structure analysis (SALSA) and the TKC effect. *Computer Networks (Amsterdam, Netherlands: 1999)*, 33:387–401, 2000.

[18] C. Moler. The worlds largest matrix computation. *Matlab News and Notes*, pages 12–13, October 2002.

[19] P.Boldi, B. Codenotti, M.Santini, and S. Vigna. Ubicrawler: a scalable fully distributed web crawler. *Software - Practice and Experience*, 34(6):711–725, 2004.

[20] S. Raghavan and H. Garcia-Molina. Representing Web graphs. *IEEE Intl. Conference on Data Engineering*, 2003.

[21] M. Sydow. Random surfer with back step. *In the Proceedings of the 13th international World Wide Web conference on Alternate track papers and posters*, pages 352–353, 2004.

[22] S. C. M. van den Berg Byron Dom. Focused crawling: a new approach to topic-specific web resource directory. *Proc. 8th Int. World Wide Web Conf*, pages 545–562, 1999.

[23] Y. Wang and D. DeWitt. Computing pagerank in a distributed internet search system. *In Proceedings of the International Conference on Very Large Databases,(VLDB)*, pages 420–431, 2004.

[24] Y. Zhu, S. Ye, and X. Li. Distributed pagerank computation: Based on iterative aggregation-disaggregation methods. *CIKM'05 October 31-November 5*, 2005.