Scalable Computing for Power Law Graphs

… experience with parallel PageRank

David Gleich
Institute of Computation and Mathematical Engineering
Stanford University

Leonid Zhukov
Yahoo! Inc.

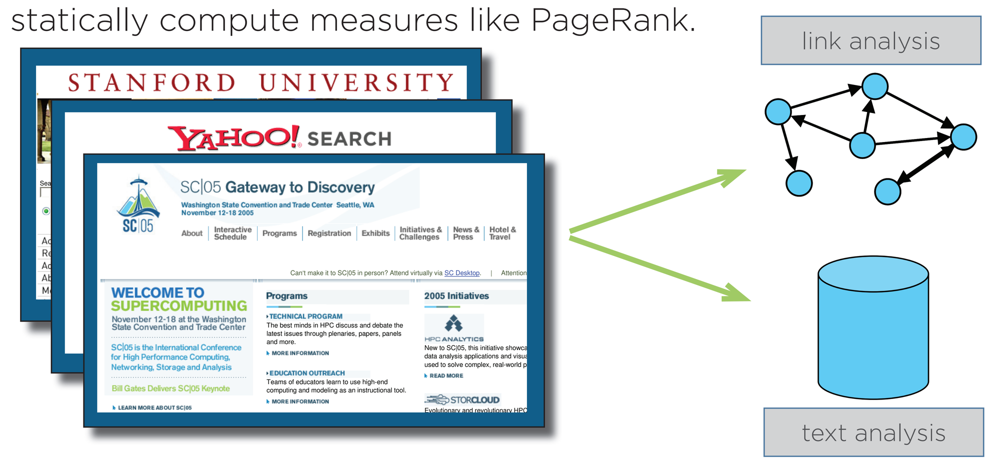
Abstract

We investigated the numerical and parallel performance of linear algebra algorithms when applied to power law data typical for information retrieval. In particular, we report on the behavior of parallel numerical algorithms applied to computing the PageRank vector for a 1.4 billion node and 6.6 billion edge directed web graph.

We also describe the details of our parallel implementation that was able to compute the PageRank vector for this web graph on a distributed memory cluster in under 6 minutes. The cluster had 120 blades with two 2.8 GHz Intel Xeon processors and 4 GB of memory each.

1. Websearch

Modern websearch engines consist of two stages: an indexing stage, and a query stage. At the indexing stage, a web-crawler traverses links between web pages and builds a text database and link database for all pages on the web. These two databases form the core of a search engine. We perform off-line analysis of the link database to statically compute measures like PageRank.



At search time, (1) an engine finds pages that contain the query word in the text database. Then, (2) it computes a query similarity score for each page and retrieves the PageRank score (as well as other features). Finally, it (3) sorts the pages and returns the results.

Page	Query Score	PageRank
1	0.7	0.4
2	0.8	0.1
2	0.1	0.5

2. PageRank Model

To rank all the pages on the web, PageRank models a random surfer browsing the web and uses the stationary distribution of the associated Markov chain as the ranking score. Assume we are given a webgraph adjacency matrix, A, a parameter, c, and a prior probability distribution over pages, v.

1. Construct the random walk matrix.

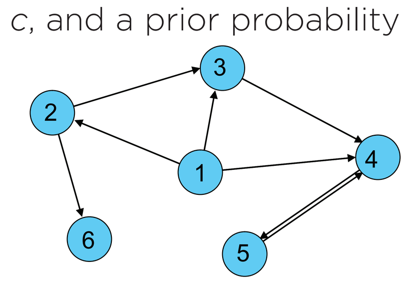
$$P = D^{-1}A$$

2. Add links from dangling pages.

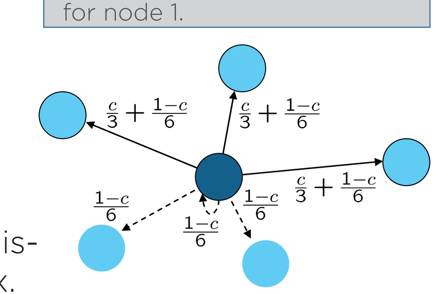
$$P' = P + dv^T$$

3. Add random moves.

$$P'' = cP' + (1-c)ev^T$$



above: the original webgraph.
below: transition probabilities for node 1.



The PageRank vector is the stationary distribution of the Markov transition matrix.

$$\lambda p = P''^T p$$

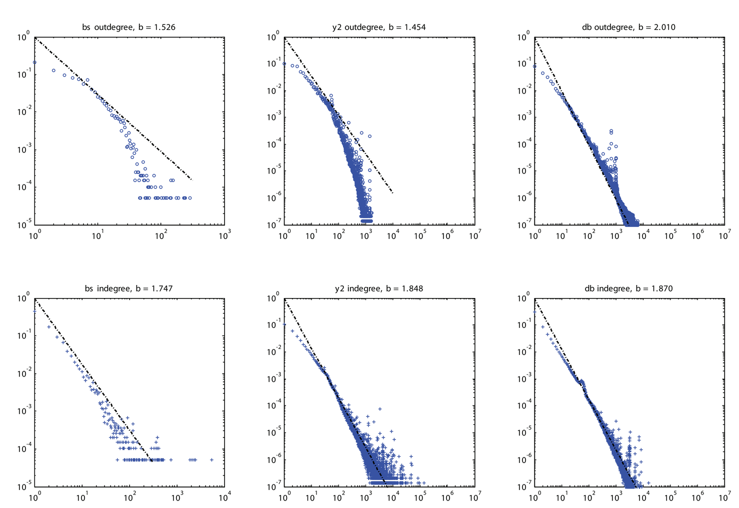
A unique stationary distribution exists because P'' is a strictly positive matrix and the Perron-Frobenius theorem applies.

3. Power Law Data

A power law graph is defined by the property that the number of vertices with degree k is proportional to

$$k^{-\beta}$$

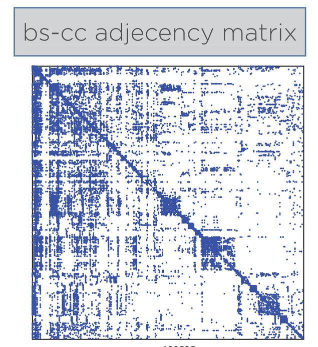
for power law exponent β. Our graphs have β between 1.4 and 2.0.



power law degree distribution

The implication of the power-law distribution is that many pages have very low degree and a few pages have extremely high degree. Futher, there is little structure in the associated adjacency matrix as the matrix-plot shows.

bs-cc adjacency matrix



Our datasets have a wide variety of sizes, but all display the power-law distribution.

Name	# Nodes	# Links	Storage
bs-cc	20 K	130 K	1.6 MB
edu	2 M	14 M	176 MB
yahoo-r2	14 M	266 M	3.25 GB
uk	18.5 M	300 M	3.67 GB
yahoo-r3	60 M	850 M	10.4 GB
db	70 M	1 B	12.3 GB
av	1.4 B	6.6 B	80 GB

datasets and sizes

4. Computation

PageRank can be converted from an eigensystem to a linear system by way of the following transformation. This transformation gives us a wide-range of computational possibilities.

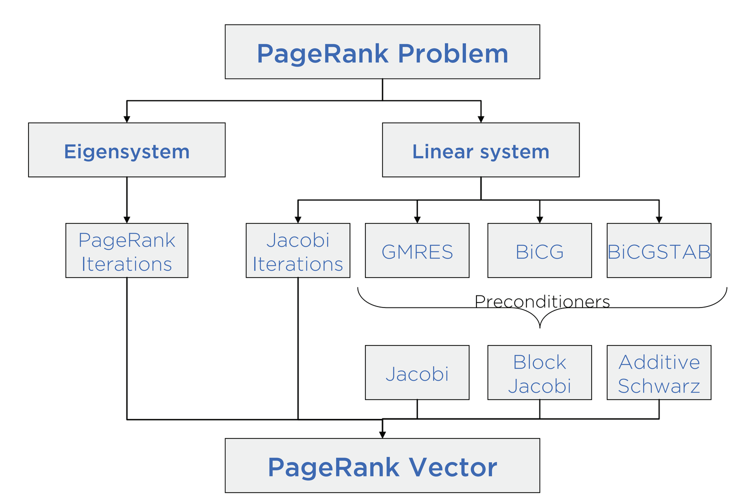
Eigensystem

$$P''^T p = \lambda p$$
$$\lambda = 1$$
$$P'' = cP + c(dv^T) + (1-c)(ev^T)$$

Linear system

$$(I - cP^T)x = kv$$
$$p = \frac{x}{||x||}$$
$$k = k(x) = ||x|| - c||P^T x||$$

The linear system is **large, sparse, and unsymmetric**. This limited our choice of linear solvers to parallel, unsymmetric solvers. From this subset of solvers, we examined Jacobi, GMRES, BiCG, and BiCGSTAB. (Other available solvers in PETSc did not converge.)



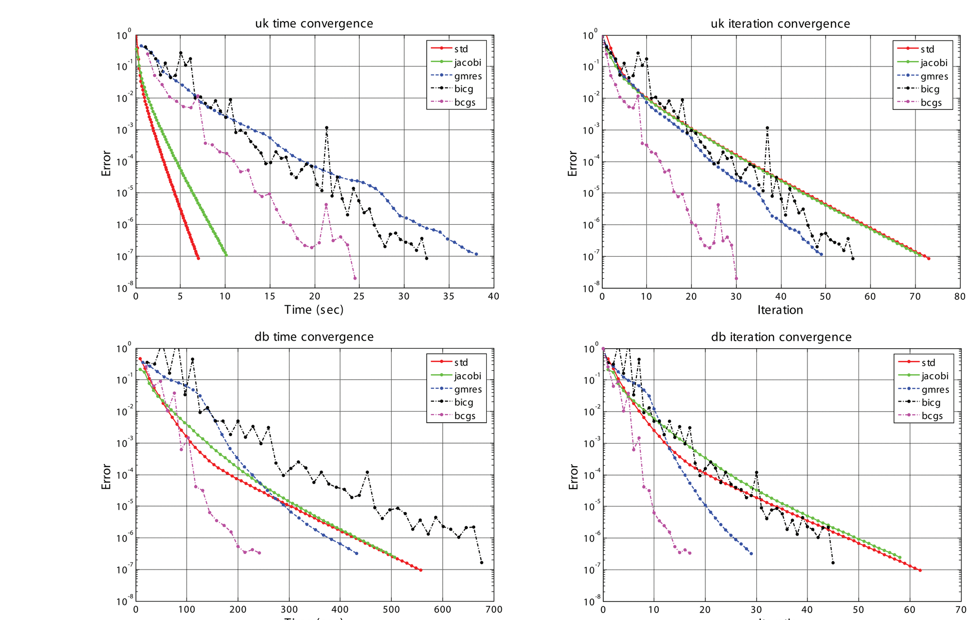
Our computational flowchart shows all the methods from our experiments. The following tables gives the computational costs.

Method	Inner Products	SAXPY	Matrix-Vector	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	$M + 10v$

computational cost

5. Results

Our goal was to compute the PageRank vector as quickly as possible, to a given tolerance. Each plot below shows convergence of all of the methods in terms of number of iterations and time. The graphs used were uk and db, which in our experience, show the behavioral extremes.



computational cost

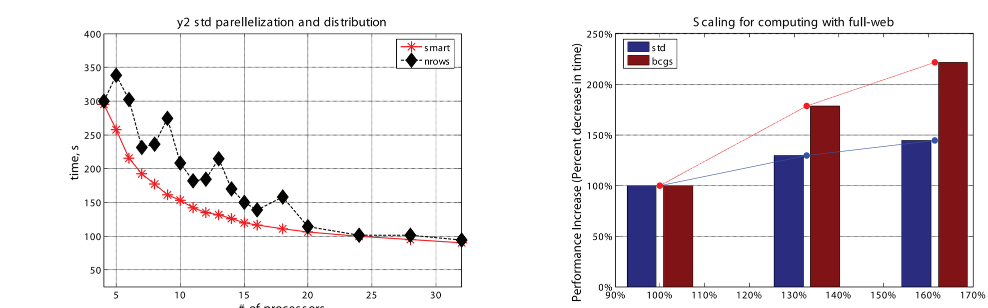
The next table shows the runtime for the each method.

Name	Size	Power	Jacobi	GMRES	BiCG	BCGS
edu 20 procs	2M 14M	84 0.09/7.5s	84 0.07/6.5s	21* 0.6/13.2s	44* 0.4/17.7s	21* 0.4/8.7s
yahoo-r2 20 procs	14M 266M	71 1.8/129s	65 1.9/126s	12* 16/194s	35* 8.6/300s	17* 9.9/168s
uk 60 procs	18.5M 300M	73 0.09/7s	71 0.1/10s	22* 0.8/17.6s	25* 0.8/19.4s	11* 1.0/10.8s
yahoo-r3 60 procs	60M 850M	76 1.6/119s	75 1.5/112s			
db 60 procs	70M 1B	62 9.0/557s	58 8.7/506s	29 15/432s	45 15/676s	15* 15/220s
av 140 procs	1.4B 6.6B	72 4.6/333s				26 15/391s

The size is the number of nodes (pages) and number of edges (links). Each entry is the number of iterations, average time per iteration, and total time. * denotes a preconditioner. The residual tolerance is 10-7.

6. Scaling

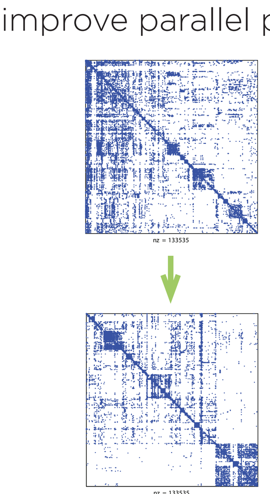
We observed that a trivial distribution of the matrix to processors (balancing only number of rows) displayed oscillatory behavior as we scaled the number of processors. This behavior is smoothed by our heuristic load-balancing distribution. Additionally, the more sophisticated linear solvers (i.e. the KSP methods) showed better scalability than simple power iterations.



distribution and scaling plots

7. Host Ordering

The webgraph has the property that pages within a particular host (e.g. icme.stanford.edu) are much more connected than pages between hosts. Because we have the URL for each page (which includes the host), we can exploit this property to reorder the matrix and improve parallel performance.



reorder matrix

speedup

8. Conclusions

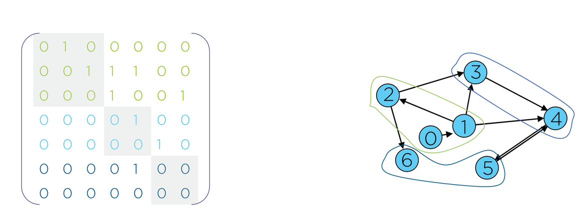
- The power iteration and Jacobi methods have approximately the same behavior on our graphs.
- The convergence of Krylov methods strongly depends on the graph and is non-monotonic.
- Krylov methods converge fastest by number of iterations, but the actual run time may be longer than simple power iterations.
- BiCGSTAB and GMRES converge in the smallest number of iterations. GMRES demonstrates more stable behavior.
- The BiCGSTAB algorithm scales better than power iterations due to the parallelism in the extra work performed.
- The best method to use is either power iterations or BiCGSTAB. The final choice of method is dependent on the time of a parallel matrix-vector multiply compared with the time of the extra work performed in the BiCGSTAB algorithm.

Parallel Distribution

We stored the webgraph in parallel using a sparse adjacency matrix representation. This corresponds to putting groups of nodes on each processor. We were unsuccessful in using existing graph partitioning tools and implemented a heuristic balancing technique. In our technique, we do not reorder the matrix and fill up a processor until:

$$w_{\text{rows}} n_p + w_{\text{nnz}} nnz_p > \frac{w_{\text{rows}} n + w_{\text{nnz}} nnz}{p}$$

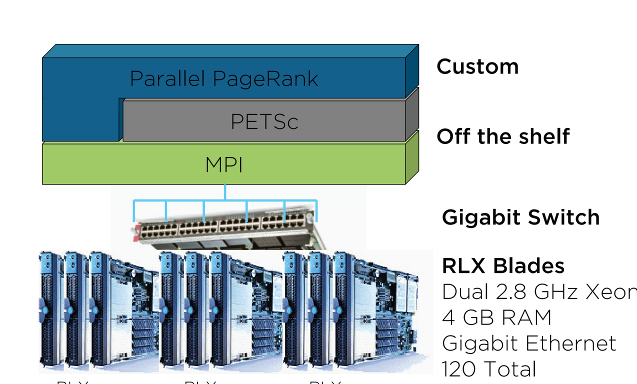
$$w_{\text{rows}} / w_{\text{rows}} = 1/1 \text{ or } 2/1 \text{ or } 4/1$$



parallel graph and adjacency matrix

Parallel System

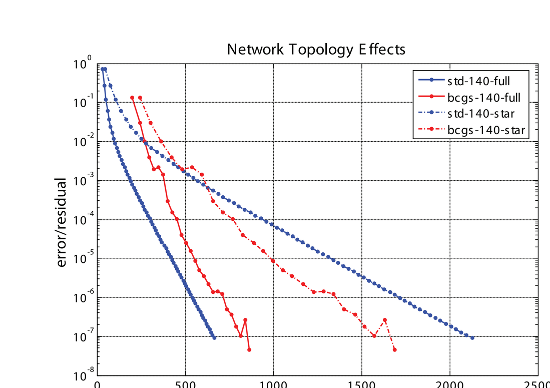
Our parallel computer was a cluster of 120 blades connected with a full Gigabit ethernet switch. We implemented our PageRank codes on top of PETSc and MPI. We used PETSc for the basic linear algebra operations and iterative methods on parallel sparse matrices.



our parallel system

Parallel Topology

We investigated two network topologies: a star topology with 10 blades on each switch and a fully connected topology with one master Gigabit ethernet switch. The different topologies changed the relative behavior of the algorithms.



topology speedup