Abstract

This work is a summary of Kang et. al.'s paper "PEGASUS: A Peta-Scale Graph Mining System". It presents GIM-V, a generalization of matrix-vector multiplication, a number of applications, and its implementation with map-reduce.

1 Generalized Iterative Matrix-Vector Multiplication

GIM-V (Generalized Iterative Matrix-Vector Multiplication) is a generalization of iterative matrix-vector multiplication with three variation points. Given a matrix M and a vector V, the result of $M \times_G V$ is a vector V', where vector elements are defined as follows.

$$V'_i = \mathit{assign}(V_i, \mathit{combineAll}\{x_j \mid 1 \leq j \leq n \wedge x_j = \mathit{combine2}(M_{i,j}, v_j)\})$$

This step is repeated iteratively until a convergence criteria is met. A number of applications can be implemented as specializations of GIM-V by defining functions *assign*, *combine2*, and *combineAll*. Figure 1 gives an overview of a number of interesting examples.

Application	M	$\mathit{combine2}(m, v)$	$\mathit{combineAll}(x_1, \ldots, x_n)$	$\mathit{assign}(v_{old}, v_{new})$
Matrix-Vector Multiply	M	$m \times v$	$\sum_{j=1}^{n} x_j$	v_{new}
PageRank	col.-norm. E^T	$c \times m \times v$	$\frac{1-c}{n} + \sum_{j=1}^{n} x_j$	v_{new}
Random Walk	col.-norm. A^T	$c \times m \times v$	$(1-c) \times \max\{\lvert i-k \rvert, 1\} + \sum_{j=1}^{n} x_j$	v_{new}
Diameter Estimation	A	$m \times v$	$\bigoplus_{j=1}^{n} x_j$	$v_{old} \oplus v_{new}$
Connected Components	A	$m \times v$	$\min_{j=1}^{n} x_j$	$\min\{v_{old}, v_{new}\}$

Figure 1: Applications of GIM-V

The *PageRank* vector P of n web pages is defined as the eigenvector equation $P = (cE^T + (1-c)U)P$, where c is a damping factor, E is a row-normalized adjacency matrix, and U is a matrix where every element is $1/n$. In GIM-V, the PageRank P is defined as an iterative application of $P' = M \times_G P$ with an initial vector $P = (1/n, \ldots 1/n)^T$ and M being a column-normalized matrix E^T.

In Random Walk with Restart (*RWR*), a proximity vector R_k from node k is defined as $R_k = cMR_k + (1-c)e_k$, where only the k-th element is set to 1 in e_k and c is a restart probability parameter. In GIM-V, RWR is defined as an iterative application of $R'_k = M \times_G R_k$ with M being a column-normalized transposed adjacency matrix.

The diameter of a graph can be estimated using HADI (*HAdoop based DIameter estimator*). For an iteration h, HADI maintains a bit vector B_i^h of already reached nodes per starting node i. For every bit vector B_i^h, HADI marks those nodes j as visited in B_i^{h+1}, for which $(k, j) \in E$ and $B_i^k[k] = 1$. In GIM-V, HADI is defined as an iterative application of $B^{h+1} = M \times_G B^h$ with an initial vector $B^0 = ((\top, \bot, \ldots, \bot), (\bot, \top, \ldots, \bot), \ldots, (\bot, \bot, \ldots, \top))$ (only visited node from i in iteration 0 is node i itself) and M being and adjacency matrix[1].

HCC is an algorithm for finding connected components maintaining a component ID C_i^h for every node i in iteration h. In iteration $h+1$, C_i^{h+1} is set to the minimum value of all component IDs of neighboring nodes. In GIM-V, HCC is defined as an iterative application of $C^{h+1} = M \times_G C^h$ with an initial vector $C^0 = (1, 2, \ldots, n)$ and M being an adjacency matrix.

2 Implementation of GIM-V

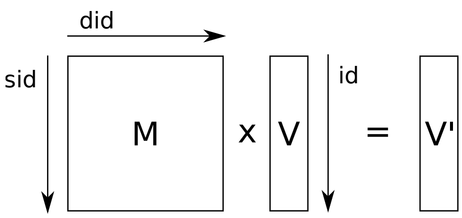

Figure 2: Dimensions of M and V

GIM-V can be implemented with a 2-phase map-reduce algorithm. Figure 2 illustrates matrix/vector dimensions and their identifiers, which are used in the following map/reduce descriptions.

- Phase 1 Mapper: The mapper receives key-value input pairs for M and V in the form of $(\mathit{sid}, (\mathit{did}, m))$ and (id, v), respectively. It outputs pairs $(\mathit{did}, (\mathit{sid}, m))$ and (id, v) for input originating from M and V, respectively.
- Phase 1 Reducer: The reducer outputs a pair $(\mathit{id}, (\text{"self"}, v))$ representing the old vector value and multiple pairs $(\mathit{id}, (\text{"others"}, \mathit{combine2}(m, v)))$ for partially-combined results.
- Phase 2 Mapper: The mapper outputs unmodified key-value pairs generated by Phase 1.
- Phase 2 Reducer: For an ID *id*, the reducer receives a value $(\text{"self"}, v))$ and multiple values $(\text{"other"}, c_i)$ for $i = 1 \ldots n$. It outputs the pair $(\mathit{id}, \mathit{assign}(v, \mathit{combineAll}(c_1, \ldots, c_n)))$.

[1] $A \oplus B$ denotes a bit-wise OR operation on vectors A and B.

3 Optimizations of GIM-V

This section gives an overview of a number of optimizations that can speed up the performance of GIM-V.

Matrix/Vector Blocking In *GIM-V BL*, the matrix M is divided in blocks of size $b \times b$ and the vector V is divided in blocks of length b. Matrix-Vector multiplication is performed on blocks instead of the entire matrix/vector, resulting in multiple partial results (block results) that must be combined in an additional step. M and V are encoded block-wise as a list of (sid, did, m) tuples and (id, v) tuples, respectively. Zero values are omitted. GIM-V BL is faster than GIM-V, because blocking reduces the number of elements to sort during the *shuffle* phase in map-reduce and because of compression opportunities due to smaller relative indices inside a block.

Clustered Edges *GIM-V CL* is an optimization for graph algorithms, where M respresents an adjacency matrix. GIM-V CL is GIM-V BL with a preprocessed matrix M'. M' represents a graph isomorphic to the one represented by M, i.e., it is essentially a reordering of node IDs. M' should be computed in such a way that the number of blocks consisting of only zeros is maximized. These blocks can then be omitted entirely.

Diagonal Block Iteration *GIM-V DI* is an optimization for HCC that reduces the number of required iterations and thus the bottleneck of disk I/O and shuffling. The basic idea is to multiply diagonal matrix blocks with corresponding vector blocks multiple times in a single iteration. The authors proved an upper bound of $2 \times \lceil \frac{m}{b} \rceil - 1$ for the number of iterations in GIM-V DI for a chain graph of length m with block size b.

4 Performance and Results

GIM-V as described in the original paper has a run-time complexity of $\mathcal{O}(\frac{|V|+|E|}{M} \times \log \frac{|V|+|E|}{M})$, where M is the number of machines. Its space complexity is $\mathcal{O}(|V| + |E|)$.

Benchmark Results Benchmarks run on Yahoo!'s M45 Hadoop cluster indicate GIM-B CL with blocking has the best performance for PageRank with a synthetic Kronecker graph. The relative performance of each optimization compared to the basic version (GIM-V BASE) decreases when adding more machines due to overhead that cannot be optimized (e.g., JVM load time, I/O, network communication). Futhermore, experiments show that all versions of GIM-V scale linearly with the number of edges.

Analysis of Real Networks Experiments with LinkedIn social network data, Wikipedia page linking data, and the YahooWeb graph suggest the following properties of connected components in real networks.

- Connected components are formed by processes similar to the *Chinese Restaurant Process* and the *Yule distribution*.
- The distribution of connected components remains stable after a *gelling* point at year 2003.
- The maximum community size in social networks is around 150.
- Replicated web pages served by a domain company resulted in a large number of components with the same structure.
- There are a number of more or less "disconnected" components in the WWW graph, some of them being porn sites.

The PageRank of YahooWeb graph follows a power law distribution. The average diameter of real network graphs is less than 6.09.