Efficient Parallel Computation of PageRank

Christian Kohlschütter, Paul-Alexandru Chirita, and Wolfgang Nejdl

L3S Research Center/University of Hanover,
Deutscher Pavillon, Expo Plaza 1,
30539 Hanover, Germany
{kohlschuetter, chirita, nejdl}@l3s.de

Abstract. PageRank inherently is massively parallelizable and distributable, as a result of web's strict host-based link locality. We show that the Gauß-Seidel iterative method can actually be applied in such a parallel ranking scenario in order to improve convergence. By introducing a two-dimensional web model and by adapting the PageRank to this environment, we present efficient methods to compute the exact rank vector even for large-scale web graphs in only a few minutes and iteration steps, with intrinsic support for incremental web crawling, and without the need for page sorting/reordering or for sharing global rank information.

1 Introduction

Search engines are the enabling technology for finding information on the Internet. They provide regularly updated snapshots of the Web and maintain a searchable index over all retrieved web pages. Its size is currently reaching several billions of pages from about 54 million publicly accessible hosts, but these amounts are rapidly increasing [19]. When searching such huge datasets, one would usually receive quite a few pages in response to her query, some of them being much more relevant than others. This gave birth to a lot of ordering (or ranking) research, the most popular algorithm being Google's PageRank [20], which recursively determines the importance of a web page by the importance of all the pages pointing to it.

Although improvements for a centralized computation of PageRank have been researched in detail [1, 5, 9, 11, 12, 16, 13, 18], approaches on distributing it over several computers have caught researchers' attention only recently. In this paper we introduce a new approach to computing the exact PageRank in a parallel fashion. We obtain exact results faster than all the other existing algorithms, improving by orders of magnitude over the other algorithms generating exact PageRank scores. We achieve this by modeling the web graph in a two-dimensional fashion (with the URL's hostname as the primary criterion), thus separating it into reasonably disjunct partitions, which are then used for distributed, incremental web crawling [6] and PageRank computation.

The remainder of the paper is organized as follows. After reviewing the PageRank algorithm, common web graph representation techniques and existing parallel versions of PageRank in Section 2, we introduce our two-dimensional

M. Lalmas et al. (Eds.): ECIR 2006, LNCS 3936, pp. 241–252, 2006.

web graph model in Section 3. We then present a refined PageRank algorithm in Section 4, and show that the convergence improvements of the Gauß-Seidel method for solving linear systems [1] can also be efficiently applied in a parallelized PageRank scenario. Experimental results are discussed in Section 5. Finally, Section 6 concludes with discussion of further work.

2 Related Work

2.1 PageRank

The main concept behind the PageRank paradigm [20] is the propagation of importance from one Web page towards others, via its out-going (hyper-)links. Each page $p \in P$ (P is the set of all considered pages) has an associated rank score $r(p)$, forming the rank vector $\vec{r}$. Let L be the set of links, where (s, t) is contained iff page s points to page t and $L(p)$ be the set of pages p points to (p's outgoing links). The following iteration step is then repeated until all scores r stabilize to a certain defined degree $\delta < \varepsilon$:

$$\forall t \in P : r^{(i)}(t) = (1-\alpha) \cdot \tau(t) + \alpha \sum_{(s,t)\in L} \frac{r^{(i-1)}(s)}{|L(s)|} \tag{1}$$

The formula consists of two portions, the jump component (left side of the summation) and the walk component (right side), weighted by α (usually 0.85). $r^{(i-1)}(s) \cdot |L(s)|^{-1}$ is the uniformly distributed fraction of importance a page s can offer to one of its linked pages t for iteration i. Intuitively, a "random surfer" follows an outgoing link from the current page (walk) with probability α and will get bored and select a random page (jump) with probability $1 - \alpha$. The main utility of α is however to guarantee convergence and avoid "rank sinks" [3].

This "random-walk" is in fact the interpretation of the Markov chain associated to the web graph, having $\vec{r}$ as the state vector and A (see Equation 2) the transition probability from one page to another. We can therefore also write Equation 1 in matrix terms as follows:

$$\vec{r} = (1-\alpha) \cdot \vec{\tau} + \alpha A\vec{r} \tag{2}$$

Equation 1 also represents the linear system representation of this matrix computation using the Jacobi iterative method. This enables the consideration of using other stationary iterative solvers, such as the Gauß-Seidel method, which converges two times faster than Jacobi but was said not to be efficiently parallelizable here [1, 5, 18]. Actually, there already are parallel Gauss-Seidel implementations for certain scenarios such as the one described in [14], using block-diagonally-bordered matrices; however, they all admit their approach was designed for a static matrix; after each modification, a specific preprocessing (sorting) step is required, which can take longer than the real computation. Because the web is highly dynamic, almost 40% of all links change in less than one week [6], disregarding this preparation step would veil the real overall processing time. Steady reorganization of coordinates in a huge link matrix simply imposes an unjustified management overhead.

2.2 Web Graph Representation

Computing the PageRank vector for a large web graph using a materialized in-memory matrix A is definitely not feasible. A common solution is to store the links in a format like "Destination Page ID, Out-degree, Source Page IDs..." (which resembles L). Because pages only link to a few others (the link matrix is sparse), this results in much lower memory requirements of the link structure, in the magnitude of $\mid L \mid \cdot \overline{n}^{-1} \cdot c$ bytes ($\overline{n}$ = average outdegree; $c = const.$)

Of course, compression techniques [15] or intelligent approaches to disk-based "swapping" [9, 5, 18] can improve the space requirements even further (e.g. by relying on a particular data order, or on the presence of caches). But with the permanent growth of the web, even such techniques will soon hit memory limits of a single computer, or unacceptably slow down the computation process. See [18] for a thorough discussion of these optimizations.

In this paper, we thus propose a new strategy for keeping the web graph and rank information completely in RAM of several networked machines, based on the separation between global (host) and local information about each page.

2.3 Other Parallel PageRank Algorithms

Existing approaches to PageRank parallelization can be divided into two classes: Exact Computations and Approximations.

Parallel Computations. In this scenario, the web graph is initially partitioned into blocks: grouped randomly [21], lexicographically sorted by page [17, 22, 26] or balanced according to the number of links [8].

Then, standard iterative methods such as Jacobi (Equation 1) or Krylov subspace [8] are performed over these pieces in parallel. The partitions periodically must exchange information: Depending on the strategy this can expose suboptimal convergence speed because of the Jacobi method and result in heavy inter-partition I/O (e.g., in [17], computing the rank for a page t requires access to all associated source page ranks $r(s)$ across all partitions).

PageRank Approximations. The main idea behind these approaches is that it might be sufficient to get a rank vector which is comparable, but not equal to PageRank. Instead of ranking pages, higher-level formations are used, such as the inter-connection/linkage between hosts, domains, server network addresses or directories, which is orders of magnitudes faster. The inner structure of these formations (at page level) can then be computed in an independently parallel manner ("off-line"), as in BlockRank [10], SiteRank [25], the U-Model [4], ServerRank [24] or HostRank/DirRank [7].

We will try to take the best out of both approaches: the exactness of a straight PageRank computation but the speed of an approximation, without any centralized re-ranking.

3 The Two-Dimensional Web

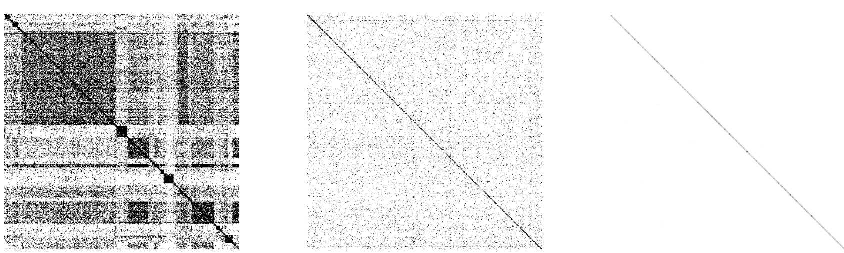
3.1 Host-Based Link Locality

Bharat et al. [2] have shown that there are two different types of web links dominating the web structure, “intra-site” links and “inter-site” ones. A “site” can be a domain (`.yahoo.com`), a host (`geocities.yahoo.com`) or a directory on a web server (`http://www.geocities.com/someuser/`). In general, we can define a site as an interlinked collection of pages identified by a common name (domain, host, directory etc.), and under the control of the same authority (an authority may of course own several sites).

Due to web sites' hypertext-navigable nature, it is supposable that a site contains more internal than external links. In fact, about 93.6% of all non-dangling links are intra-host and 95.2% intra-domain [10]. This assumed block structure has been visualized by Kamvar et al. [10] using dotplots of small parts (domain-level) of the ”LargeWeb” graph's link matrix [23]. In these plots, the point (i, j) is black, if there is a link from page p_i to p_j, clear otherwise.

We performed such a plot under the same setting, but on whole-graph scale. The outcome is interesting: a clear top-level-domain (TLD) dominant structure (see Figure 1a). For example, the `.com` TLD represents almost 40% of the complete structure and has high connectivity with `.net` and `.org`, whereas the `.jp` domain shows almost no interlinkage with other TLDs. However, if we only inspect the `.com` domain (see Figure 1b, the dotplot depicts a diagonally dominant structure. The diagonal represents links from target pages near by the source page (which are *inter-host* pages). Both results are primarily caused by the lexicographical order of URLs (with hostnames reversed, e.g. `http://`*com.yahoo.www*`/index.html`).

But is this costly *sorting* over all URLs necessary at all? To further analyze the impact of hostname-induced link locality, we redraw the LargeWeb dotplot in a *normalized* (histographical) fashion, where a dot's greyscale value depicts the cumulative percentage of links in a specific raster cell. In addition, we do not sort the pages lexicographically, but only group them per host and permute all hosts randomly to avoid any lexicographical or crawl-order-dependent



(a) LargeWeb, sorted (b) .com subgraph of (a) (c) LargeWeb, normalized

Fig. 1. Linkage dotplots

relationship between them. The clear diagonal dominance now also becomes visible on whole-graph scale (Figure 1c).

3.2 From Numbers to Tuples

It should be obvious that the web was already designed to be two-dimensional: Hostnames are "namespaces" aimed to disambiguate different local contexts (i.e., paths like "`/dir/index.html`"). Previous approaches to web graph partitioning always resulted in having *one* unique ID associated to each page, eventually sorted lexicographically [10, 17, 22] or in crawling order to exploit specific graph properties [18].

Such a single page ID provides a very compact representation of the web graph, which can be visualized in a matrix dotplot as shown above. But it also requires continuous reorganization (sorting) for newly added or removed pages in the course of incremental crawling. Otherwise, a mixture of hosts along the URL IDs would render a host no longer characterizable by a closed interval of IDs, thereby losing the advantage of link locality. One may introduce gaps in the numbering to reduce the sorting costs, but still, all subsequent pages will have to be renumbered once the gap is filled. In a distributed scenario, this can cause extensive network I/O by repeatedly moving pages from one partition to another.

We therefore propose a different page identification scheme, based on the affiliation of each page to a specific host and independently of pages from other hosts. More specifically, we propose using a *tuple* consisting of two independent, positive integers, a HostID (only dependent on the URL's hostname) and a LocalID (only identifying the remaining local components – path and query string). The addition of new local pages to a specific host, as well as of new hosts, is very easy, since renumbering is no longer necessary.

As an implementation-specific note, we expect that for current web graphs, it is sufficient to store the tuples as two `uint32` four-byte integers. We then can address a maximum of 4.29 billion hosts and a maximum of 4.29 billion pages per host in 8 bytes. For small hosts, we could even reduce the local part to 16 bit, thereby further cutting down memory footprint.

4 Partitioned PageRank

We will now consider the impact of such a partitioning scheme on the PageRank algorithm. We will first present an analysis that unifies two of the most common algorithms for solving linear systems, Gauß-Seidel and Jacobi. Then, we will apply this analysis to propose an improved parallel PageRank algorithm, and finally we will discuss several optimization issues.

4.1 Unifying Jacobi and Gauss-Seidel

It has been observed that the Gauß-Seidel iteration method compared to the Jacobi method can speed-up PageRank convergence by a factor of 2, as it uses scores of the current iteration as soon as they become available [1]:

$$\forall (s,t) \in L: \quad r^{(i)}(t) = (1-\alpha)\ \tau(t)\ +\ \alpha \left(\sum_{s<t} \frac{r^{(i)}(s)}{|L(s)|} + \sum_{s>t} \frac{r^{(i-1)}(s)}{|L(s)|} \right) \tag{3}$$

As opposed to the Jacobi iteration, the Gauß-Seidel variant requires iterating over the links $(s,t) \in L$ in a strictly ascending order. At first glance, this seems to be a major drawback when we want to apply it to a distributed, partitioned web graph. To clarify the impact of the restriction of link order, we derive a common base algorithm for both, Jacobi (equation 1) and Gauß-Seidel (equation 3) algorithms: We define an intermediate ranking vector $r^{(i-1,i)}$ that combines the vectors of the previous and the current iteration, depending on the state of a ranked page p in the set of available pages P ($P = P' \cup P''$; $\nexists\ p : p \in P' \wedge p \in P''$; P' contains all pages which have already been ranked for iteration i; P'' contains all other pages, whose score has not been touched since iteration $i-1$):

$$r^{(i-1,i)}(p) := \begin{cases} r^{(i)}(p) & \text{if } \ p \in P' \\ r^{(i-1)}(p) & \text{if } \ p \in P'' \end{cases} ;\ r^{(i)}(t) = (1-\alpha)\ \tau(t) + \alpha \sum_{(s,t)\in L} \frac{r^{(i-1,i)}(s)}{|L(s)|} \tag{4}$$

Under this setting, for the Gauß-Seidel method, $P' = \{\ p \mid p < k\ \}$ and $P'' = \{\ p \mid p \geq k\ \}$, with $k \in \{1, 2, ..., |P|\}$, whereas for the Jacobi method, we have $P' = \emptyset$ and $P'' = P$. Both iteration methods, Jacobi and Gauß-Seidel, can then be simplified to this joint formula:

$$r^{(\star)}(t) = (1-\alpha)\ \tau(t) + \alpha \sum_{(s,t)\in L} \frac{r^{(\star)}(s)}{|L(s)|}, \text{ with } r^{(\star)}(t) = r^{(i-1,i)}(t) \tag{5}$$

From Equation 4, we know that before each iteration i, $\vec{r}^{(\star)} = \vec{r}^{(i-1)}$ and after the iteration $\vec{r}^{(\star)} = \vec{r}^{(i)}$. The state of $\vec{r}^{(\star)}$ during the iteration then only depends on the order of links $(s,t) \in L$ (the way how P' and P'' are determined). This iteration method has worst-case convergence properties of Jacobi and best-case of Gauß-Seidel, depending on the order of elements, random order vs. strictly ascending order, while always providing the same per-iteration running time as the Jacobi iteration.

We further generalize the impact of the rules for P' and P'': We argue that if only a small fraction F of all links concerned ($|F| \ll |\ L\ |$) is not in strictly ascending order, the overall convergence speed still remains in the magnitude of standard Gauß-Seidel. In our case, in order to be able to parallelize the Gauß-Seidel algorithm, we will assign inter-host/inter-partition links (about 6%) to this small fraction.

4.2 Reformulating PageRank

For such an optimization, let us reformulate our above mentioned unified PageRank equation using our new two-dimensional page numbering scheme. Thus, page variables "p" will be replaced by page tuples $\mathsf{p} = (p_x, p_y)$, with p_x representing the page's HostID, $host(\mathsf{p})$, and p_y its LocalID, $local(\mathsf{p})$. To account for the separation of inter- and intra-host links, the formula now reads as follows:

$$
\begin{aligned}
r^{(\star)}(\mathsf{t}) &= (1-\alpha)\ \tau(\mathsf{t})\ +\alpha\ \left(v_I^{(\star)}(\mathsf{t}) + v_E^{(\star)}(\mathsf{t})\right) \\
v_I^{(\star)}(\mathsf{t}) &= \sum_{(\mathsf{s},\mathsf{t})\in L} \frac{r^{(\star)}(\mathsf{s})}{|L(\mathsf{s})|} \quad \forall\ host(\mathsf{s}) = host(\mathsf{t}) \\
v_E^{(\star)}(\mathsf{t}) &= \sum_{(\mathsf{s},\mathsf{t})\in L} \frac{r^{(\star)}(\mathsf{s})}{|L(\mathsf{s})|} \quad \forall\ host(\mathsf{s}) \neq host(\mathsf{t})
\end{aligned}
\tag{6}
$$

Since $v_I^{(\star)}(\mathsf{t})$ solely requires access to *local* (intra-host) rank portions, it can efficiently be computed from scores stored in RAM. The local problem of ranking intra-host pages is solvable via a fast, non-parallel Gauß-Seidel iteration process. There is no need for intra-host vote parallelization – instead, we parallelize on the host-level, thus necessitating only inter-host communication, which is limited to the exchange of external votes.

Our approach produces the same ranks as the original PageRank, while being more scalable than the other parallel PageRank algorithms. This is mainly due to the parallelization of the Gauß-Seidel algorithm, in which we take advantage of web's host-oriented block structure.

4.3 Reaching Optimal Performance

Communication Cost Optimization. While votes between hosts of the same partition (server) can easily be conveyed in RAM, votes across hosts of different partitions require network communication. The gross total for exchanging external votes over the network must not be underestimated. With the LargeWeb graph setup, almost 33 million are exchanged between partitions. For bigger web graphs, this could rise up to a few billion and can easily lead to network congestion if too much information is transmitted per vote.

As opposed to other approaches, where a vote consisted of target page ID (sometimes along with source page ID) and score, we simply reduce this to transmitting a single value per page (the score), because the link *structure* does not change during the iteration cycle. More generally, the link structure of all the pages that exchange votes between two partitions pages only needs to be determined whenever the graph changes (in the case of incremental web crawling) and then to be sent to the specific target partition. Moreover, the source page does not need to be specified in order to compute the PageRank score, but only the target page ID (see Equation 6). Additionally, by grouping the list of target pages by host, we need to transmit each target host ID only once.

Most notably, each partition has to transmit only one single value per target page, not per link to that page, since all votes from local pages that link to a specific page can be aggregated to a single value (surprinsingly, this simple but very effective approach did not appear in any previous work):

$$
v_E^{(\star)}(\mathsf{t}) = \sum_{\beta\in\Pi} \sum_{(\mathsf{s},\mathsf{t})\in L_\beta} \frac{r^{(\star)}(\mathsf{s})}{|L|} = \sum_{\beta\in\Pi} v_\beta^{(\star)}(\mathsf{t}) \ \ \forall\ host(s) \neq host(t) \tag{7}
$$

Table 1. LargeWeb Inter-Partition links and votes

Type	Amount	Percent
Total Links	601,183,777	100%
Inter-Partition Links	32,716,628	5.44%
Inter-Partition Votes	3,618,335	0.6%

with Π being the set of partitions containing links towards t, and β each one of these partitions.

Transferring $v_\beta(\mathtt{t})$ (the sum of votes from partition L_β to $\mathtt{t}$) as a single value reduces the network load dramatically. Using this optimization, we can show a reduction of vote exchanges by 89% with the DNR-LargeWeb graph. Table 1 depicts the difference between inter-partition links and votes and their quota of all links.

Computational Load Balancing. In order to keep the convergence behavior of the centralized PageRank in our parallel scenario, inter-partition votes must be exchanged after every iteration (see [17] for a discussion of consequences of not doing so). To keep the overall computation time still low, all intra-partition computations and after that all network communication should terminate isochronously (at the same time). Because intra-partition computation is directly proportional to the number of pages per partition (see Equation 6), this either means that all available servers must be equally fast, or the graph has to be at least partitioned adequately to the performance of the servers. Moreover, other slow-down factors could also influence the running time, such as different network throughput rates of cheap NICs and system boards (even with the same nominal speed).

A good strategy to load-balancing Parallel PageRank in a heterogeneous environment could be running a small test graph on all new servers, measure computation speeds, and balance the real graph accordingly. In any case, memory overflows due to bad balancing parameters like in [8] are avoided, and no manual interaction to find these parameters is necessary.

5 Experiments

We first converted the Stanford DNR-LargeWeb graph [23] into the new tuple representation, resulting in 62.8M pages and 601M links distributed over 470,000 hosts with averaged 137.5 pages each (maximum was 5084 pages per host); the inter-host link percentage[1] is 6.19% (see Table 2).

For our PageRank experiments, we sorted the available hosts by their page count in descending order and distributed the pages host-wise in a round-robin manner over 8 partitions of equal size ($\frac{1}{8}$ of the graph just fitted into our smallest server's RAM).

[1] Unfortunately, the last 8 million pages of DNR-LargeWeb could not be converted, since there was no URL associated with them – thus, our numbers slightly differ from the ones in [10].

Table 2. LargeWeb link distribution

Type	Amount	Percent
Total	601,183,777	100%
Intra-Host	563,992,416	93.81%
Inter-Host	37,191,361	6.19%
Inter-Partition	32,716,628	5.44%
Intra-Partition	4,474,733	0.74%

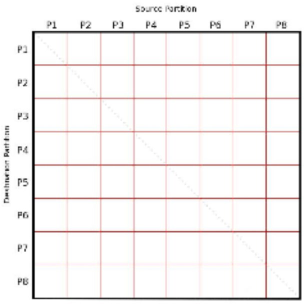


Fig. 2. Partitioned LargeWeb-Dotplot

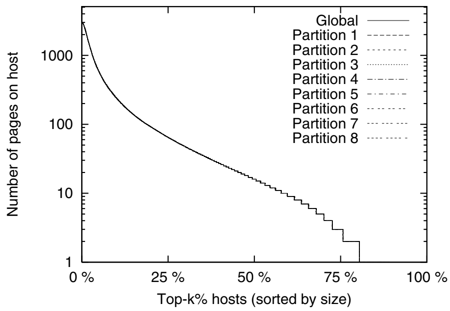


Fig. 3. Partitioned Host Distribution

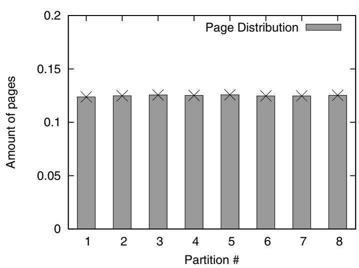


Fig. 4. Pages per Partition

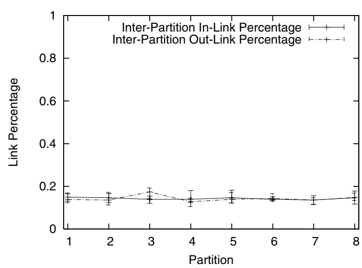


Fig. 5. Inter-partition link distribution

Although the pages-per-host distribution was not strictly exponential, it resulted in an equal page and link distribution (see Figures 2, 3, 4, 5). Remarkably, the intra-partition ratio (inter-host links inside the same partition) is negligible, as the inter-partition link rate nearly equals to the inter-host ratio. This means that hosts can arbitrarily be shifted from one partition to another one (which is necessary for fast re-balancing with incremental web crawling).

5.1 Implementation

We have implemented Partitioned Parallel PageRank in Java using a P2P-like network with a central coordinator instance. This coordinator is only responsible for arranging the iteration process at partition-level and does not know anything about the rank scores or the link structure (it is much simpler than the coordinator in [26]). Before the computation, all nodes announce themselves to the coordinator, communicating the hosts they cover. The iteration process is started as soon as all nodes are ready. The coordinator then broadcasts the global host structure to all known nodes and instructs them to iterate. Whenever a node's subgraph changes, it sends lists of external outgoing link targets to the corresponding nodes.

For every iteration step, a node will compute its votes using our reformulated PageRank (Equation 6); the partition itself is again divided into subpartitions processed in parallel. The nodes then aggregate all outgoing inter-partition votes by target page and send them directly to the other nodes responsible for these target pages, in the order specified beforehand. Finally, each node reports its local rank status (using the sum and number of its PageRank scores) to the coordinator, in order to compute the global residual δ. As soon as all nodes have succeeded, the coordinator decides whether to continue iterating, by broadcasting another "iterate" command unless the residual reached the threshold ε.

The addition of new pages during incremental crawling may happen at any time. If the addition covers new hosts, the coordinator selects a node according to the current balancing. From then on, this node is responsible for all pages of that host. The assignment is broadcasted to all nodes in case that there were dangling links to that (previously uncovered) host.

5.2 Results

We conducted most of the experiments on four Linux machines, an AMD Dual Opteron 850 2.4 GHz, 10GB RAM ("A"), an Intel Dual Xeon 2.8 GHz, 6GB

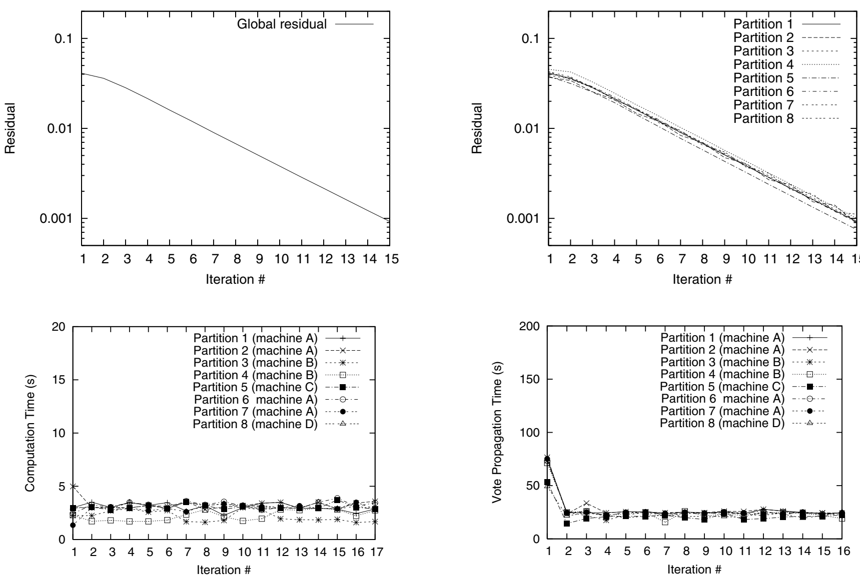


Fig. 6. Partitioned PageRank convergence, vote calculation and network communication times using 8 partitions on 4 machines; $\varepsilon = 0.001$

RAM ("B") and two Intel Xeon 3.0 GHz, 1.5GB RAM ("C" and "D"). They were connected via 100MBit Ethernet LAN and not under load before our experiments. We divided the LargeWeb graph into eight partitions and distributed them among the four servers according to available memory (Machine A holds four partitions, B two, C and D one) and performed unbiased PageRank computations.

We examined the convergence behavior, rank distribution and elapsed time both globally and per-partition. All per-partition results matched almost perfectly with the global counterpart and therefore confirmed our assumptions (see Figure 6). The PageRank computation converged below $\varepsilon = 10^{-3}$ after 17 iterations, and the entire computation took less than 9 minutes, with only 66 seconds accounted for rank computation, the rest being network I/O. With a Gigabit-Ethernet connection, network communication costs would probably go down to the same magnitude as computation costs.

Compared to the running times of a centralized PageRank computation with disk I/O, using our networked servers, Parallel PageRank is about 10 times faster per iteration. The recomputation itself (ignoring network transmission) was about 75 times faster. Thus, for further experiments, it might be interesting how our algorithm performs on a massive parallel machine.

6 Conclusions and Further Work

In this paper, we have presented an efficient method to perform the PageRank calculation in parallel over arbitrary large web graphs. We accomplished this by introducing a novel two-dimensional view of the web, having the host ID as the only discriminator, as well as by adapting the Gauß-Seidel method for solving linear systems in this scenario. Additionally, we have presented optimizations for the distributed computation, such as vote aggregation and utilizing the partitioning scheme for fast re-balancing in the course of incremental crawling.

Our next goal is to combine our approach with other PageRank specific enhancements that reduce convergence time, under extensive memory demanding scenarios.

References

1. Arvind Arasu, Jasmine Novak, Andrew Tomkins, and John Tomlin. PageRank computation and the structure of the web: Experiments and algorithms, 2001.
2. Krishna Bharat, Bay-Wei Chang, Monika Rauch Henzinger, and Matthias Ruhl. Who links to whom: Mining linkage between web sites. In *Proc. of the IEEE Intl. Conf. on Data Mining*, pages 51–58, 2001.
3. Sergey Brin, Rajeev Motwani, Lawrence Page, and Terry Winograd. What can you do with a web in your pocket? *Data Engineering Bulletin*, 21(2):37–47, 1998.
4. Andrei Z. Broder, Ronny Lempel, Farzin Maghoul, and Jan Pedersen. Efficient pagerank approximation via graph aggregation. In *Proc. of the 13th International World Wide Web Conference*, pages 484–485, 2004.

5. Yen-Yu Chen, Qingqing Gan, and Torsten Suel. I/o-efficient techniques for computing pagerank, 2002.
6. Junghoo Cho and Hector Garcia-Molina. The evolution of the web and implications for an incremental crawler. In *Proceedings of the 26th International Conference on Very Large Databases*, 2000.
7. Nadav Eiron, Kevin S. McCurley, and John A. Tomlin. Ranking the web frontier. In *Proc. of the 13th Intl. Conf. on the World Wide Web*, pages 309–318, 2004.
8. David Gleich, Leonid Zhukov, and Pavel Berkhin. Fast parallel PageRank: A linear system approach. Technical report, Yahoo! Research Labs, 2004.
9. Taher H. Haveliwala. Efficient computation of PageRank. Technical Report 1999-31, Stanford Library Technologies Project, 1999.
10. Sepandar Kamvar, Taher Haveliwala, Christopher Manning, and Gene Golub. Exploiting the block structure of the web for computing PageRank. Technical report, Stanford University, 2003.
11. Sepandar D. Kamvar, Taher H. Haveliwala, and Gene H. Golub. Adaptive methods for the computation of PageRank. Technical report, Stanford University, 2003.
12. Sepandar D. Kamvar, Taher H. Haveliwala, Christopher D. Manning, and Gene H. Golub. Extrapolation methods for accelerating PageRank computations. In *Proc. of the 12th Intl. Conf. on the World Wide Web*, pages 261–270, 2003.
13. Sung Jin Kim and Sang Ho Lee. An improved computation of the PageRank algorithm. In *Proc. of the European Conference on Information Retrieval (ECIR)*, pages 73–85, 2002.
14. D. P. Koester, S. Ranka, and G. C. Fox. A parallel gauss-seidel algorithm for sparse power system matrices. In *Proc. of the ACM/IEEE Conf. on Supercomputing*, pages 184–193, 1994.
15. Amy N. Langville and Carl D. Meyer. Deeper inside PageRank, 2004.
16. Chris P. Lee, Gene H. Golub, and Stefanos A. Zenios. A fast two-stage algorithm for computing PageRank. Technical report, Stanford University, 2003.
17. Bundit Manaskasemsak and Arnon Rungsawang. Parallel PageRank computation on a gigabit pc cluster. In *Proc. of the 18th International Conference on Advanced Information Networking and Application (AINA'04)*, 2004.
18. Frank McSherry. A uniform approach to accelerated pagerank computation. In *Proc. of the 14th international conference on World Wide Web*, pages 575–582, New York, NY, USA, 2005. ACM Press.
19. Netcraft. Web server survey, 2005.
20. Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.
21. Karthikeyan Sankaralingam, Simha Sethumadhavan, and James C. Browne. Distributed pagerank for p2p systems. In *Proc. of the 12th IEEE Intl. Symp. on High Performance Distributed Computing (HPDC)*, page 58, 2003.
22. Shu-Ming Shi, Jin Yu, Guang-Wen Yang, and Ding-Xing Wang. Distributed page ranking in structured p2p networks. In *Proc. of the 2003 International Conference on Parallel Processing (ICPP'03)*, pages 179–186, 2003.
23. Taher H. Haveliwala et al. 2001 Crawl of the WebBase project, 2001.
24. Yuan Wang and David J. DeWitt. Computing PageRank in a distributed internet search system. In *Proceedings of the 30th VLDB Conference*, 2004.
25. Jie Wu and Karl Aberer. Using SiteRank for P2P Web Retrieval, March 2004.
26. Yangbo Zhu, Shaozhi Ye, and Xing Li. Distributed pagerank computation based on iterative aggregation-disaggregation methods. In *Proc. of the 14th ACM international conference on Information and knowledge management*, 2005.