Fast Parallel PageRank: A Linear System Approach

David Gleich*
Stanford University, ICME
Stanford, CA 94305
dgleich@stanford.edu

Leonid Zhukov
Yahoo!
701 First Ave
Sunnyvale, CA 94089
zhukov@yahoo-inc.com

Pavel Berkhin
Yahoo!
701 First Ave
Sunnyvale, CA 94089
pberkhin@yahoo-inc.com

ABSTRACT

In this paper we investigate the convergence of iterative stationary and Krylov subspace methods for the PageRank linear system, including the convergence dependency on teleportation. We demonstrate that linear system iterations converge faster than the simple power method and are less sensitive to the changes in teleportation.

In order to perform this study we developed a framework for parallel PageRank computing. We describe the details of the parallel implementation and provide experimental results obtained on a 70-node Beowulf cluster.

Categories and Subject Descriptors

G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra; D.2 [**Software**]: Software Engineering; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms

Algorithms, Performance, Experimentation

Keywords

PageRank, Eigenvalues, Linear Systems, Parallel Computing

1. INTRODUCTION

The PageRank algorithm, a method for computing the relative rank of web pages based on the Web link structure, was introduced in [25, 8] and has been widely used since then. PageRank computations are a key component of modern Web search ranking systems. For a general review of PageRank computing see [22, 7].

Until recently, the PageRank vector was primarily used to calculate a global importance score for each page on the web. These scores were recomputed for each new Web graph crawl. Recently, significant attention has been given to *topic-specific* and *personalized* PageRanks [14, 16]. In both cases one has to compute multiple PageRanks corresponding to various teleportation vectors for different topics or user preferences.

*Work performed while at Yahoo!

PageRank is also becoming a useful tool applied in many Web search technologies and beyond, for example, spam detection [13], crawler configuration [10], or trust networks [20]. In this setting many PageRanks corresponding to different modifications – such as graphs with a different level of granularity (HostRank) or different link weight assignments (internal, external, etc.) – have to be computed. For each technology, the critical computation is the PageRank-like vector of interest. Thus, methods to accelerate and parallelize these computations are important. Various methods to accelerate the simple power iterations process have already been developed, including an *extrapolation* method [19], a *block-structure* method [18], and an *adaptive* method [17].

Traditionally, PageRank has been computed as the principle eigenvector of a Markov chain probability transition matrix. In this paper we consider the PageRank linear system formulation and its iterative solution methods. An efficient solution of a linear system strongly depends on the proper choice of iterative methods and, for high performance solvers, the computation architecture as well. There is no best overall iterative method; one of the goals of this paper is to investigate the best method for the particular class of problems arising from PageRank computations on parallel architectures.

It is well known that the random teleportation used in PageRank strongly affects the convergence of power iterations [15]. It has also been shown that high teleportation can help spam pages to accumulate PageRank [13], but a reduction in teleportation typically hampers the convergence of standard power methods. In this paper, we investigate how the convergence of the linear system is affected by a reduction in a degree of teleportation.

To perform multiple numerical experiments on real Web graph data, we developed a system that can compute results within minutes on Web graphs with one billion or more links. When constructing our system, we did not attempt to optimize each method and instead chose an approach that is amenable to working with multiple methods in a consistent manner while still easily adaptable to new methods. An example of this trade-off is that our system stores edge weights for each graph, even though many Web graphs do not use these weights.

The remainder of the paper is organized as follows. The next section, *PageRank Linear System*, contains an analytical derivation of a linear system version of PageRank, followed by discussion of numerical methods. We then provide details on our parallel implementation. The *Numerical*

.

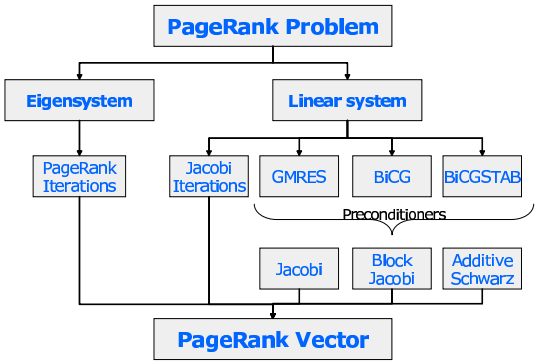


Figure 1: Chart of computational methods

Experiments section contains experimental results including timings of the iterative methods on various size graphs and an analysis of error metrics.

2. PAGERANK LINEAR SYSTEM

2.1 PageRank Formulation

Consider a Web graph adjacency matrix A with elements A_{ij} equal to 1 when there is a link $i \to j$ and equal to 0 otherwise. Here $i, j = 1 : n$ and n is a number of Web pages. For pages with a non-zero number of out-links $deg(i) > 0$, the rows of A can be normalized (made *row-stochastic*) by setting $P_{ij} = A_{ij}/deg(i)$. Assuming there are no dangling pages (vide infra) the PageRank x can be defined as a limiting solution of the iterative process

$$x_j^{(k+1)} = \sum_i P_{ij} x_i^{(k)} = \sum_{i \to j} x_i^{(k)}/deg(i). \qquad (1)$$

At each iterative step, the process distributes page authority weights equally along the out-links. The PageRank is defined as a *stationary* point of the transformation from $x^{(k)}$ to $x^{(k+1)}$ defined by (1). This transfer of authority corresponds to the Markov chain transition associated with the random surfer model when a random surfer on a page arbitrarily follows one of its out-links uniformly.

The Web contains many pages without out-links, called dangling nodes. Dangling pages present a problem for the mathematical PageRank formulation. A review of various approaches dealing with dangling pages can be found in [11].

One way to overcome this difficulty, is to slightly change the transition matrix P to a truly row-stochastic matrix

$$P' = P + d \cdot v^T, \qquad (2)$$

where $d_i = \delta_0^{deg(i)}$ is the dangling page indicator, and v is some probability distribution over pages. This model means that the random surfer jumps from a dangling page according to a distribution v. For this reason vector v is called a *teleportation* distribution. Originally uniform teleportation, $v_i = 1/n$, was used, but non-uniform teleportation received much attention after the advent of PageRank personalization [14, 16].

The classic simple power iterations (1) starting at an arbitrary initial guess converge to a principal eigenvector of P under two conditions specified in the *Perron-Frobenius* theorem. The first condition, matrix *irreducibility* or *strong connectivity* of a graph, is not satisfied for a Web graph, while the second condition, *aperiodicity*, is routinely fulfilled (after the first condition is satisfied). The easiest way to achieve strong connectivity is to add to every page, dangling or not, a small degree of teleportation

$$P'' = cP' + (1-c)ev^T, \quad e = (1, \ldots, 1). \qquad (3)$$

where c is a teleportation coefficient. In practice $0.85 \le c < 1$. After these modifications, matrix P'' is row-stochastic and irreducible, and therefore, simple power iterations

$$x^{(k+1)} = P''^T x^{(k)} \qquad (4)$$

for the eigen-system $P''^T x = x$ converge to its principal eigenvector.

Now, combining Eq.(2 - 4) we get

$$[cP^T + c(vd^T) + (1-c)(ve^T)]x = x. \qquad (5)$$

Noting that $(e^T x) = (x^T e) = ||x||_1 = ||x||$ (henceforth, all norms will be assumed to be 1-norms) we can derive a convenient identity

$$(d^T x) = ||x|| - ||P^T x||. \qquad (6)$$

Then, Eq. (5) can be written as a linear system

$$(I - cP^T)x = k\ v \qquad (7)$$

where $k = k(x) = ||x|| - c||P^T x|| = (1-c)||x|| + (d^T x)$.

A particular value of k only results in a rescaling of x, which has no effect on page ranking. The solution can always be normalized to be a probability distribution by $x/||x||$. This brings us to the point that equation (7) with a fixed k constitutes a linear system formulation of PageRank. In the future we simply set $k = 1 - c$. We also introduce the notations $A = I - cP^T$ and $b = k\ v = (1-c)v$, thus the linear system under consideration has the standard form $Ax = b$. The non-normalized solution to the linear system has an advantage of depending linearly on the teleportation distribution. This property is important for blending topical or otherwise personalized PageRanks. For this linear system, we study the performance of advanced iterative methods in a parallel environment.

Casting PageRank as a linear system was suggested by Arasu et al. [2] where Jacobi, Gauss-Seidel, and Successive Over-Relaxation iterative methods were considered. Numerical solutions for various Markov chain problems are also investigated in [26].

Method	IP	SAXPY	MV	Storage
PAGERANK		1	1	$M + 3v$
JACOBI		1	1	$M + 3v$
GMRES	$i+1$	$i+1$	1	$M + (i+5)v$
BiCG	2	5	2	$M + 10v$
BiCGSTAB	4	6	2	$M + 10v$

Table 1: Computational Requirements. Operations per iteration: IP counts inner products, SAXPY counts AXPY operations, MV counts matrix vector multiplications, and Storage counts the number of matrices and vectors required for the method.

2.2 Iterative Methods

The PageRank linear system matrix $A = I - cP^T$ is very large, sparse and non-symmetric. Solution of the linear system Eq. (7) by a direct method is not feasible due to the

Name	Nodes	Links	Storage Size
edu	2M	14M	176 MB
yahoo-r2	14M	266M	3.25 GB
uk	18.5M	300M	3.67 GB
yahoo-r3	60M	850M	10.4 GB
db	70M	1B	12.3 GB
av	1.4B	6.6B	80 GB

Table 2: Basic statistics for the data sets used in the experiments.

matrix size and computational recourses. Sparse LU factorization [23] can still be considered for smaller size problems (or subproblems), but it does create additional fill in. It is also notoriously hard to parallelize.

In this paper we concentrate on the use of iterative methods [12, 3, 6]. There are two main requirements for the iterative linear solver: i) it should work with nonsymmetric matrices and ii) it should be easily parallelizable. Thus, from the stationary methods we use Jacobi iterations (in particular, since the matrix is strongly diagonally dominant) and from the non-stationary methods we have chosen several Krylov subspace methods. Figure (1) presents a chart of the methods used in this paper.

Power iterations. Simple power iterations (4) are the classic and the most widely used process for finding PageRank. It has a convergence rate equal to c [15] (that is, the second eigenvalue of the matrix P''). Computationally efficient formulation of these iterations can be found in [25].

Jacobi iterations. The Jacobi process is the simplest stationary iterative method and is given by

$$x_{k+1} = cP^T \cdot x_k + b. \tag{8}$$

Jacobi iterations formally result in the geometric series $x = \sum_{k\geq 0}(cP^T)^k b$, when started at $x_0 = b$ and thus, also have the rate of convergence of at least c. It is indeed equal to c on graphs without dangling pages, in which case Jacobi and power iterations are the same.

Krylov subspace methods. We also consider a set of Krylov subspace methods to solve the linear system Eq. (7). These methods are based on certain minimization procedures and only use the matrix through matrix-vector multiplication. Detailed description of the algorithms are available in [6] and [4].

In this study we have chosen several Krylov methods satisfying our criteria:
- Generalize Minimum Residual (GMRES)
- Biconjugate Gradient (BiCG)
- Quasi-Minimal Residual (QMR)
- Conjugate Gradient Squared (CGS)
- Biconjugate Gradient Stabilized (BiCGSTAB)
- Chebyshev Iterations.

The convergence of Krylov methods can be improved by use of preconditioners. In this study we have used parallel Jacobi, Block Jacobi and Adaptive Schwarz preconditioners.

In all of the above methods we start with initial guess $x^{(0)} = v$. Note that the norm of $||x^{(k)}||$ is not preserved in the linear system iterations. If desired, one can normalize the solution to become a probability distribution by $x^{(k)}/||x^{(k)}||$.

The computational complexity and space requirements of the above methods are given in Table (1).

2.3 Parallel Implementation

We chose a parallel implementation of the PageRank algorithms to meet the scalability requirement. We wanted to be able to compute PageRank vectors for large graphs (one billion links or more) quickly to facilitate experiments with the PageRank equation. To that end, our goal was to keep the entire Web graph in memory on a distributed memory parallel computer while computing the PageRank vector. An alternate approach explored in [24] is to store a piece of the Web-graph on separate hard disks for each processor and iterate through these files as necessary.

Our parallel computer was a Beowulf cluster of RLX blades connected in a star topology with a gigabit ethernet. We had seven chassis composed of 10 dual processor Intel Xeon blades with 4 GB of memory each (140 processors, and 280 GB memory total). Each blade inside a chassis was connected to a gigabit switch and the seven chassis' were all connected to one switch.

The parallel PageRank codes use the Portable, Extensible Toolkit for Scientific Computation (PETSc) [4, 5] to implement basic linear algebra operations and basic iterative procedures on parallel sparse matrices. In particular, PETSc contains parallel implementations of many linear solvers, including GMRES, BiCGSTAB, and BiCG. While PETSc provided many of the operations necessary for working with sparse matrices, we still had to develop our own tools to load the Web graphs as parallel sparse matrices, distribute them among processors and perform load balancing.

PETSc stores a sparse matrix in parallel by dividing the rows of the matrix among the p processors. Thus, each processor only stores a submatrix of the original matrix. Further, PETSc stores vectors in parallel by only storing the rows of the vector corresponding to the matrix on that processor. That is, if we partition the n rows of a matrix A among p processors then processor i will store both the matrix rows and the corresponding vector elements. This distribution allows us to load graphs or matrices which are larger than any individual processor can handle, as well as to operate on vectors which are larger than any processor can handle. While this method allows us to work on large data, it may involve significant off-processor communication for matrix-vector operations. See [5] for a discussion of how PETSc implements these operations.

Ideally, the best way to divide a matrix A among p processors is to try to partition and permute A in order to balance work and minimize communication between the processors. This classical graph partitioning problem is NP-hard, and approximate graph partitioning schemes such as ParMeTiS [21] and Pjostle [27] do not work well on such large power-law data. Thus, we restricted ourselves to a simplified heuristic method to balance the work load between processors.

By default, PETSc balances the number of matrix rows on each processor by assigning each approximately n/p rows. For large Web graphs this scheme will result in a dramatic imbalance among the number of non-zero elements stored on each processor and often will result in at least one processor with more than 4 GB of data. To solve this problem we implemented a balancing scheme whereby we can choose how to balance the data among processors at run-time. We allow the user to specify two integer weights w_{rows} and w_{nnz} and try to balance the quantity $w_{\text{rows}}n_p + w_{\text{nnz}}nnz_p$ among processors, where n_p and nnz_p are the number of rows and

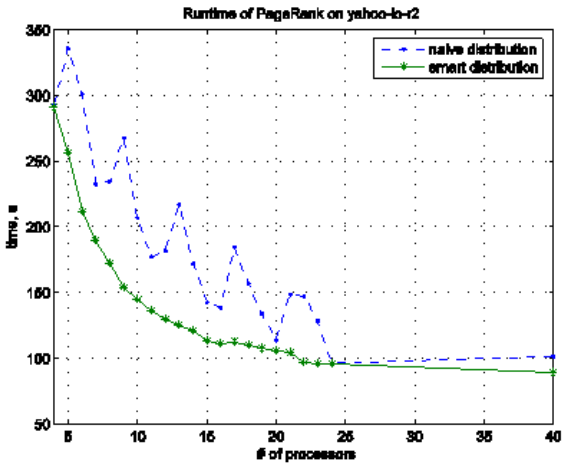


Figure 2: Load balancing experiment for "yahoo-r2" graph.

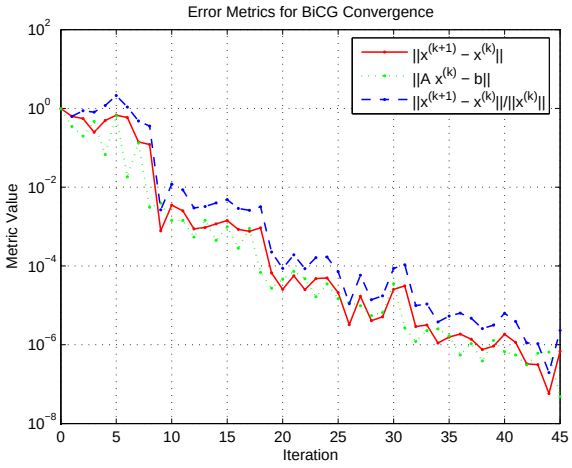


Figure 3: Tracking all error metrics during convergence on "db" graph for BiCG method.

non-zeros on processor p, respectively. To implement this approximate balancing scheme, we always store entire rows on one processor and keep adding rows to a processor and incrementing a counter until this counter exceeds a determined threshold $(w_{\text{rows}}n + w_{\text{nnz}}nnz)/p$. Typically, we used equal weighing between rows and non-zeros ($w_{\text{rows}} = 1, w_{\text{nnz}} = 1$). Due to the large number of low-degree nodes in our power-law dataset, this approximate scheme, in fact, gave us a good balance of work and memory among processors.

3. NUMERICAL EXPERIMENTS

3.1 Data

In our experiments we used six Web related directed graphs. The "av" graph is the Alta Vista 2003 crawl of the Web, the largest dataset used in experiments. We also constructed and used three subsets of "av" graph: the "edu" graph containing web pages from the .edu domain; "yahoo-r2" and "yahoo-r3" graphs containing pages reachable from yahoo.com by following two and three in- or out-links. We also used a Host graph "db" obtained from a Yahoo! crawl by aggregation (a sum of) links between the pages within the host. Finally, the "uk" graph contains only UK sites and was obtained from [1]. Basic statistics for these graphs is given in the Table (2).

3.2 Load Balancing Experiments

We have performed a set of load balancing experiments with the goal to find the best matrix partition and node distribution strategy that optimizes the algorithm performance. Figure (2) shows the computational (run) time as a function of number of processors for two load balancing schemes. The upper curve corresponds to an equipartitioned graph with n/p rows per processor, and the lower curve is obtained using our load balancing method.

It is clear that for the equipartitioned distribution, increasing the number of processors can lead to slower computations due to significant communications overhead. Web graphs contain clusters corresponding to blocks in the transition matrix. If the clusters are split naively over several processors, it leads to diminishing performance.

It is also seen from the bottom curve that when the communication and work load balance is approximately preserved, increasing the number of processors leads to a smaller computation time, but that speedup curve eventually saturates.

3.3 Equivalence of Metrics

When comparing the error among methods, we used absolute error $||x^{(k+1)} - x^{(k)}||$ for power iterations and normalized residual error $||Ax^{(k)} - b||/||x^{(k)}||$ for the linear systems. These are the equivalent metrics as the following analysis demonstrates. For a graph with no dangling nodes using power iterations,

$$||x^{(k+1)} - x^{(k)}|| = ||cP^T x^{(k)} - x^{(k)} + (1-c)v||, \quad (9)$$

and using a linear system,

$$||Ax^{(k)} - b|| = ||cP^T x^{(k)} - x^{(k)} + (1-c)v||. \quad (10)$$

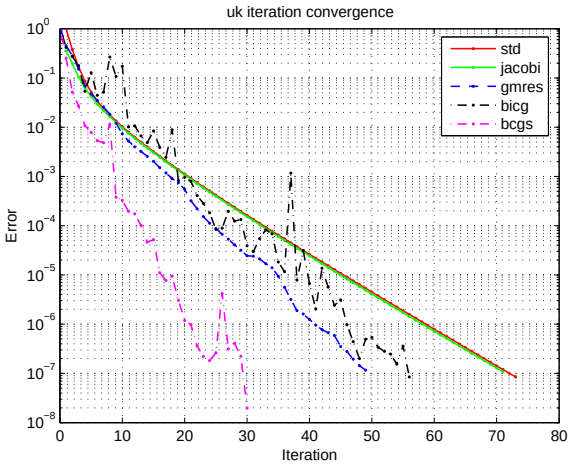
However, since we do not enforce $||x^{(k)}| = 1$ for the linear systems, we report $||Ax^{(k)} - b||/||x^{(k)}||$ instead.

In Figure (3) we show the value of the above metrics, plus relative error $||x^{(k+1)} - x^{(k)}||/||x^{(k)}||$, for a linear system solved with the BiCG solver on the "db" Host graph. The metrics follow the same behavior, with the residual slightly more stable for some methods. Thus residual was used to control and stop iterations when the desired tolerance was achieved. The other error metrics are almost linearly scaled because $||x^{(k)}||$ approaches a constant as k increases.

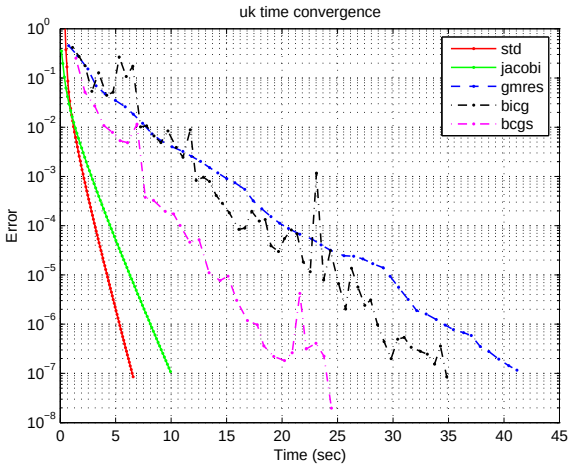
3.4 Convergence Experiments

We have performed multiple experiments on convergence of the iterative methods on a set of graphs. We have studied the rate of convergence, the number of iterations and total time taken by a method to converge to a given accuracy.

The detailed results for all tested methods and graphs are presented in Table (3). The quasi-minimum residual, conjugate gradient squared, and Chebyshev methods did not converge on our data and we do not report additional results on these methods. In the table, the first line for each graph denotes the number of iterations required for each method

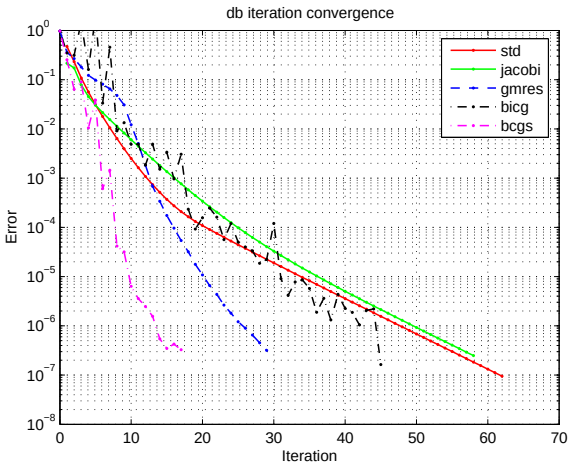


(a) Convergence Iterations

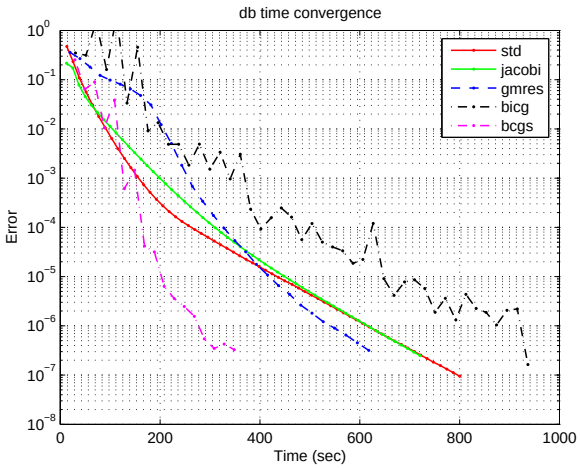


(b) Convergence Time

Figure 4: Convergence of iterative methods on the "uk" Web graph.



(a) Convergence Iterations



(b) Convergence Time

Figure 5: Convergence of iterative methods on the "db" Host graph.

Graph	PR	Jacobi	GMRES	BiCG	BCGS
edu	84	84	21†	44*	21*
20 procs	0.06 s	0.06 s	0.6 s	0.85 s	0.41 s
yahoo-r2	71	65	12	20	10
uk	73	71	22*	25*	11*
60 procs	0.08 s	0.14 s	0.8 s	0.78 s	1.05 s
yahoo-r3	76	75			
60 procs	2.4 s	2.2 s			
db	62	58	29	45	15*
60 procs	13 s	12 s	22 s	21 s	21 s
av	72	76			26
140 procs	30 s	30 s			60 s

Table 3: Timing Results. See discussion in text.

to converge to an absolute residual value of 10^{-7}. The second line shows the mean time per iteration at the given number of processors. For the Krylov subspace methods, no superscript means we used no preconditioner, whereas $*$ denotes a block Jacobi preconditioner and † denotes an additive Schwartz preconditioner. We only report results for the fastest (by time) method. All Krylov solvers failed on "yahoo-r3" due to memory overloads on only 60 processors.

The convergence behavior of algorithms on "uk" and "db" graphs is shown in Figures (4) and (5). In these figures, we plot absolute error $||x^{(k+1)} - x^{(k)}||$ for power iterations and normalized residual $||Ax^{(k)} - b||/||x^{(k)}||$ for the linear systems.

Our analysis of the results shows that

- Power and Jacobi methods have approximately the same rate and the most stable convergence pattern. This is an advantage of stationary methods that perform the same amount of work per any iteration.
- Convergence of Krylov methods strongly depends on the graph and is non-monotonic.
- Although the Krylov methods have the highest average convergence rate and fastest convergence by number of iterations, on some graphs, the actual run time can be longer than the run time for simple Power iterations.
- BiCGSTAB and GMRES have the highest rate of convergence and converge in the smallest number of iterations, with GMRES demonstrating more stable behavior.

3.5 Convergence and Teleportation

We have performed several experiments to learn the dependency of the convergence rate on the teleportation coefficient $c \geq 0.85$. The results are Figure (7). It shows that when $c = 0.99$, power iterations were not able to obtain desired accuracy within a large number of iterations. For the values of c where each algorithm converges, the deterioration of convergence rate for the Krylov subspace methods was much less pronounced than for simple power iterations. In other words they are less affected by decreasing the teleportation level.

3.6 Experiment with the "av" Web graph

In this section, we discuss the results of our parallel linear system PageRank on a full Web graph. We used a September 2003 crawl of the Web from Alta Vista with 1.4 billion

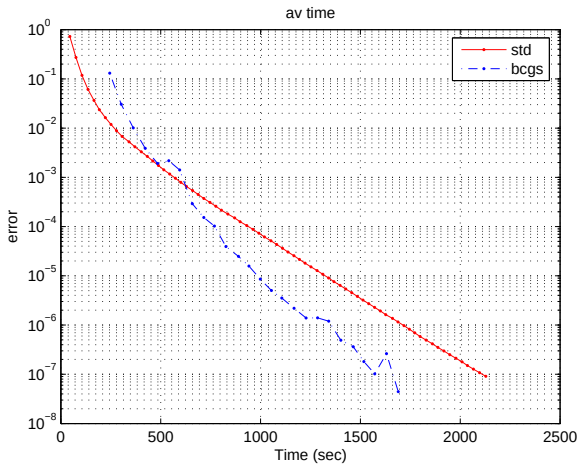


Figure 6: Convergence on the full "av" Web graph.

nodes and 6.6 billion edges. Other experiments with this data have been done by Broder et al. [9]. Our main results on this graph are presented in Figure (6) and Table (3). It presents performance of simple power iterations and the BiCGSTAB method. For the linear system we show the normalized residual error and absolute error for power iterations.

A few tweaks were required to allow this large graph to work with our implementation. First, equal balancing between nodes and edges is insufficient for such large graphs and overloads the 4 GB memory limit on some nodes. Thus, we attempted to estimate the optimal balance for the algorithm. From Table (1), the PageRank algorithm requires storing the matrix and 3 vectors. Since the matrix is stored in compressed-row format with double sized floating point values, PETSc requires at least $(12 \cdot nnz + 4n) + 3 \cdot (8n)$ bytes. Because there is some overhead in storing the parallel matrix, we typically estimate about $16nnz$ total storage for the matrix (i.e. 33% overhead). This yields a row-to-non-zero distribution of approximately 2 to 1. Thus, for simple power iterations we set $w_{\text{rows}} = 2$ and $w_{\text{nnz}} = 1$ for our dynamic matrix distribution. A similar analysis for the BiCGSTAB algorithm yields the values $w_{\text{rows}} = 5$ and $w_{\text{nnz}} = 1$.

It is reported in [9] that an efficient implementation of the serial PageRank algorithm took 12.5 hours on this graph using a quad-processor Alpha 667 MHz server. A similar serial implementation on a 800 MHz Itanium takes approximately 10 hours. Our implementation takes 35.5 minutes (2128 secs) for PageRank and 28.2 minutes (1690 secs) for BiCGSTAB on the full cluster of 140 processors (70 machines). These parallel run times do not include the time necessary to load the matrix into memory as repeated runs could be done with the in-memory matrix.

4. CONCLUSIONS

In this paper we have demonstrated that PageRank can be successfully computed using linear system iterative solvers. We have developed an efficient scalable parallel implementation and studied Jacobi and Krylov subspace iterative methods. Our numerical results show that GMRES and BiCGSTAB are overall the best choice of solution methods

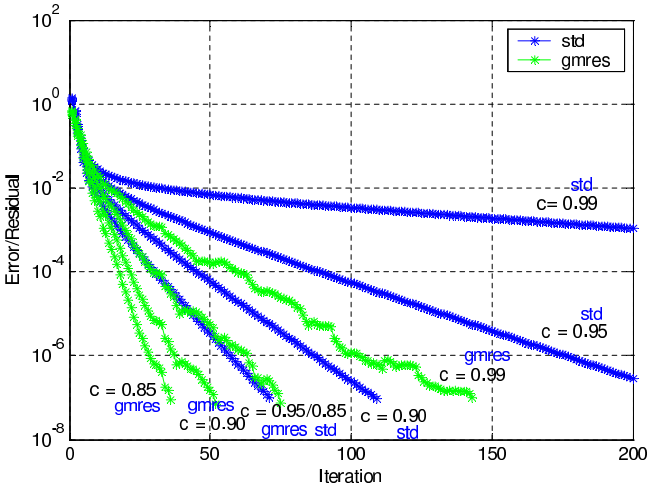


Figure 7: Convergence at high c.

for PageRank class of problems and, for most graphs, provide faster convergence than power iterations. We have also demonstrated that the linear system PageRank can converge for much larger values of the teleportation coefficient c than standard power iterations.

5. ACKNOWLEDGMENTS

We would like thank Farzin Maghoul for the Alta Vista web graph, Alexander Arsky for the Host graph, Jan Pedersen, Gary Flake, and Vivek Tawde for help and discussions and Yahoo! Research Labs for providing valuable resources and technical support.

6. REFERENCES

[1] Webgraph datasets. http://webgraph-data.dsi.unimi.it.

[2] A. Arasu, J. Novak, A. Tomkins, and J. Tomlin. PageRank computation and the structure of the web: Experiments and algorithms. In *WWW11*, 2002.

[3] O. Axelsson. *Iterative solution methods.* Cambridge University Press, New York, NY, USA, 1994.

[4] S. Balay, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 2.1.5, Argonne National Laboratory, 2004.

[5] S. Balay, V. Eijkhout, W. D. Gropp, L. C. McInnes, and B. F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997.

[6] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. V. der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition.* SIAM, Philadelphia, PA, 1994.

[7] P. Berkhin. A survey on pagerank computing. Technical report, Yahoo! Inc., 2004.

[8] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 33(3):107–117, 1998.

[9] A. Z. Broder, R. Lempel, F. Maghoul, and J. Pedersen. Efficient pagerank approximation via graph aggregation. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 484–485. ACM Press, 2004.

[10] J. Cho, H. García-Molina, and L. Page. Efficient crawling through URL ordering. *Computer Networks and ISDN Systems*, 30(1–7):161–172, 1998.

[11] N. Eiron, K. McCurley, and J. Tomlin. Ranking the web frontier. In *WWW13*, 2004.

[12] G. H. Golub and C. F. V. Loan. *Matrix computations (3rd ed.).* Johns Hopkins University Press, 1996.

[13] Z. Gyongyi, H. Garcia-Molina, and J. Pedersen. Combating web spam with trustrank. In *30th International Conference on Very Large Data Bases*, pages 576–587, 2004.

[14] T. Haveliwala. Topic-sensitive pagerank, 2002.

[15] T. Haveliwala and S. Kamvar. The second eigenvalue of the Google matrix. Technical report, Stanford University, California, 2003.

[16] G. Jeh and J. Widom. Scaling personalized web search. In *WWW12*, pages 271–279. ACM Press, 2003.

[17] S. Kamvar, T. Haveliwala, and G. Golub. Adaptive methods for the computation of pagerank, 2003.

[18] S. Kamvar, T. Haveliwala, C. Manning, and G. Golub. Exploiting the block structure of the web for computing pagerank, 2003.

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

[20] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the Twelfth International World Wide Web Conference*, 2003.

[21] G. Karypis and V. Kumar. A coarse-grain parallel formulation of multilevel k-way graph-partitioning algorithm. In *Proc. 8th SIAM Conference on Parallel Processing for Scientific Computing*, 1997.

[22] A. N. Langville and C. D. Meyer. Deeper inside pagerank. Technical report, NCSU Center for Res. Sci Comp., 2003.

[23] X. Li and J. Demmel. Superlu dist: A scalable distributed-memory sparse direct solver for unsymmetric linear systems, 2003.

[24] B. Manaskasemsak and A. Rungsawang. Parallel pagerank computation on a gigabit pc cluster. In *Proceedings of the 18th International Conference on Advanced Information Networking and Applications*, 2004.

[25] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[26] W. J. Stewart. Numerical methods for computing stationary distribution of finite irreducible Markov chains. In W. Grassmann, editor, *Advances in*

Computational Probability, chapter 4. Kluwer Academic Publishers, 1999.

[27] C. Walshaw, M. Cross, and M. G. Everett. Parallel dynamic graph partitioning for adaptive unstructured meshes. *Journal of Parallel and Distributed Computing*, 47(2):102–108, 1997.