A Mixed MPI-Thread Approach for Parallel Page Ranking Computation

Bundit Manaskasemsak[1], Putchong Uthayopas[2], and Arnon Rungsawang[1]

[1] Massive Information & Knowledge Engineering
Department of Computer Engineering, Faculty of Engineering
Kasetsart University, Bangkok 10900, Thailand
{un, arnon}@mikelab.net
[2] Thai National Grid Center, Software Industry Promotion Agency
Ministry of Information and Communication Technology, Thailand
putchong_ut@thaigrid.or.th

Abstract. The continuing growth of the Internet challenges search engine providers to deliver up-to-date and relevant search results. A critical component is the availability of a rapid, scalable technique for PageRank computation of a large web graph. In this paper, we propose an efficient parallelized version of the PageRank algorithm based on a mixed MPI and multi-threading model. The parallel adaptive PageRank algorithm is implemented and tested on two clusters of SMP hosts. In the algorithm, communications between processes on different hosts are managed by a message passing (MPI) model, while those between threads are handled via a inter-thread mechanism. We construct a synthesized web graph of approximately 62.6 million nodes and 1.37 billion hyperlinks to test the algorithm on two SMP clusters. Preliminary results show that significant speedups are possible; however, large inter-node synchronization operations and issues of shared memory access inhibit efficient CPU utilization. We believe that the proposed approach shows promise for large-scale PageRank applications and improvements in the algorithm can achieve more efficient CPU utilization.

Keywords: PageRank computation, parallel computing, link analysis.

1 Introduction

The information and knowledge resources provided via the Internet continue to grow at a rapid rate, making effective search technology an essential tool for users of this information. However, the continual growth in volume of web pages presents a great challenge to search engines that must classify the relevance of web resources to user searches. Much current research is directed toward finding more efficient methods to obtain effective search results. One important research area is the development of algorithms to estimate the authoritative score of each web page by analyzing the web's hyperlinked structure. Leading algorithms are HITS [6] and PageRank [10], first proposed in 1998 and subsequently enhanced. The scores computed by these algorithms are utilized by the search engine's ranking algorithm: pages with the more significant or higher scores are more highly ranked in search results ordering. Currently, web link analysis remains one of the most important components of the search engine.

The PageRank algorithm is less complex than HITS, making it more practical for large-scale application; PageRank is utilized by several well-known search engines such as Google. PageRank requires analyzing the entire hyperlinked structure of the web graph once and then iteratively calculates the page scores. Unfortunately, the vast number of pages that must be ranked increasingly make PageRank very computationally expensive. For large-scale computation, most researchers usually propose to first partition a huge web graph into several parts and then to compute them separately. Some studies, such as [3, 1, 5], propose to sequentially compute each partition and then combine the sub-results into the global PageRank scores. Other studies utilize parallel computing on a PC cluster [8, 2] or even distributed P2P architecture [14, 15] to improve performance.

In this paper, we investigate the use of cluster technology together with an efficiently parallelized version of the PageRank computation to improve performance. The main idea of our algorithm is to efficiently employ the computing power of the cluster to compute subsets of PageRank scores in parallel, and then combine them to obtain the total scores of web pages. We also investigate the use of lightweight processes, or threads [11], in a Symmetric Multiprocessing (SMP) environment to reduce communication overhead and take advantage of shared memory. Cluster communication is implemented using the standard Message Passing Interface (MPI) protocol [9]. Since our implementation employs both the multi-threading and cluster computing models, we call this approach to parallelization a “mixed model”.

The rest of this paper is organized as follows. Section 2 briefly reviews the PageRank algorithm and introduces acceleration techniques for PageRank computation. Section 3 discusses the need of parallelization and gives the detail of such algorithms as well as system design. Section 4 describes our experiments and discusses the results. Finally, section 5 presents conclusions and planned future work.

2 Basic PageRank Concept

2.1 The Intuition

The concept behind PageRank [10] is to estimate the importance of web pages by hyperlinked structure analysis. A link from page u to page v indicates that the author of u recommends and thus confers some importance on page v. Furthermore, a page mostly referred by other “important” pages is also important.

To mathematically formulate this intuitive concept, let n be the total number of pages in the web graph and let R be the PageRank vector: $R(u)$ is the PageRank score of page u. Also let $O(u)$ be the number of pages that u points out, called “out-degree”. All pages u linked to v are grouped into a set of backward pages of v, denoted B_v. The rank score of all pages v can be computed using the iterative formula:

$$\forall v : R^{(k+1)}(v) = \alpha \sum_{u \in B_v} \frac{R^{(k)}(u)}{O(u)} + (1-\alpha)E(v) \tag{1}$$

The two terms in this equation represent two factors contributing to a page's rank. The first is the traditional rank propagation calculated from the hyperlinked structure, weighted by α (usually set to 0.85). The second term represents a random surfer process over a uniform distribution (i.e., $\forall v : E(v) = 1/n$). When the surfer visits any page he can subsequently jump to a randomly selected page, with probability $1/n$. This term also guarantees convergence of $R^{(k)}$ and avoids the "rank sink" problem [10]. The convergence of $R^{(k)}$ can be proved by application of Markov's Theorem [12].

$R^{(0)}$ is the initial distribution; in general, the process is started with a uniform distribution, $\forall u : R^{(0)}(u) = 1/n$. Iterative computation of $R^{(k)}$ is performed using Equation (1) until the rank scores converge. The usual convergence criterion is that the relative change in scores of all pages between iterations k and k+1 be below a prescribed tolerance. That is, $\forall v$

$$\frac{\left|R^{(k+1)}(v) - R^{(k)}(v)\right|}{\left|R^{(k)}(v)\right|} \le \delta \tag{2}$$

The relative tolerance for convergence, δ, is a pre-assigned value; in our experiments, δ is set to 0.0001.

2.2 Adaptive PageRank Technique

Application of the PageRank computation in Equation (1) reveals that the convergence rate of elements of $R(v)$ is highly non-uniform. Kamvar et al. [4] have found that the rank computation for many pages with small rank scores converge quickly to their final values, while the ranks of pages with high scores take a much longer time for the values to converge.

To eliminate the redundant computation of converged page ranks, Kamvar et al. proposed an "adaptive PageRank algorithm" [4] that omits recomputation of PageRank scores that have already satisfied the convergence criterion. For these pages, they simply use the previous score in successive iterations. This reduces the running time of the original algorithm. Using their modification to Equation (1) yields:

$$\forall v : R^{(k+1)}(v) = \begin{cases} R^{(k)}(v) & \text{if } R(v) \text{ converged} \\ \alpha \sum_{u \in B_v} \frac{R^{(k)}(u)}{O(u)} + (1-\alpha)E(v) & \text{otherwise} \end{cases} \tag{3}$$

A PageRank score $R^{(k)}(v)$ is marked as converged when it satisfies Equation (2). The algorithm terminates when all PageRank scores have been marked as converged. We utilize this adaptive technique in our parallel implementation, described next.

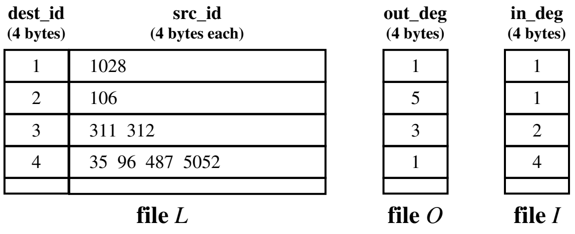
3 Parallel PageRank Computation

To compute the PageRank scores of the totality of pages comprising the hyperlinked structure of the Internet by application of Equations (1) or (2) would require massive computing power as well as enormous amounts of memory. In practice, this computation is nearly infeasible. To obtain a more tractable algorithm, we exploit parallelism and partitioning of the problem, and utilize shared resources of a computational cluster.

In this section, we introduce a parallelized version of the PageRank algorithm. First we develop the web graph representation. Then, we provide detail of the parallel algorithm.

3.1 Web Graph Representation

From the crawled web collection, we only consider the hyperlinks between pages. So we first map URLs into ordinal numbers, and represent the web's hyperlinked structure using three binary files. The first one, called a *link structure file* (*L*), represents the relationship between pages via their hyperlinks. Each record in this file consists of *dest_id* field (the target page) and a list of *src_id* fields (the set of authoritative pages.) The other two files, called the *out-degree file* (*O*) and *in-degree file* (*I*), contain the numerical out-degree and in-degree, respectively, corresponding to each *dest_id* in file *L*. An example of these files is textually shown in Fig. 1.



dest_id (4 bytes)	**src_id** (4 bytes each)
1	1028
2	106
3	311 312
4	35 96 487 5052

file *L*

out_deg (4 bytes)
1
5
3
1

file *O*

in_deg (4 bytes)
1
1
2
4

file *I*

Fig. 1. Three binary files representing the link structure of a web graph

As shown in Fig. 1, all values are expressed as 4-byte integers. In this example, *dest_id* 1 is the target of a hyperlink from page (*src_id*) 1028, and it also has a hyperlink to one other page. The *dest_id* 2 is the target of a hyperlink from page (*src_id*) 106 and has hyperlinks to five other pages.

In the following subsections we describe an approach to parallelizing the PageRank computation utilizing a partitioning of these data files.

3.2 PPR-M Algorithm

The PPR-M Algorithm [13] applies the adaptive PageRank technique (expressed in Equation (2)) to a cluster computing environment. Implementation of the PPR-M algorithm uses the MPI model for message-based communication between nodes in the cluster.

The algorithm first partitions the three binary files representing a web graph (preprocessing phase) for assignment to compute nodes. The files *L* and *I* are partitioned for assignment to compute nodes, while the file *O* is not partitioned. Let *p* be the number of compute nodes used in the computation. Then we partition *L* and *I* into *p* equal parts by *dest_id*. Each node is assigned an identifying number *i* ($0 \le i < p$) and allotted a partition of the *dest_id* with data L_i and I_i. Each node will receive a copy of the entire *O* file.

Pseudo-code of the PPR-M algorithm is shown in Fig. 2. Before beginning the computation, each node loads the files I_i and *O* into main memory. The algorithm also loads as much of the file L_i into memory as possible (line 1), while the remaining values are loaded from hard disk as required. The algorithm iteratively performs the adaptive PageRank computation (lines 7-12) until all ranks converge. After completing each iteration, every node exchanges its computed rank scores, called a synchronization process (line 14). Further details of this process are given in [13].

```
    PPR-M ( $i, L_i, I_i, O, p, n$ )
1:      load file $L_i$, $I_i$, and $O$ into main memory
2:      $V_i$ : source PageRank vector, is initialized to $[1/n]_{n\times 1}$
3:      $V_i'$ : target PageRank vector, is initialized to $[0]_{\frac{n}{p}\times 1}$
4:      $score$ : a temporary score
5:      While all pages do not converge
6:          For each record $l \in L_i$
7:              $score = 0$
8:              If $l.dest\_id$ converges then
9:                  $V_i'[l.dest\_id] = V_i[l.dest\_id]$
10:             Else
11:                 Compute all $l.src\_id$ as $\frac{V_i[l.src\_id]}{O[l.src\_id]}$ and add to $score$
12:                 $V_i'[l.dest\_id] = (\alpha \times score) + \frac{(1-\alpha)}{n}$
13:         Store $V_i'$ in $V_i$
14:         Synchronize $V_i'$ with other processess and also store in $V_i$
15:     Report $V_i'$ as local PageRank vector
```

Fig. 2. The parallel PageRank algorithm using only MPI

3.3 PPR-MT Algorithm

The use of parallel processing in the PPR-M algorithm reduces the elapsed computational time required to compute PageRank scores, but adds significant time for network communications during the synchronization process after every iteration. In the PPR-M algorithm, these communications are managed entirely by the MPI library, which also adds overhead. To reduce the communication overhead, we investigate the use of lightweight processes in combination with MPI-based inter-process communication. In this subsection, we present the PPR-MT algorithm (*threaded* PPR-M) that improves on PPR-M by using POSIX threads [11] for both computation and inter-process communication.

PPR-MT begins the pre-processing phase by partitioning a web graph and allocating partitions to computing nodes as done in PPR-M. After that, each node will load a portion of web graph into memory, the same as in PPR-M. The difference is that a node may create a number of threads for cooperative computing. The synchronization process between threads is done via shared memory within the node, while synchronization between nodes is still done by MPI. Pseudo-code of the PPR-MT algorithm in shown in Fig. 3.

```
     PPR-MT ( $i, L_i, I_i, O, p, n$ )
1:     load file $L_i$, $I_i$, and $O$ into main memory
2:     $V_i$ : source PageRank vector, is initialized to $\left[\frac{1}{n}\right]_{n\times 1}$
3:     $V_i'$ : target PageRank vector, is initialized to $[0]_{\frac{n}{p}\times 1}$
4:     $score$ : a temporary score
5:     Create threads
6:     While all pages do not converge
7:         For each record $l \in L_i$ will be assigned to a thread
8:             $score = 0$
9:             If $l.dest\_id$ converges then
10:                $V_i'[l.dest\_id] = V_i[l.dest\_id]$
11:            Else
12:                Compute all $l.src\_id$ as $\frac{V_i[l.src\_id]}{O[l.src\_id]}$ and add to $score$
13:                $V_i'[l.dest\_id] = (\alpha \times score) + \frac{(1-\alpha)}{n}$
14:        Store $V_i'$ in $V_i$
15:        Synchronize $V_i'$ with other processes and also store in $V_i$
16:    Join threads
17:    Report $V_i'$ as local PageRank vector
```

Fig. 3. The parallel PageRank algorithm using mixed MPI and threads

Before computation, the algorithm loads I_i, O, and all or part of L_i into memory to reduce disk I/O and improve CPU utilization (line 1). In lines 5 and 16, threads are created and terminated, respectively. During iterative computation, an unloaded thread will be assigned a record of L_i for rank score computation (line 7). Inter-thread communication occurs via shared memory used for accessing/storing the PageRank scores (line 10 and 13); while inter-process synchronization between nodes occurs after each iteration (line 15) via MPI.

The architecture of the algorithm is depicted in Fig. 4. It consists of three components: allotment, computation, and merge. The Allotment component performs pre-processing, including transforming the web graph into the link structure, in-degree, and out-degree (*L, I, O*) files, partitioning them, and allocating partitions to computing nodes. The Computation component invokes each node (i.e., processor-*i*) to compute rank scores for its partition. At any node, multiple threads collaborate in the computation. Finally the Merge component merges the computed scores from all computing nodes to obtain the global scores as output.

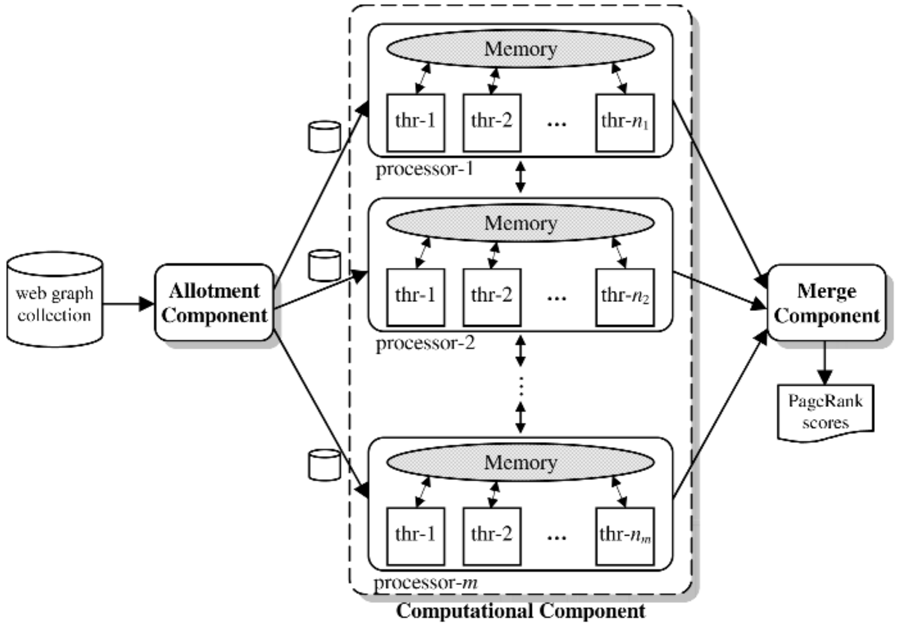


Fig. 4. An architecture of PPR-MT algorithm

4 Experiments and Results

In this section we present the experimental configuration, results obtained, and a discussion of the experimental results.

4.1 Experimental Setup

Environment and Configuration: The algorithm was tested on two clusters of x86-based SMP computers. The first cluster (WSC cluster) consists of dual 3.2GHz Intel quad-core processors, 4GB of main memory, and 120GB SATA disk in each node. The second cluster (MAEKA cluster [7]) consists of hosts with dual AMD Opteron240 processors, 3GB main memory, and 72GB SCSI disk. All hosts in both clusters run the Linux operating system and are inter-connected via switched Gigabit Ethernet. In our implementation, we use the MPICH library, version 1.2.7, and standard POSIX thread library. Experiments were run under unloaded conditions, without competition from other computing tasks in the cluster.

Data Setup: The test data is a subset of the web graph compiled from the Stanford WebBase Project [16], hereafter termed SF-Graph, consisting of approximately 28 million pages and 227 million hyperlinks. We also synthesize another larger and denser web graph based on the first one to investigate scalability, termed Syn-Graph. Syn-Graph contains approximately 62.6 million pages and 1.37 billion hyperlinks.

4.2 Evaluation Results and Discussion

The main objective of our experiments is to study the performance of the PageRank algorithms as a function of cluster size, threads per compute node, and size of the web graph. We measured the total time needed for the PageRank computation to converge using a relative tolerance of 0.0001, varying the number of compute nodes and threads per node. Due to limitations in the amount of time available on the clusters used, we limited the number of threads per machine in PPR-MT experiments to 2, 4, 8, and 16 threads. We repeated each experiment at least 3 times and averaged the run times.

Table 1. The average run time of PPR-M (1 thread per machine) and PPR-MT algorithms

			Average run time (seconds)				
		# of mach	1thr/mach	2thrs/mach	4thrs/mach	8thrs/mach	16thrs/mach
WSC cluster	SF-Graph	1	266.40252	126.32589	85.35699	83.34633	**83.26815**
		2	214.84687	113.87603	80.89533	79.93591	**77.96629**
		4	152.33151	104.42615	84.71894	68.88086	**67.00578**
	Syn-Graph	1	1750.74109	1186.18676	986.33184	**924.34139**	1120.25558
		2	435.94806	275.22307	182.84255	119.10821	**82.63472**
		4	346.88886	188.56568	108.77058	67.92434	**40.10643**
MAEKA cluster	SF-Graph	1	420.38131	219.53706	173.33427	**120.43113**	130.17395
		2	285.86809	174.88750	136.59907	110.10362	**100.15860**
		4	209.13478	137.59722	124.17826	95.84211	**81.45755**
		8	148.58302	109.82293	93.88049	**74.50150**	76.56715
	Syn-Graph	1	2342.90540	**2009.79438**	2210.43035	3229.86019	12551.50322
		2	1836.33356	1314.93663	**992.73288**	1043.49289	1572.52810
		4	677.85948	378.02183	262.63831	**227.48226**	276.16426
		8	622.81540	330.72717	218.10580	**178.85681**	190.38320

Table 1 shows the average total run times required for both SF-Graph and Syn-Graph data. The upper half of the table shows results using 1-4 machines in the WSC cluster for each web graph. The bottom half show results for experiments using 1-8 nodes from the MAEKA cluster.

The column “1 thr/mach” (1 thread per machine) gives average times using the PPR-M algorithm. The other columns give average times using multiple threads and the PPR-MT algorithm. Using the pseudo-code shown in Fig. 3, the total time needed in each experiment consists mainly of the time for PageRank vector initialization and thread creation (lines 2-5), the adaptive PageRank computation (lines 7-13), and vector synchronization between processors and/or machines (line 15). In each row of Table 1, the best time is shown in bold text. For example, the best run time for Syn-Graph data using 1 machine on the WSC cluster is 924.34 seconds, decreasing to 40.11 seconds for runs utilizing 4 machines.

Table 2 summarizes the best run times from Table 1, and gives a speedup factor for cluster-based run times relative to the single machine case. The results in Table 2 show that the performance, in terms of speedup as the number of nodes increases, is very poor for the smaller SF-Graph data. This indicates that the communication time required for costly PageRank vector synchronization is relatively high, reducing efficiency of processor utilization.

For the larger and denser Syn-Graph, the results are more encouraging: we obtain a speedup of 23.05 and 8.83 using 4 machines in the WSC and MAEKA clusters, respectively. This suggests that the processors are better utilized in the PageRank computation part of overall process. This suggests that it may be more cost effective to invest in more computing resources for cluster farms when we need to compute the PageRank of a very large web graph, such as those on commercial search engines, and using the proposed mixed model algorithm to incorporate multi-threading.

For both clusters, the speedup when increasing from 1 to 4 nodes is greater than the rate of increase in number of nodes. Normally, when increasing from 1 to 4 nodes the best one can hope for is a 4-fold speedup, yet the experimental times show an 8.83 and 23.05 fold acceleration. The unexpectedly high increase in performance is due to improved memory utilization. As the number of nodes increases, the size of each node’s data partition decreases. This can increase the percentage of data held in physical memory and reduce or eliminate paging. Utilizing more threads (and more CPU core) per node does not reduce the size of a node’s data partition and, conversely, can actually increase contention for memory access. No super-linear speedup was observed as a function of increasing threads per node.

Table 2. The best performance speedup as a function of machines utilized

		WSC cluster		MAEKA cluster	
	# of mach	Best time	Speedup	Best time	Speedup
SF-Graph	1	83.26815	1.00000	120.43113	1.00000
	2	77.96629	1.06800	100.15860	1.20240
	4	67.00578	1.24270	81.45755	1.47845
	8	-	-	74.50150	1.16149
Syn-Graph	1	924.34139	1.00000	2009.79438	1.00000
	2	82.63472	11.18587	992.73288	2.02450
	4	40.10643	**23.04721**	227.48226	**8.83495**
	8	-	-	178.85681	11.23689

5 Conclusion

We investigate the use of cluster technology for parallelization of page rank calculations as a solution to the challenging problem of rapid growth in the Internet page data that must be scored by a page rank algorithm. In this paper, we present the PPR-M and PPR-MT algorithms that efficiently run on SMP based clusters. The first algorithm uses simple message passing for inter-process communication while the second algorithm combines the standard thread library for inter-thread communication with MPI for cluster communications. Both algorithms exploit the power of SMP based clusters to compute the rank scores of a large-scale web graph in parallel. Our experiments show encouraging results, speeding the computational process up to 23 times using 4 machines, compared to base run times on a single machine.

In future work, we plan to explore some web graph partitioning algorithms for better load balancing of the computation between the compute nodes. We will also investigate way to reduce the communication overhead of PageRank synchronization, as well as study the convergence rate for accelerating the algorithm.

Acknowledgments. We would like to thank to all anonymous reviewers for their comments and suggestions. We also thank to Dr. James Edward Brucker for his reading of the final version of this paper. The research is funded by the Kasetsart University Research and Development Institute (KURDI).

References

1. Chen, Y., Gan, Q., Suel, T.: I/O-efficient techniques for computing PageRank. Proceedings of the 11th International Conference on Information and Knowledge Management (2002)
2. Gleich, D., Zhukov, L., Berkhin, P.: Fast parallel PageRank: a linear system approach. Technical Report, Yahoo! Research Labs (2004)
3. Haveliwala, T.H.: Efficient computation of PageRank. Technical Report, Stanford University (1999)
4. Kamvar, S.D., Haveliwala, T.H., Golub, G.H.: Adaptive methods for the computation of PageRank. Technical Report, Stanford University (2003)
5. Kamvar, S.D., Haveliwala, T.H., Manning, C.D., Golub, G.H.: Exploiting the block structure of the web for computing PageRank. Technical Report, Stanford University (2003)
6. Kleinberg, J.M.: Authoritative sources in a hyperlinked environment. Journal of the ACM (JACM), Vol. 46 (1999) 604-632
7. MAEKA: Massive Adaptable Environment for Kasetsart Application. Available source: http://maeka.ku.ac.th (2003)
8. Manaskasemsak, B., Rungsawang, A.: Parallel PageRank computation on a gigabit PC cluster. Proceedings of the International Conference on Advanced Information Networking and Applications (2004)
9. MPI library: Massive Passing Interface. Available source: http://www-unix.mcs.anl.gov/mpi (2006)
10. Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank citation ranking: bringing order to the web. Technical Report, Stanford University (1999)

11. POSIX Threads Programming. Available source: http://www.llnl.gov/computing/tutorials/pthreads (2006)
12. Ross, S.M.: Introduction to probability models. 8th Edition, Academic Press (2003)
13. Rungsawang, A., Manaskasemsak, B.: Parallel adaptive technique for computing PageRank. Proceedings of the 14th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (2006)
14. Sankaralingam, K., Sethummadhavan, S., Browne, J.C.: Distributed PageRank for P2P systems. Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (2003)
15. Shi, S., Yu, J., Yang, G., Wang, D.: Distributed page ranking in structured P2P networks. Proceedings of the International Conference in Parallel Processing (2003) 179-186
16. The Stanford WebBase Project. Available source: http://www-diglib.stanford.edu/~testbed/ doc2/WebBase (2004)