Parallel Adaptive Technique for Computing PageRank

Arnon Rungsawang and Bundit Manaskasemsak
Massive Information & Knowledge Engineering
Department of Computer Engineering, Faculty of Engineering
Kasetsart University, Bangkok 10900, Thailand.
{arnon, un}@mikelab.net

Abstract

Re-ranking the search results using PageRank is a well-known technique used in modern search engines. Running an iterative algorithm like PageRank on a large web graph consumes both much computing resource and time. This paper therefore proposes a parallel adaptive technique for computing PageRank using the PC cluster. Following the study of the Stanford WebBase group on convergence patterns of PageRank scores of pages using the conventional PageRank algorithm, PageRank scores of most pages converge more quickly than the remainder, we then devise our parallel adaptive algorithm to reiterate the computation for pages whose PageRank scores are still not converged. From experiments using a synthesized web graph of 28 million pages and around 227 million hyperlinks, we obtain the acceleration rate up to 6-8 times using 32 PC processors.

1. Introduction

The vital goal of any web search engine is to serve the most satisfying search answers to end-users. Some researchers interest in retrieving results with the fastest response time; some try to find an efficient technique to present the most possible relevant pages. In addition, much of work has been focused on exploiting the relationship between web pages via hyperlinks. The web link analysis may lead researchers to examine how a web page links to others with some topics or concealed authoritativeness. There are two famous techniques such as HITS [10] and PageRank [12] that have been proposed since 1998. However, we will only concentrate ourselves on PageRank here.

PageRank, approached to re-rank the search results, has been one of the best-known web search algorithms since the advent of the Google^{TM} [12]. PageRank is both a compute-intensive algorithm and a computing resource hunger. With the sheer volume of increasing web pages today, despite computing the subset of their PageRank scores would still take days and may consume unlimited computing resources. In addition, the continual unceasing update, addition or removal of web pages, directly effect the frequent re-computation of the PageRank to freshen the relevance of the search results. Also, in the context of topic-sensitive or personalized web search [5, 6], the large number of PageRank scores needs to be recomputed to reflect the user preferences. Inevitably, faster PageRank computational technique is still needed to explore.

Many researchers have performed many extensive studies in accelerating the PageRank computation. To mention for a few examples, Arasu et al. [1] speed up the convergence rate of the computation by both using the Gauss-Seidal method and exploiting structural properties of the web graphs. Haveliwala [4] and Chen et al. [2] explore efficient memory and disk-based computation. Jeh and Widom [6] exploit dynamic programming technique to compute a large number of personalized PageRank vectors simultaneously. Kamvar et al. [8, 9] exploit the implicit block structures of web graphs, as well as accelerate the computation by periodical subtraction of estimates of the non-principal eigenvectors from the current iteration. Sankaralingam et al. [15] make use of the distributed P2P architecture to speed up the computation, while Rungsawang and Manaskasemsak [14] also suggest a fast parallel computing solution on the PC cluster.

In this paper, we propose another fast parallel PageRank implementation, as well as preliminary computing experiments, using the PC cluster. A large synthesized input web graph is first divided into several partitions so that PageRank scores of web pages in every partition can be computed in parallel using several processors. Following the findings of Kamvar et al. [7], the PageRank scores of the majority of the web pages converge more quickly, while those of the rest take much longer time to converge. We then devise our algorithm, named "parallel adaptive PageRank", to accelerate the overall computation by reiterating the computation for only those pages whose PageRank scores are still not converged. From experiments using a web graph of around 28 million pages and 227 million hyperlinks synthesized from the Stanford WebBase project [17], we found that the proposed approach could accelerate the computation up to 6-8 times using 32 processors.

We organize this paper in the following ways. Section 2 briefly introduces the basic PageRank concept, as well as the idea of the adaptive PageRank computation. Section 3 provides more detail about the proposed parallel adaptive PageRank algorithm. Section 4 describes how we perform the experiments, and discuss the preliminary results. Finally, section 5 concludes the paper.

2. Basic concept

In this section, we introduce the basic PageRank concept, and continue to review the adaptive PageRank idea reported in [7].

2.1. PageRank computing concept

The concept of PageRank algorithm inspires from human behavior of voting. Large numbers of web pages, recursively referenced by others via hyperlinks, constitute a gigantic consensus web graph of the Internet. Pages mostly voted by many authors (i.e., referenced by many other pages) will thus have high voting scores. In other words, those pages have been considered to be the most interesting pages. If one is relevant with user's query, it consequently should be ranked at the first part of the search results.

Theoretically, PageRank can be computed as the principal eigenvector of a Markov chain described by transition probability matrix using a simple power method. The PageRank score of a web page is the summation of transitional probability from others to this page. Consider a transition probability matrix P of size n by n describing any element with

$$P_{ij} = \begin{cases} 1/N_i & \text{if there is a transition from page } i \text{ to page } j \\ 0 & \text{otherwise} \end{cases}$$

where N_i is the number of pages which page i points to (called later "out-degree"). P is said to be valid if it is a row-stochastic matrix or has no rows consisting of all zeros. This involves the web graph to be pruned any pages with out-degree 0. An approach dealing with dangling pages (i.e., pages with zero out-degree) is to change the transition probability matrix P to the new matrix P'. Let n be the total number of pages in the web graph, $\vec{v}$ be a uniform teleportation vector:

$$\vec{v} = \left[\tfrac{1}{n}\right]_{n\times 1}$$

The row-stochastic matrix P' can be defined as:

$$P' = P + \vec{d} \cdot \vec{v}^T$$

where $\vec{d}$ is an n-dimensional column vector indicating those dangling pages:

$$d_i = \begin{cases} 1 & \text{if } N_i = 0 \\ 0 & \text{otherwise} \end{cases}$$

In term of $\vec{d} \cdot \vec{v}^T$, used to modify the transition probability matrix, means that if the transition visits a dangling page, it will randomly jump to another page with the probability $1/n$.

The power method can be guaranteed to convert to a principal eigenvector of the transition probability matrix if the matrix follows the Markov's properties (i.e., Ergodic theorem) [13]. The two conditions of Markov chain have been defined that the matrix P' is *aperiodic* and *irreducible*. The former leads the transition to visit any page and then may be possible for randomly jumping back to that page with period 1. The latter involves the web graph that must be strongly connected. Let $\vec{e}$ be an n-dimensional column vector containing all elements of 1:

$$\vec{e} = [1]_{n\times 1}$$

The general way to produce a strong connectivity is to add the transition probability matrix with a small teleportation distribution given by $\vec{v}$ to all pages.

$$P'' = \alpha P' + (1-\alpha)\vec{e} \cdot \vec{v}^T$$

The surfer affects to visit any page and then will randomly jump to a random page with the probability $(1-\alpha)$, where α is a teleportation coefficient that has been set to 0.85 by many studies.

Consequently, the modified transition probability matrix P'' both is a row-stochastic matrix and satisfies above Markov's properties. Let $\vec{x}^{(k)}$ be n-dimensional column vector of probability distribution over all pages at time k. Then the simple power iterative computation may be formulated as:

$$\vec{x}^{(k+1)} = P''^T \vec{x}^{(k)}$$

The probability distribution at time 0 given by $\vec{x}^{(0)}$ is an independently initial distribution. In general, the process should be started with the uniform distribution (i.e., $\vec{x}^{(0)} = \vec{v}$), and then reiteratively computed until convergence. The power iteration system ($\vec{x} = P''^T \vec{x}$) will finally convert to the principal eigenvector with eigenvalue equal to 1.

In practice, the iterative computation for a unique stationary probability distribution of a directed graph of web pages (or vertices) related with their hyperlinks (or edges) can be considered as a matrix-vector multiplication form. The probability distribution at time k+1 is calculated by using the transition probability matrix (P) multiplies by the previous one at time k and then pluses the teleportation vector. Therefore, to compute a rank score of a web page, we simply re-formulate the PageRank computation in the following way. Let $R(u)$ represent the rank score of a page u, n be the total number of web pages, N_u be the out-degree of page u, and S_v represent the set of pages pointing to page v. If a page v has many other pages u pointing to, then the new rank score of v can iteratively be computed by:

$$\forall_v R'(v) = \alpha \sum_{u \in S_v} \frac{R(u)}{N_u} + \frac{(1-\alpha)}{n} \tag{1}$$

Here α is the teleportation coefficient of the random surfer model as mentioned. The computation reiterates until all rank scores converge. We say that the PageRank of any page v converges when the comparison of its new computed rank score and the old one satisfies the following condition:

$$\frac{|R'(v) - R(v)|}{|R(v)|} \leq \delta \tag{2}$$

The δ, called the "relative tolerance", is the predetermined value used as condition to stop the iteration. Practically, this relative tolerance is set to 0.001 or 0.0001.

2.2. Adaptive PageRank idea

The need to accelerate the PageRank computation for a large web graph challenges many researchers in web link analysis and search engine development. From extensive observations on a web dataset extracted from the Stanford WebBase project (i.e., around 80 million pages and closed to a million links), Kanvar et al. [7] found that PageRank scores have converged on a non-uniform distribution. Many pages with their true small scores converge very fast while few higher score ones take a longer time to converge. Moreover, they found that the PageRank scores for over two-thirds of pages converge after 20 iterations when setting the relative tolerance to 0.001.

Accordingly, to eliminate the redundancy of computational need, Kamvar et al. [7] proposed to omit the re-computation of the PageRank scores of pages that have already converged, and then immediately employ those PageRank scores in the next iterations. This can thus reduce the total running time of the original PageRank algorithm. From this finding, the previous Equation (1) can also be adapted to:

$$\forall_v R'(v) = \begin{cases} R(v) & \text{if } R(v) \text{ converges} \\ \alpha \sum_{u \in S_v} \frac{R(u)}{N_u} + \frac{(1-\alpha)}{n} & \text{otherwise} \end{cases} \tag{3}$$

In our parallel implementation proposed in this study, we will refer to Equation (3) as parallel adaptive PageRank computational method, while we will refer to Equation (1) as the original one.

3. Parallel adaptive PageRank algorithm

The most intuitive goal is how PageRank scores of a large web graph can compute as fast as possible because of frequent computing need in many reasons: the up to date collection of web search engine, or the need of several number of stationary probability distribution using in Topic-sensitive search [5], etc. In this section we persuade into computing PageRank scores in a parallel way. There are two approaches in numerical solution. The former tries to solve the probability distribution by using power iteration that is the standard way, while the latter focuses on the PageRank linear system.

In this study, we propose a parallel computing on power law solution and investigate the advantage of adaptive PageRank idea. Therefore, we will first mention our data structure, and continue to the proposed parallel adaptive PageRank implementation. However, there is an alternative way explored in [3] which concentrates on linear system solution by using Jacobi iterations and several Krylov subspace methods, but we do not go into that detail here.

3.1. The binary link structure file

For the crawled web graph collection used in our proposed algorithm, we first sorted URLs into alphabetical order and assigned their corresponding increasing numbers for page mapping since we will refer to this number in computing process. Secondly, the relationship among each web page (i.e., via its hyperlinks) in this crawled collection has to be transformed into a binary link structure file *L*, out-degree file *O* and in-degree file *I*, as illustrated textually in Figure 1. Each number (storing in 4-byte integer) here represents a web page URL. For example of meaning, the page number 3 has been pointed by page number 311 and 312, and it has 3 links pointing out to other pages and 2 links from other pages (here, they are page number 311 and 312) pointing to it.

dest_id (4 bytes)	**src_id (4 bytes each)**	**out_deg (4 bytes)**	**in_deg (4 bytes)**
1	1028	1	1
2	106	5	1
3	311 312	3	2
4	35 96 487 5052	1	4
file *L*		file *O*	file *I*

Figure 1: The binary link structure file L, out-degree file O, and in-degree file I, respectively.

3.2. Parallel implementation

To accelerate the PageRank computation for a large web graph using β processors, we first equally divide the corresponding binary link structure file *L*, as well as the corresponding in-degree file *I*, by destination URL numbers (i.e., *dest_id*) into β partitions, named later by L_i and I_i; $0 \le i \le \beta$, and distribute those partitions, including with the entire out-degree file *O*, to those β processors in the PC cluster. Before computing PageRank scores, any processor *i* should allocate three arrays of floating point in main memory (if it is possible). The first one is the R_i storing source rank vector (i.e., the old rank scores) of *n* entries, where *n* is the total number of unique pages existing in the input web graph collection. The second one is the O_i storing the out-degree information that correspond to the whole *n* web pages, and the last one is the R'_i storing the destination rank vector (i.e., the new computed rank scores) of n/β entries. The parallel adaptive PageRank algorithm running at a processor *i* can be written in Algorithm 1 below.

Algorithm 1. The parallel adaptive PageRank algorithm.

```
∀_u R_i[u] = 1/n // initialize all ranks with 1/n
initialize( S_{R'C} , S_{R'N} )
while ( S_{R'N} ≠ φ ) {  // check empty set of non-converged ranks
    while ( L_i is not end of file) { // read link structure file
        scanIndegree ( I_i.in_deg ) // scan corresponding in-degree
        scanLink ( L_i.dest_id , L_i.src_id_1 , L_i.src_id_2 , …, L_i.src_id_{I_i.in_deg} )
        if ( L_i.dest_id ∈ S_{R'C} ) // whether page is in the set of converged ranks
            R'_i[L_i.dest_id] = R_i[L_i.dest_id]
        else { // in case that page is in the set of non-converged ranks
            score = 0
            for ( j = 1… I_i.in_deg )
                $score = score + \frac{R_i[L_i.src\_id_j]}{O_i[L_i.src\_id_j]}$
            $R'_i[L_i.dest\_id] = (\alpha \times score) + \frac{(1-\alpha)}{n}$
        }
    }
    [ S_{R'C} , S_{R'N} ] = detectConverge ( R'_i , R_i , δ )
    synScores ( ∀_{v∈S_{R'N}} R'_i[v] ) // update non-converged ranks at other processors
    R_i ← R'_i
}
```

The Algorithm 1 proceeds as follows. We first initial the rank scores of all web pages to $1/n$, then empty the set of converged rank $S_{R'C}$ and register all pages to the set of non-converged ranks $S_{R'N}$. The block of while loop between lines 3-19 reiterates until there is nothing left in the set of non-converged ranks. The inner block of while loop between lines 4-15 reads the binary link structure file L_i and computes the new rank score R' following the Equation (3).

The detectConverge( ) function at line 16 identifies the pages whose ranks have already converged under the relative tolerant condition written in the Equation (2), and then updates both converged set $S_{R'C}$ and non-converged sets $S_{R'N}$.

The synScores(…) function at line 17 performs the duty called "pairwise rank synchronization" that updates new computed and non-converged rank scores locating at other processors using a simple message passing approach. During each communication step, any processor may send or receive a package to/from its assigned pair, and then the next step it will send or receive to/from another pair, and so on till the synchronization process is complete (see more detail in [11]). This synchronization algorithm guarantees to finish (i.e., all processors could receive non-converged rank scores of each other) in the optimal $2\log_2 \beta$ communication steps.

Finally, the code at line 18 prepares the new computed rank scores to be ready for the next iteration. The Algorithm 1 can be easily converted to a parallel non-adaptive PageRank one by simply removing line 7-9 and 14, respectively.

4. Experiments and preliminary results

We implemented both Algorithm 1 (noted later by "APR"), as well as another parallel non-adaptive one for comparison (noted later by "PR"), in C language. We also used the standard MPICH message passing library version 1.2.6 in synchronization process. The preliminary test collection was synthesized from the subset of the web graph obtained from the Stanford WebBase project [17]. This collection consists of around 28 million pages, and 227 million hyperlinks. All experiments had been performed on the F32 cluster at the AIST (Japan) under the ApGrid contract [16]. Each machine in the cluster is equipped with double 3.0 GHz Intel Xeon CPUs, 4GB of main memory, and a SCSI hard disk, connected to each other via a Gigabit Ethernet and runs the Linux RedHat 8.0.

Both the APR and PR algorithms had been tested with the synthesized web data using 1, 2, 4, 8, 16, and 32 machines. During the experiments, we only used one processor in each machine for running an assigned task, and investigated the wall-clock time needed to converge all rank scores with the pre-defined relative tolerant values of 0.001 and 0.0001. We also repeated the experiments several times and then averaged the obtaining results. Figure 2 concludes the results.

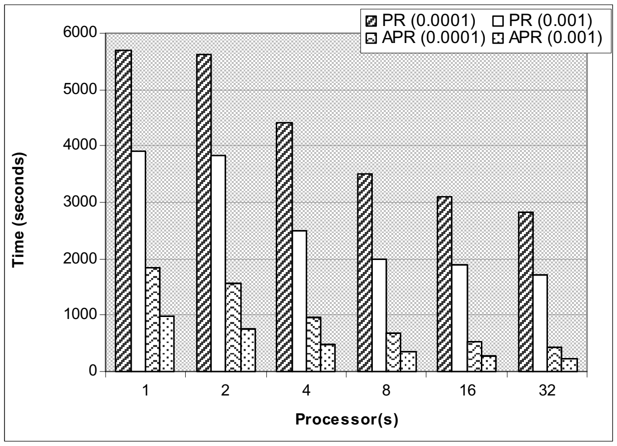

Figure 2: Wall-clock time need to converge the rank scores to relative tolerance values of 0.001 and 0.0001.

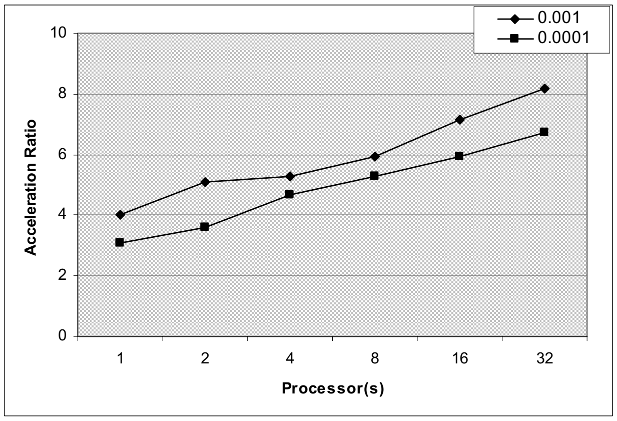

Figure 3: Acceleration ratio graphs concluded from the experiments.

From the results graphically shown in the Figure 2, we can see that the running time of the parallel adaptive PageRank algorithm (APR) can be significantly reduced by eliminating the redundant computation needed for the already converged pages. The running time gradually decreases when we employ more processors in the PC cluster. To clearly see how much acceleration we can obtain, we additionally define the acceleration ratio as the ratio between the average wall-clock time needed for PR and APR algorithms using *n* processor(s), and re-plot the graphs in Figure 3.

The two acceleration ratio graphs in Figure 3 clearly show the nearly linear speed up rates. The slopes of the curves tend to increase as the excessive computation that is needed to converge the PageRank scores to lesser relative tolerance value decreases.

5. Conclusion

This paper proposes an implementation of a parallel adaptive PageRank algorithm on the PC cluster. The main idea of the algorithm is based on the study of Kamvar et al. [7] on their findings that the majority of pages in the test dataset have their PageRank scores converged more quickly than the remaining. Following this idea, we then devise the parallel algorithm to reiterate the PageRank computation only on those pages whose rank scores do still not converge.

We test the algorithm on the F32 cluster, a powerful set of PC machines on the ApGrid network [16]. From the experiments performed on a web graph synthesized from the Stanford WebBase project [17], we found that the average wall-clock time required to converge the PageRank scores to relative tolerance value of 0.001 and 0.0001 drastically reduces. The algorithm can provide nearly linear acceleration rate when the number of computing processors has been increased. We can obtain up to 6-8 times acceleration rates when using up to 32 processors.

Since the results in this paper are only concluded from the preliminary experiments, more extensive study on a set of larger and realistic web graphs is expected. Moreover, we believe that there are still more rooms to better optimize the algorithm to utilize the computing power of the both processors on each F32 machine, and employ the maximum performance of the Gigabit Ethernet network that connects all those machines.

6. References

[1] A. Arasu, J. Novak, A. Tomkins, and J. Tomlin. PageRank Computation and the structure of the web: Experiments and algorithms. In *Proceedings of the 11th International World Wide Web Conference, Poster Track*, 2002.

[2] Y. Chen, Q. Gan, and T. Suel. I/O efficient techniques for computing PageRank. In *Proceedings of the 11th International Conference on Information and Knowledge Management*, 2002.

[3] D. Gleich and L. Zhukov. Scalable Computing for Power Law Graphs: Experience with Parallel PageRank. *Yahoo! Technical Report*, 2005.

[4] T.H. Haveliwala. Efficient computation of PageRank. *Computer Science Technical Report, Stanford University*, 1999.

[5] T.H. Haveliwala. Topic-Sensitive PageRank. In *Proceedings of the 11th International World Wide Web Conference*, 2002.

[6] G. Jeh and J. Widom. Scaling Personalized Web Search. In *Proceeding of the 12th International World Wide Web Conference*, 2003.

[7] S.D. Kamvar, T.H. Haveliwala, and G.H. Golub. Adaptive Methods for the Computation of PageRank. *Computer Science Technical Report, Stanford University*, 2003.

[8] S.D. Kamvar, T.H. Haveliwala, C.D. Manning, and G.H. Golub. Exploiting the block structure of the web for computing PageRank. *Computer Science Technical Report, Stanford University*, 2003.

[9] S.D. Kamvar, T.H. Haveliwala, C.D. Manning, and G.H. Golub. Extrapolating methods for accelerating PageRank computations. In *Proceeding of the 12th International World Wide Web Conference*, 2003.

[10] J. Kleinberg. Authoritative sources in a hyperlinked environment. In *Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms*, 1998.

[11] B. Manaskasemsak and A. Rungsawang. Parallel PageRank Computation on a Gigabit PC cluster. In *Proceedings of the International Conference on Advanced Information Networking and Applications*, 2004.

[12] L. Page, S. Bill., R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. *Stanford Digital Libraries Working Paper*, 1998.

[13] S.M. Ross. Introduction to Probability Models, 8th Edition, Academic Press, 2003.

[14] A. Rungsawang and B. Manaskasemsak. PageRank Computation using PC Cluster. In *Proceedings of the 10th European PVM/MPI Users' group Meeting*, 2003.

[15] K. Sankaralingam, S. Sethummadhavan, and J.C. Browne. Distributed PageRank for P2P Systems. In *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, 2003.

[16] The ApGrid, http://www.apgrid.org/, 2004.

[17] The Stanford WebBase Project, http://www-diglib.stanford.edu/~testbed/doc2/WebBase/, 2004.