John von Neumann Institute for Computing

Asynchronous Iterative Computations with Web Information Retrieval Structures: The PageRank Case

G. Kollias, E. Gallopoulos, D. Szyld

published in

Parallel Computing:
Current & Future Issues of High-End Computing,
Proceedings of the International Conference ParCo 2005,
G.R. Joubert, W.E. Nagel, F.J. Peters, O. Plata, P. Tirado, E. Zapata (Editors),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **33**, ISBN 3-00-017352-8, pp. 309-316, 2006.

http://www.fz-juelich.de/nic-series/volume33

Asynchronous iterative computations with Web information retrieval structures: The PageRank case

Giorgos Kollias[a], Efstratios Gallopoulos[a], Daniel B. Szyld[b]

[a]Computer Engineering and Informatics Department, University of Patras, GREECE

[b]Department of Mathematics, Temple University, Philadelphia, USA

1. Introduction

There are several ideas being used today for Web information retrieval, and specifically in Web search engines [20]. The PageRank algorithm [22] is one of those that introduce a content-neutral ranking function over Web pages. This ranking is applied to the set of pages returned by the Google search engine in response to posting a search query. PageRank is based in part on two simple common sense concepts: (i) A page is important if many important pages include links to it. (ii) A page containing many links has reduced impact on the importance of the pages it links to.

In this paper we focus on asynchronous iterative schemes [9,15] to compute PageRank over large sets of Web pages. The elimination of the synchronizing phases is expected to be advantageous on heterogeneous platforms. The motivation for a possible move to such large scale distributed platforms lies in the size of matrices representing Web structure. In orders of magnitude: 10^{10} pages with 10^{11} nonzero elements and 10^{12} bytes just to store a small percentage of the Web (the already crawled); distributed memory machines are necessary for such computations. The present research is part of our general objective, to explore the potential of asynchronous computational models as an underlying framework for very large scale computations over the Grid [14]. The area of "internet algorithmics" appears to offer many occasions for computations of unprecedent dimensionality that would be good candidates for this framework.

After giving a formulation of PageRank and its common interpretations in Section 2, we present its treatment under synchronous computational models. We next consider the asynchronous approach and comment on key aspects, specifically convergence, termination detection and implementation. In Section 5, we describe the experimental framework and present preliminary numerical experiments, while in Section 6 we draw our conclusions and discuss our future work on this topic. In this paper, as is common practice, we do not address the effects of finite precision arithmetic and roundoff error.

2. Formulation and Interpretations

In order to appreciate the PageRank computation, we present its standard formulation using the following set of four $n \times n$ matrices, where n is the number of pages being modeled.

An *adjacency matrix* A can be obtained through a web crawl or synthetically generated using statistical results, e.g., as in [10]. Thus, $A_{ij} = 1$ *iff* page i points to page j, and $A_{ij} = 0$ otherwise.

A *transition matrix* P has nonzero elements $P_{ij} = A_{ij}/\deg(i)$ when $\deg(i) \neq 0$, and zero otherwise (in which case page i is called a *dangling* page); here $\deg(i) = \sum_j A_{ij}$ is the outdegree of page i.

A *stochastic matrix* S is given by $S = P^T + w\, d^T$; $w = \frac{1}{n}e$, where e is the size n vector of all 1's, and d is the *dangling index vector* whose nonzero elements are $d_i = 1$ *iff* $\deg(i) = 0$.

The *Google matrix* G is $G = \alpha\, S + (1-\alpha)\, v\, e^T$. For a random web surfer about to visit his next page, the relaxation parameter α is the probability of choosing a link-accessible page. In choosing

otherwise, i.e., with probability $1-\alpha$, from the complete Web page set vector v contains respective conditional probabilities of such *teleportations*. Typically $v = w$ and $\alpha = 0.85$.

The PageRank vector x is the solution of the linear system

$$x = G\,x\,, \tag{1}$$

where the matrix G is an irreducible stochastic matrix, and thus its largest eigenvalue in magnitude is $\lambda_{max} = 1$ [26]. Thus, the PageRank vector x is the eigenvector corresponding to $\lambda_{max} = 1$, and when normalized, it is the reachability probability in a random walk on the Web, i.e., the invariant measure or stationary probability distribution of a Markov process modeled by the matrix G. It can also be computed as the solution to a system of linear equations. Using the fact that x is normalized to unity, i.e., $e^T\,x = 1$, equation (1) yields

$$(I-R)\,x = b \tag{2}$$

where $b = (1-\alpha)\,v$ and $R = \alpha\,S$ is the *relaxed stochastic matrix* [13].

3. Synchronous PageRank

To make the computation of x practical for the problem sizes we are considering, it is necessary to employ an iterative method, e.g., executing until convergence

$$x(t+1) \leftarrow f(x(t)) \tag{3}$$

with $t = 0,1,\ldots$ for a suitable operator, f and some initial vector $x(0)$. The vector $x(t)$ denotes the approximation to x obtained after t iterations. The above process needs to be mapped on a specific execution environment, corresponding to a computational model that typically preserves the semantics of the mathematical model in (3). The environment constitutes a virtual machine for the computation and is largely characterized by the types of units of execution (UE) (e.g., processes, threads) and communication mechanisms (e.g., shared memory, message passing) it readily supports, especially in hardware. Execution and communication entities are ultimately hosted by actual machines typically attached to nets (e.g., clusters) and internets (e.g., the Internet).

In the single UE case the aforementioned mapping on the execution environment is straightforward. For multiple UEs, however, this requires care: In the shared memory case, a semantics preserving mapping must involve synchronized access to shared memory cells between cooperating UEs, protected by locks, whereas in the message passing case, this synchronization is achieved through a barrier mechanism implemented atop collective blocking communication. For the PageRank computation, we can easily turn (1) into the following simple iteration:

$$x(t+1) = G\,x(t), \quad x(0) \text{ given.} \tag{4}$$

That is, $f(\cdot)$ amounts to a matrix-vector multiplication. This is the well-known power method for finding the eigenvector of G corresponding to the eigenvalue of largest magnitude [26], except that no per-step normalization needs to be performed. The normalization is not needed since a stochastic matrix such as G does not alter $\|x(t)\|_1$; and thus no danger of overflow or underflow is present here.

Single UE implementations of (4) with an emphasis on convergence acceleration, support for personalization through different teleportation vectors and utilization of naturally occurring block structure in the adjacency matrix A can be found in [17–19]. For multiple UEs, message passing computation of PageRank using the formulation (2) was presented in [16].

4. Asynchronous PageRank

Unfortunately, the necessary per-step synchronization of the synchronous algorithm described above grows into a significant overhead, especially as it is governed by the rate of the slowest UE and the costs of lock or barrier management. One radical transformation to harness this problem is to reduce the requirement for synchronization, e.g., by using non-blocking access to shared memory cells or network buffers. A central theme of our work is to investigate the effect of this transformation on the convergence, speed and overall effectiveness of the computations.

For an environment with p UEs, denote by $x_{\{i\}}$ the set of indices assigned to i^{th} UE during the iterative computation, T^i the set of times at which $x_{\{i\}}$ is updated (i.e., i^{th} UE finishes its computation) and $\tau^i_j(t)$ the time when the fragment $x_{\{j\}}$, which is available at time t in the i^{th} UE, was actually produced at its respective j^{th} UE. Then for $t \in T^i$, the i^{th} UE updates

$$x_{\{i\}}(t+1) \leftarrow f_i(x_{\{1\}}(\tau^i_1(t)),\ldots,x_{\{p\}}(\tau^i_p(t))), \tag{5}$$

while $x_{\{i\}}(t+1) = x_{\{i\}}(t)$ at other times. Delays due to omission of synchronization phases are expressed as differences $t - \tau^i_j(t) \geq 0$. The relation (5) is the asynchronous analog of (3) where f_i expresses the distributed operator component executing at the i^{th} UE. Obviously the form of f_i is independent of the asynchronism introduced. It thus follows that the normalization-free power method for PageRank computation at the i^{th} UE reads

$$x_{\{i\}}(t+1) = G_i\,[x^\top_{\{1\}}(\tau^i_1(t)),\ldots,x^T_{\{p\}}(\tau^i_p(t))]^\top \tag{6}$$

for $t \in T^i$, and $x_{\{i\}}(t+1) = x_{\{i\}}(t)$ at other times, where G_i is a set of rows of the Google matrix G indexed by $\{i\}$. Alternatively, while the synchronous, linear system equation approach would lead to an iterative scheme of the form $x(t+1) = R\,x(t) + b$ which can be seen to be identical to (4), its asynchronous formulation would lead to another, slightly different computational kernel, namely

$$x_{\{i\}}(t+1) = R_i\,[x^\top_{\{1\}}(\tau^i_1(t)),\ldots,x^T_{\{p\}}(\tau^i_p(t))]^\top + b_i \tag{7}$$

for $t \in T^i$, and $x_{\{i\}}(t+1) = x_{\{i\}}(t)$ at other times, at the i^{th} UE. Here R_i is a set of rows of the relaxed stochastic matrix R indexed by $\{i\}$, and b_i is the corresponding set of elements of vector b.

Also of interest are P2P computations of PageRank [12,23,25,29] These fall into the multiple UEs, message passing category and are asynchronous in nature. An important novelty in these studies is the dynamically generated link information through a notification protocol proposed to be integrated with the host Web servers.

The lack of synchronization annuls the semantics of the original mathematical algorithm. Therefore, it becomes necessary to discuss the convergence properties of the asynchronous scheme (5). We discuss this and related issues in the remainder of this section.

4.1. Convergence

Convergence of asynchronous iterative algorithms is usually established through constructing a sequence of nested boxed sets in the spirit of the following theorem [9]:

Theorem 1 *Let* $\{X(k)\} : \ldots \subset X(k+1) \subset X(k) \subset \ldots \subset X$, *with the following two conditions.*
Synchronous Convergence Condition: For all $k = 1,\ldots, x \in X(k)$, $f(x) \in X(k+1)$, *and for* $\{y^k\}, y^k \in X(k)$: *the limit points of* $\{y^k\}$ *are fixed points of* f.
Box Condition: For all $k = 1,\ldots, X(k) = X_1(k) \times \ldots \times X_p(k)$.
Then if $x(0) \in X(0)$, *the limit points of* $\{x(t)\}$ *are fixed points of* f, *where* $\{x(t)\}$ *are given by (5).*

Process (6) involves a nonnegative matrix of unit spectral radius; it is proved in [21] that the corresponding asynchronous iteration converges to the true solution within a multiplicative factor that can easily be factored out in the end by renormalization. A discussion on the misconception by some authors that for a nonnegative matrix B, spectral radius $\rho(B) < 1$ is a necessary condition for convergence of an asynchronous normalization-free power method can be found in [27]. On the other hand, process (7) involves a matrix R with $\rho(R) < 1$. Asynchronous iterations with such matrices are well known to converge to the true solution [9].

computing UE	**monitor UE**

if(checkConvergence())
  if(not converged)
  converged = true
  pc++
  if(pc = pcMax)
  send(CONVERGE, monitor)
  recv(STOP, monitor)
else
  if(converged)
  converged = false
  send(DIVERGE, monitor)
  pc = 0
	recv(CONVERGE | DIVERGE, all)
	if(checkConvergence())
	  if(not converged)
	  converged = true
	  pc++
	  if(pc = pcMax)
	  send(STOP, all)
	else
	  if(converged)
	  converged = false
	  pc = 0

Figure 1. `pc`: persistence counter, `pcMax`: its max value; reaching it triggers `CONVERGE/STOP` messages. They can have different values in monitor, computing UEs; all: all computing UEs

4.2. Termination Detection

The termination of asynchronous iterative algorithms is a non-trivial matter since local convergence at an UE does not automatically ensure global convergence. Even in the extreme case when all UEs have locally converged, one can devise scenarios where messages not yet delivered could destroy local convergence.

Both centralized and distributed protocols for termination detection can be found in the literature, [8,24]. In a centralized approach, a special UE acts as a monitor of the convergence process of other computing UEs; it keeps a log of the convergence status and issues `STOP` messages to all computing UEs when all of them have signaled their local convergence. In fact, computing UEs can issue either `CONVERGE` (when achieving local convergence) or `DIVERGE` (when exiting such a state) messages to the monitor UE. Distributed protocols for global convergence detection (see, e.g., [28]) are flexible but rather complex to implement. They typically assume a specific underlying communication topology. For example in [6] a leader election protocol is used, which in turn assumes a tree topology.

Our draft version of a practical centralized protocol, in part inspired by [7], is presented in Figure 1. It enforces *persistence* of convergence both at the computing UEs (for issuing a `CONVERGE` message) and at the monitor UE (for issuing a `STOP` message). Persistence is introduced to provide time for pending -and perhaps divergence causing- messages to be actually delivered.

4.3. Implementation

We focus on multiple UEs message passing environments, which is the case for our experiments. In that case, we need non-blocking communication primitives. These are actually implemented either by using multithreading (e.g., one thread per communication channel) or by multiplexing such channels and probing from within a single thread (through `select()` type mechanisms) for new data. Since multiple messages might have been received in the meantime, messages should be kept in queues organized under a common discipline.

5. Numerical Experiments

5.1. Application Structure

Our application consists of scripts steering `Java` classes. These scripts are written in `Jython` [2], which is an implementation of the `Python` [4] programming language in `Java` [1]. Such a mixed-language approach facilitates writing portable, interactive, easily extensible and flexible systems; after all, performance critical operations can always be isolated into compiled `.class` code.

Scripts build and use objects. `Configuration` objects can load/store parameters from/to configuration files - accessible from all other objects, partition and distribute matrix or vector data and optionally send code or launch processes over the cluster nodes. `Computation` objects perform computations and exchange information related to convergence status with `Monitor` objects implementing the termination detection protocol; cf. Figure 1. Communications are established through `Communication` objects which set up suitable communicators upon their instantiation; these communicators expose communication primitives to be invoked at each step.

We use multithreading in order to implement non-blocking communications. An asynchronous `send()` or `recv()` is just its blocking counterpart wrapped in a thread object and submitted to a thread pool endowed with a suitable task-handling strategy. Data are imported/exported through read/write channels with locks synchronizing those concurrently executing threads which happen to be managing messages with identical source and target IDs. Access to thread pool queues and pending communication-task-handles is provided so that a customized thread-management policy can be applied. At startup, a single file containing computation parameters should be available. This file is used by a `Configuration` object for the generation of node-specific configuration files and a script for distributing these files (optionally with other data or updated source code files) to the cluster nodes and initiating the computation. An option for automatic report generation is also provided.

5.2. Numerical Results

We used a Beowulf cluster of Pentium-class machines at 900 MHz, with 256 MB RAM each, running Linux, version 2.4 and connected to a 10 Mbps Ethernet LAN. We used Java 5.0, Jython 2.1 and *Matrix Toolkits for Java* [3] for composing our scripts and classes, all freely available on the Web. We report on some of the results of this ongoing work. The transition matrix used in the experiments is the Stanford-Web matrix [5], generated from an actual web-crawl. It contains connectivity info for $281,903$ pages ($2,312,497$ non-zero elements, 172 dangling nodes). We used the computational kernel (6) with a local convergence threshold of 10^{-6}. Note that in each case, blocks of consecutive $[n/p]$ rows were distributed among computing machines. Termination detection used `pcMax = 1` on both monitor and computing UEs. Configurations with 2, 4, and 6 machines were tested for both synchronous and asynchronous computations. Results in Table 1 are encouraging. On the other hand, it is fair to note that they correspond to reaching local convergence threshold. Assembling vector fragments resulting from asynchronous computations at monitor UE and then checking global

	Synchronous		**Asynchronous**		
procs	*iters*	*t* **(sec)**	$[iters_{min}, iters_{max}]$	$[t_{min}, t_{max}]$ **(sec)**	$\langle speedUp \rangle$
2	44	179.2	[68, 69]	[86.3, 94.5]	1.98
4	44	331.4	[82, 111]	[139.2, 153.1]	2.27
6	44	402.8	129, 148]	[141.7, 160.6]	2.66

Table 1
Numerical results: For the asynchronous case iteration ranges, computation time ranges are given ([max, min] values) since local convergence threshold is not 'simultaneously' reached at all nodes. A column with the average speedup offered by asynchronous computation over synchronous one is given (averages are over extreme values in the asynchronous case).

convergence reveals that a threshold of the order of 5×10^{-5} has actually been reached. Preliminary results of timing with respect to reaching a common global threshold (instead of a local one) reveals a modest speedup of asynchronous vs. synchronous computation in the $10-20\%$ range. Responsibility for the degradation of performance when increasing the number of UE's appears to lie with the overall large communication-to-computation ratio of the current algorithm. Observe, however, that what is important are not the accurate values of the PageRank vector components, but their relative ranking. Therefore, an issue in our present investigations is the effect of a more relaxed global threshold criterion on the computed page ranks.

Asynchronous iterative algorithms also seem to naturally adapt to heavy communication demands in a computation; current `send()/receive()` threads can block but computation thread is free to advance to next step iteration. On the contrary, in synchronous mode, no option exists except for blocking all threads (even the computation one), until data emitted from all nodes actually reach their destinations and synchronization completes, no matter whether the supporting network's characteristics suffice. In this case asynchronous computation can exhibit a low message import ratio (always with respect to iteration count which is obviously increased relative to synchronous setting); see Table 2.

	Sender				
Receiver	*id = 0*	*id = 1*	*id = 2*	*id = 3*	**Completed Imports (%)**
id = 0	109	46	23	26	29
id = 1	40	107	22	27	28
id = 2	35	37	111	66	41
id = 3	27	30	54	82	45

Table 2
Completed imports for the 4 computing UEs, asynchronous case. Rows contain the number of different vector fragments actually received during the computation from peers with respective IDs. Diagonal numerical entries contain the total number of locally computed and thus locally used vector fragments. *Completed Imports* column contains percentage averages of imports actually completed (should all be 100% for the synchronous case).

6. Conclusions and Future Work

The major performance bottleneck in our experiments to date is due to the large volume of data and the frequency that it is being produced. The latter is caused by the small computation time per-iteration (sparse matrix-vector multiplication). Note also that the communication pattern is an all-to-all scheme at each step; all these factors conspire to surpass the available network bandwidth and thus build memory consuming buffers of pending messages at the sending ends.

In the case of asynchronous iterations, data is being produced at a rate that is even higher than in the synchronous case, because part of the time gained from eliminating the synchronization phases is actually used for the production of extra messages; these (favorably) advance local iteration counters but they could also (unfortunately) overload the network; we guard against this misfortune by cancelling `send()/recv()` threads not having completed within a time window. The following is thus a hardly surprising conclusion from our experiments: Asynchronous iterative algorithms make up an alternative computation methodology in distributed environments. However this is not a black-box methodology and is most effectively utilized by iterative methods with heavy computational component and light communication. A frequent, all-to-all, fat message passing can saturate network infrastructure capacity, even in modest but dedicated cluster environments; heterogeneous environments like the Grid would be even more sensitive to such message passing scenarios. We would thus like to avoid the use of all-to-all communication schemes; after all the flexibility of asynchronous iterations gives us a choice on the targets of produced messages. Furthermore, it is advisable to employ an adaptive communication scheme; if message sending/receiving tasks fail to complete within a number of local iterations, reduce the rate of message exchanges with this not well 'responding' node.

In our ongoing work, we explore adaptive schemes for the asynchronous computation of PageRank. We also experiment with `select()` based implementations of asynchronism in order to amortize thread management costs. Since trees are naturally occurring internetwork topologies we also plan to study the performance of moving a clique-based (i.e., all-to-all) synchronous iterative method to an asynchronous, tree-based counterpart. We are also considering the use of suitable permutations (cf. [11]) as well as larger data sets.

Acknowledgments

The work of the first two authors was partially supported by a Hellenic Pythagoras-EPEAEK-II research grant. The work of the third author was partially supported by the US NSF grant CCF-0514489. The authors would also like to thank Professor G. Kallos for providing access to the computational facilities of the Physics Department, University of Athens.

References

[1] Java language website. `http://java.sun.com`.

[2] Jython website. `http://www.jython.org`.

[3] Matrix Toolkits for Java website. `http://www.math.uib.no/ bjornoh/mtj/`.

[4] Python language website. `http://www.python.org`.

[5] Stanford Web Matrix. `http://nlp.stanford.edu/ sdkamvar/data/stanford-web.tar.gz`.

[6] J.M. Bahi, S. Contassot-Vivier, R. Couturier, and F. Vernier. A Decentralized Convergence Detection Algorithm for Asynchronous Parallel Iterative Algorithms. *IEEE Trans. Parallel Distrib. Syst.*, 16:4–13, 2005.

[7] J.M. Bahi, S. Domas, and K. Mazouzi. Jace: A Java Environment for Distributed Asynchronous Iterative

Computations. In *EUROMICRO-PDP'04*, pages 350–357. IEEE, 2004.

[8] D.El Baz. A method of terminating asynchronous iterative algorithms on message passing systems. In *Parallel Algorithms and Applications*, 9:153–158, 1996.

[9] D.P. Bertsekas and J.N. Tsitsiklis. *Parallel and Distributed Computation*. Prentice Hall, Englewood Cliffs, NJ, 1989.

[10] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A.Tomkins, and J. Wiener. Graph structure in the web: experiments and models. In *9th Int'l. WWW Conf.*, 2000.

[11] H. Choi and D.B. Szyld. Application of threshold partitioning of sparse matrices to Markov chains. In *IPDS'96*, pages 158–165. IEEE, 1996.

[12] D. de Jager. PageRank: Three Distributed Algorithms. Master's thesis, Imperial College of Science, Technology and Medicine, London, Sept. 2004.

[13] G.M. Del Corso, A. Gulli, and F. Romani. Fast PageRank computation via a sparse linear system. In *Lecture Notes in Computer Science, Vol. 3243*, pages 118–130. 2004.

[14] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann - Elsevier, San Francisco, 2004.

[15] A. Frommer and D.B. Szyld. On asynchronous iterations. *J. Comput. Appl. Math.*, 123:201–216, 2000.

[16] D. Gleich, L. Zhukov, and P. Berkhin. Fast Parallel PageRank: A Linear System Approach. Technical report, Yahoo! Inc., 2004.

[17] T.H. Haveliwala, S.D. Kamvar, and G. Jeh. An Analytical Comparison of Approaches to Personalizing Pagerank. Technical report, Stanford Univ., July 2003.

[18] S.D. Kamvar, T.H. Haveliwala, C. D. Manning, and G.H. Golub. Exploiting the Block Structure of the Web for Computing PageRank. Technical report, Stanford Univ., March 2003.

[19] S.D. Kamvar, T.H. Haveliwala, C. D. Manning, and G.H. Golub. Extrapolation Methods for Accelerating PageRank Computations. In *Proc. 12th Int'l. WWW Conf.*, May 2003.

[20] A.N. Langville and C.D. Meyer. A Survey of Eigenvector Methods for Web Information Retrieval. *SIAM Rev.*, 47:135–161, 2005.

[21] B. Lubachevsky and D. Mitra. A Chaotic, Asynhronous Algorithm for Computing the Fixed Point of a Nonnegative Matrix of Unit Spectral Radius. *J. ACM*, 33:130–150, Jan. 1986.

[22] L. Page, S. Brin, R. Montwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford Univ., 1998.

[23] K. Sankaralingam, S. Sethumadhavan, and J. C. Browne. Distributed Pagerank for P2P Systems. In *12th Int'l. Symposium on High Performance Distributed Computing*, 2003.

[24] S.A. Savari and D.P. Bertsekas. Finite termination of asynchronous iterative algorithms. *Parallel Computing*, 22(1):39–56, 1996.

[25] S.-M. Shi, J. Yu, G. Yang, and D. Wang. Distributed Page Ranking in Structured P2P Networks. In *ICPP'03*. IEEE, 2003.

[26] W.J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton Univ. Press, 1994.

[27] D.B. Szyld. The mystery of asynchronous iterations convergence when the spectral radius is one. Technical Report 98-102, Department of Mathematics, Temple Univ., Philadelphia, Oct. 1998.

[28] A.S. Tanenbaum and M. van Steen. *Distributed Systems, Principles and Paradigms*. Prentice-Hall, Upper Saddle River, NJ, 2002.

[29] L. Tsimonou. Distributed PageRank: Comparisons between a Simulation and a Peer-to-Peer Implementation of the Algorithm. Master's thesis, Imperial College of Science, Technology and Medicine, London, Sept. 2004.