Gossip-based Search Selection in Hybrid Peer-to-Peer Networks

M. Zaharia and S. Keshav
School of Computer Science, University of Waterloo, Waterloo, ON, Canada

Abstract: We present GAB, a search algorithm for hybrid P2P networks, that is, networks that search using both flooding and a DHT. GAB uses a gossip-style algorithm to collect global statistics about document popularity to allow each peer to make intelligent decisions about which search style to use for a given query. Moreover, GAB automatically adapts to changes in the operating environment. Synthetic and trace-driven simulations show that compared to a simple hybrid approach, GAB reduces the response time by 25-50% and the average query bandwidth cost by 45%, with no loss in recall. GAB scales well, with only a 7% degradation in performance despite a tripling in system size.

I. Introduction

A hybrid peer-to-peer search network combines an unstructured flooding network with a structured Distributed Hash Table (DHT)-based global index [1,2]. In such networks, partial keyword queries can either be flooded to all peers or the set of peers storing documents corresponding to each keyword can be looked up in the DHT with the results intersected in-network as in [3] or at the initiator. Which search method should a query use? Flooding is efficient for popular (i.e well-replicated) documents but inefficient for rare documents. Therefore, references [1,2] propose to first flood a query to a limited depth, and, if this returns no results, submit the query to the DHT. This allows cheap and fast searches for popular documents and simultaneously reduces the flooding cost for rare documents. However, this comes at the expense of additional infrastructure, as well as an increase in the response time for rare documents and wasted bandwidth due to unfruitful floods.

We present GAB (**G**ossip **A**daptive Hy**B**rid), a gossip-based approach to collect global statistics that allows peers to predict the best search technique for a query. GAB dynamically adapts to changes in the operating environment, making it relatively insensitive to tuning parameters. Its design does not depend on the choice of the DHT or of the unstructured flooding network: any of the solutions described in Reference [4], for example, are adequate.

Compared to a non-adaptive hybrid approach as presented in [1,2], GAB achieves a 25-50% smaller response time, and reduces mean query bandwidth usage by 45%. GAB scales well, with only a 7% degradation in performance despite a 3x increase in system size.

Section II presents GAB and Section III evaluates it using simulations. We describe related work in Section IV and conclude in Section V.

II. Adaptive Search Algorithm Selection

A. The search algorithm selection problem

An ideal search algorithm should return no more than the desired number of results, while minimizing both the query response time and the mean bandwidth cost per query. In a hybrid search network, these criteria are met if a peer can, by looking at the query keywords, decide if these keywords match a widely replicated document or not, using this information to either flood the query or look up the keywords in a DHT. In past work, observations of result size history, keyword frequency, keyword pair frequency, or sampling of neighboring nodes have been used to determine documents rarity to reduce publishing costs [2]. However, this only uses local or neighbor information. Instead, GAB collects global statistics about document availability and keyword popularity using gossip to make the search selection decision.

We describe GAB in the context of a Gnutella-like network where *ultrapeers* index content stored at *end nodes* [5]. With GAB, an ultrapeer uses histograms of (a) the number of other ultrapeer nodes whose indices contain a given keyword and (b) the fraction of ultrapeer nodes that index a copy of a given document to predict the search technique that is best for a given query. It then compares this prediction to an actual measurement of search effectiveness, and uses the error to adapt future predictions. We describe collection of global statistics in Section II.B, use of these statistics in Section II.C, and adaptation in Section II.D.

B. Gathering global statistics

The idea behind our approach is this: when an ultrapeer sees a document title it hasn't indexed already, it tosses a coin up to k times and counts the number of heads it sees before the first tail. It saves this result in a value we call *CT*. The ultrapeer then gossips its *CT* values for all titles with the other ultrapeers. During gossip, for each title, each ultrapeer computes the maximum value of *CT,* i.e. *maxCT*. If a document is widely replicated, its expected *maxCT* value will be larger than the expected *maxCT* value for a rare document. Moreover, the count of the number of ultrapeers with the document is roughly 2^{maxCT}. Using this intuition, each ultrapeer can get an approximate count of the number of other ultrapeers that have that document title (see below for details). Each ultrapeer maintains a histogram of these counts. Note that a gossip-based approach to collecting global statistics is ideally suited to P2P networks because it is decentralized, robust, and results in every peer learning of the global state.

We now formalize this intuition. GAB's gossip algorithm has three components: (1) a synopsis structure that approximates the true histogram of document and keyword popularity (based on a technique first described in [6]) (2) a synopsis fusion algorithm that merges two synopses, and (3) a randomized gossip algorithm to disseminate synopses among ultrapeers [7, 8].

The synopsis generation algorithm uses the duplicate-insensitive counting technique for multisets pioneered by Flajolet and Martin [9]. Consider the synopsis corresponding to a histogram of document (title) popularity. To create this synopsis, each ultrapeer, for each unique document in its collection, does a coin tossing experiment *CT(title, k)*, defined as: toss a fair coin up to *k* times, and return either the index of the first time 'heads' occurs or *k*, whichever is smaller[1]. This is represented by a bit vector of length *k* with the *CT(title, k)*th bit set and $k > 1.5 \log N$, where *N* is the maximum number of ultrapeers. If the first 0 bit counting from the left in *max(CT(title,k))*, where the maximum is computed over all ultrapeers, is at position *i*, then the count associated with that bitvec is, with high probability, $2^{i-1}/0.77351$ (the 'magic number' in the denominator comes from the mathematical principles described in [9]). Note that a synopsis is approximate: the number of documents can be estimated only to the closest power of 2, so the estimate may have an error of up to 50%.

A complete GAB synopsis has three components: a *document title* synopsis, a *keyword* synopsis, and a *node count* synopsis. The title synopsis is a set of tuples {(*title*, *bitvec*)}, where *title* is a list of keywords that describe a document, and *bitvec* is a bit vector representing a coin tossing experiment. The keyword synopsis is similar. Finally, the node count synopsis is a bit vector counter that counts the *number* of nodes represented by that complete synopsis. The complete synopsis therefore is of the form {node_count_bitvec, {(title, bitvec),…, (title, bitvec)}, {(keyword, bitvec),…, (keyword, bitvec)}}.

The synopsis fusion algorithm is: (a) If two tuples in the combined title or keyword synopses have the same title or keyword, then take the bitwise-OR of the two corresponding bitvecs. This has the effect of computing the max of the two bitvecs in an order-insensitive manner. (b) Update the node count synopsis using the local value of the node_count bitvec. (c) To keep a synopsis from growing too large, if the size of the fused synopsis exceeds a desired limit *L*, discard tuples in order of increasing bitvec value (treating the bitvec as a *k*-bit integer) until the limit is reached. Note that at start time, synopsis counts are small, and it is possible that a popular document (with a small bitvec count) may be accidentally pruned. To compensate, a ultrapeer can skip the pruning step if the value of the node counter in the union of the synopses is smaller than some predefined threshold.

During initialization, each ultrapeer generates a synopsis of its own document titles and keywords and labels it as its 'best' synopsis. In each round of gossip, it chooses a random neighbor and sends the neighbor its best synopsis. When a node receives a synopsis, it fuses this synopsis with its best synopsis and labels the merged synopsis as its best synopsis. As with any other gossip algorithm, this results in every ultrapeer, with high probability, getting the global statistics after (*log N*) rounds of gossip.

Note that the gossip adds a bandwidth overhead to the system, which is the price to pay for getting global statistics. However, this cost is paid rarely since global statistics change rarely. Moreover, the cost is amortized over all the queries in the system, so, as the search load increases, the amortized cost for gossip decreases.

C. Search selection using global statistics

Given a synopsis and a set of query keywords, an ultrapeer first determines if it has sufficient local matches. If so, it is done. If not, it computes the expected number of results for that set of keywords as follows: it adds the approximate counts for the titles in its 'best' synopsis that contain all the query keywords. We denote by *r* the sum of these approximate counts divided by the approximate number of ultrapeers *N*. For a given query, *r* represents the expected number of matching titles at

[1] Generation of such a 32-bit vector is fast, requiring only one multiplication and addition operation for linear congruential random number generation [10], followed by the x86 Bit Scan Forward instruction and a lookup in 32-element pre-computed bit vector array.

any ultrapeer. The greater the number of titles in the P2P network that match a particular set of keywords, the larger the value of *r*. If *r* exceeds a threshold *t*, many matches are expected, so the ultrapeer floods the query. If the flood returns no results after a conservative timeout, it uses the DHT to search for each keyword, requesting an in-network join, if that is possible.

If $r < t$ and *any* keyword in the query is *not* in the common keyword synopsis, the ultrapeer uses the DHT because a join is cheap if it is initiated using this keyword [3]. Otherwise, the document is both rare and has common keywords, so the only option is to flood the query. This is done, if possible, with an indication that this query has low priority. The idea is that flooding will need a large flood depth. The low priority ensures that the request is handled only if there is adequate search capacity.

D. Adaptation of flood threshold

An important parameter in our system is the flooding threshold, *t*. If *t* is too small, then too many documents will be flooded and *vice versa*. In either case, the system will be inefficient.

Unfortunately, it is hard for a system administrator to choose a threshold value that is optimal for all operating environments. Worse, the threshold can change over time depending, among other things, on the number of end nodes, the number of documents they store, the search load, and the available bandwidth to each ultrapeer. Therefore, GAB adjusts *t* over time, instead of using a fixed value. Adaptive thresholding also makes GAB more robust: in case of failure of the DHT, all queries would eventually be flooded because *t* would rapidly increase.

Intuitively, an ultrapeer should choose a search algorithm that maximizes a search's utility. For widely-replicated documents, where the expected number of results per node is large, flooding provides more utility than DHT search, and for unpopular ones, DHT search provides more utility. GAB adapts the flooding threshold by computing the utility of *both* flooding and DHT search for a randomly chosen set of queries. If the current threshold is correct, then when GAB chooses to flood, the utility from flooding that query should be greater than the utility of using a DHT and vice versa. Otherwise, the threshold should be modified so that future queries make the right choice.

Adapting the threshold therefore requires us to define a utility function quantitatively. We base our measure of utility on four considerations. First, getting at least one result is a lot better than getting none. So, the first term represents the benefit from this. Second, because there is little use in finding thousands of results if a user is just searching for one particular document, the marginal utility per extra utility should decrease sharply beyond some point. We approximate this by choosing some maximum number of results requires, R_{max}, beyond which each further result contributes zero utility. Therefore, the utility of receiving *R* results is proportional to min (R, R_{max}). Third, since only the first *Rmax* results are useful to the user, the utility should be proportional to the response time *T* of the min(R, R_{max})'th result, which we call the *last response time*. Finally, the cost of a query should include its bandwidth cost *B*. Assuming linearity, we denote the utility function *U*:

$$U = (R > 0?1:0) + w_1 * min(R, R_{max}) - w_2 * T - w_3 * B$$

where w_1-$_3$ are normalized weights chosen by a user. We choose to add and subtract the components of utility instead of multiplying or dividing them to prevent large fluctuations when one of the numbers is very small or very large. Note that R, R_{max}, T and B can be computed for each search if *B* is carried with each search request and reply.

The optimal value for *t* is the point of indifference, where flooding and DHT search provide equal utility. This motivates the following algorithm:

1. For each query, compute *r*, the expected number of results per ultrapeer. Because we expect variations in *t* to be small, we obtain more measurements around the current operating point by choosing *p*, the probability that this query will be used for adaptation, to be a linear function of $|r - t|$.
2. With probability *p*, use both flooding and DHT for the query and carry out steps 3 and 4.
3. Compute the utilities of each type of search.
4. Each query results in the computation of two data points (r, u_f) and (r, u_d), where u_f is the utility from flooding, and u_d is the utility from a DHT search (refer to Figure 1). If $r > t$, we expect $u_f > u_d$, otherwise, $u_f < u_d$

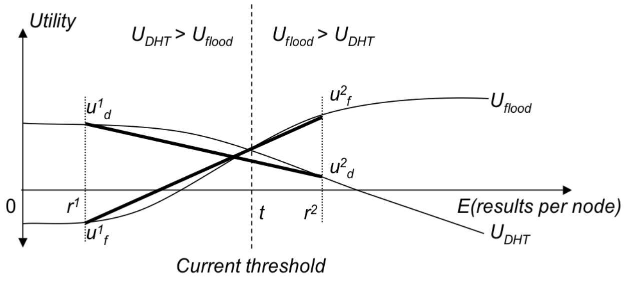

Figure 1: Adapting the flood threshold

5. After Q queries, we need to update t. Intuitively, to first order, we can approximate the utility from flooding and from a DHT as lines, so that their intersection is a reasonable estimator for t. Exponential averaging of this estimate allows us to deal with noisy estimates. Thus, for every two pairs of points $\{(r^1, u_f^1), (r^1, u_d^1)\}$ and $\{(r^2, u_f^2), (r^2, u_d^2)\}$ such that $r^1 < t < r^2$ let x be the X coordinate of the intersection of the line passing through (r^1, u_f^1) and (r^2, u_f^2) and (r^1, u_d^1) and (r^2, u_d^2). We set the new estimate of t to be the median x value from these pairs and use this to update t using an exponential averager with a forgetting factor of 0.05.
6. Optionally, the value of t computed at each ultrapeer can be gossiped so that every node is aware of the average value of t and uses this in its prediction.

III. EVALUATION

We wrote a custom simulator in Java to compare GAB with other well-known search techniques. Details about our simulator can be found in [11]. The simulator accurately models end-node lifetimes and link capacities [12,13] as well as a flooding network and a Chord-like DHT. New end nodes join the system at a rate of 0.75 end nodes a second, bringing an average of 20 documents into the system, randomly chosen from a dataset of 20,000 unique documents, when they register with an ultrapeer. These documents are then are indexed by the DHT. End nodes emit queries on average once every 300 seconds, requesting at most 25 results. Documents and keywords are assumed to have a Zipfian popularity distribution with a Zipf parameter of 1.0.

When an end node leaves, we model the deletion of its index both from ultrapeers as well as from the DHT. By modeling end node churn, which is an important factor in real peer-to-peer systems, we capture the costs of DHT and ultrapeer updates both on node arrival and on departure. Note that the because node lifetimes are chosen from a fixed distribution, we can increase the number of documents in the system, the total node population, and the number of simultaneously active node simply by modifying the node arrival rate.

We simulated about 1.7 million queries over a 22 hour period. We observed that a stable online population of about 10,000 active end nodes and 500 ultrapeers was achieved after about 20,000 simulated seconds (~6 hours). Therefore, results are presented only the queries made between 40,000 and 80,000 seconds. With these parameters, the total population over the simulation lifetime was about 91,000 end nodes. Although these numbers are still about an order of magnitude smaller than a real system, we believe that it is large enough for us to get meaningful comparisons between various search approaches. In a realistic system, the response times and bandwidth costs will be about ten times larger. However, we expect the relative costs and benefits to be roughly the same as in our simulations.

We generated queries and document/keyword sets in two different ways. First, we generated random exact search requests according to the fetch-at-most-once model in [13] (for results in III.A-III.C). We only generated exact queries, since it is difficult to generate realistic partial queries. Second, we played back partial keyword searches from the Gnutella data set in [2] (for results in III.D). We compared GAB with the following algorithms:

Pure DHT	All queries looked up in a DHT using the in-network adaptive join method of [3]
Simple Hybrid	Models [2]: queries are first flooded to depth 2, then looked up in a DHT if fewer than 25 results are received from the flood after 2s.
Central Server	An ideal central server with zero request service time.

We compare the results for each approach using the following metrics:

Recall	Percentage of queries that found a matching document, given that there exists at least one available document that matches the query
FRT	Mean first response time for successful queries (seconds)
LRT	Mean last response time i.e response time for R_{max}th, query for successful queries (seconds)
BWC	Bandwidth cost in kilobytes per query; the cost of publishing and gossiping is also included in this cost (Kilobytes/query)

To save space, we only report means, and for ease of comparison, we present normalized results for FRT, LRT, and BWC. Standard deviations, which are all well under 5% of the mean value, are reported in an extended version of this paper [11]

A. *Search approaches compared*

System	Recall	FRT	LRT	BWC
Pure DHT	99.9%	1.18	0.62	1.63
Simple Hybrid	99.9%	1.00	1.00	1.00
GAB	99.9%	0.70	0.57	0.51
Central Server	100.0%	0.41	0.21	0.22

The pure DHT approach has poor (and almost identical, though this is not apparent from the table) first and last response times because it cannot exploit document popularity to reduce response times. It also has the highest bandwidth cost. The hybrid approach of [1,2] when compared to a pure DHT, reduces the first response time by about 20%. It also uses far less bandwidth because it avoids DHT lookups for popular documents. Unfortunately, it has a higher average last response time because rare documents must be both flooded and looked up.

GAB performs much better than Simple Hybrid. Its first and last response times are both lower than Simple Hybrid (and Pure DHT) because queries for known rare documents are sent directly to the DHT rather than being "tested" using a flood. Moreover, bandwidth costs are nearly halved because GAB saves doing a flood for queries that are sent directly to the DHT. This validates the gain in performance by the use of global statistics.

A central server has perfect recall, 55% lower FRT and 70% lower LRT than even GAB. Moreover, the bandwidth cost is also 57% lower. We conclude that the price to pay for decentralization is roughly a doubling of every performance metric.

B. *Adaptive thresholding*

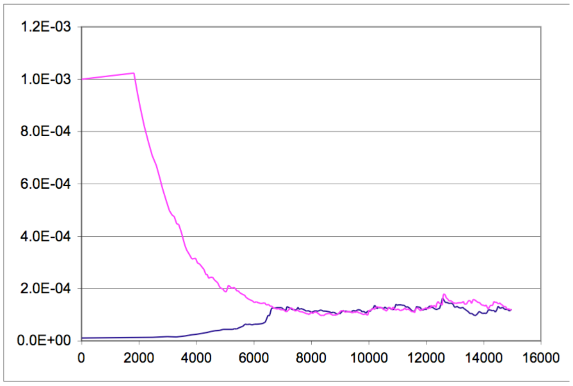

Figure 2: Threshold value t *vs. time*

Figure 2 shows the times series of flooding threshold values *t* at two ultrapeers for a particular simulation run. For this environment, the optimal value of *t* is around 1.0E-4. The nodes chose initial values of 1.0E-3 and 1.0E-5. Over time, both converge to the optimal value, illustrating GAB's adaptive behavior.

C. *Scalability*

Intuitively, GAB should scale well with increases in end node population because DHT and gossip costs increase logarithmically with system size, and flooding costs, for a fixed flood depth and node degree, are constant. We validated this intuition by choosing three different values for the mean end node inter-arrival time: 1.0s, 0.4s, and 0.3s. As described earlier, this changes the mean number of end nodes in the system. The approximate stable active populations were, respectively, 7000 end nodes, 17,500 end nodes, and 23,300 end nodes, corresponding to end node populations of roughly ten times this size. The results below are normalized.

Active Population	FRT	LRT	BWC
7000	1.00	1.00	1.00
17,500	1.00	0.85	1.06
23,300	1.03	0.80	1.07

We observe that as the population more than triples, the first response times increase by 3% due to the need to consult larger DHT indices for rare items. However, the DHT is used only about 20% of the time, so its effect on the overall average is negligible. Note that LRT actually *decreases* slightly with increase in population size because, with more nodes, sufficient numbers of results are found with shallower floods. Bandwidth costs increase by about 7%, again mostly due to larger DHTs but also because as the number of users increases while keeping node degree and flood depth constant, the fraction of non-back edges increases and a flood is more widely propagated.

D. *Trace-based simulations*

To validate the conclusions from synthetic-workload based simulation, we ran our experiments on a trace-based workload. The traces use the Planetlab-based monitoring infrastructure described in [2], and were obtained by simultaneously monitoring the queries and the results of these queries at 50 ultrapeers for 3 hours on Sunday October 12, 2003. This represents 230,966 distinct queries, 199,516 distinct keywords and 672,295 distinct documents.

System	Recall	FRT	LRT	BWC
Simple Hybrid	87.0%	1.00	1.00	1.00
GAB	87.3%	0.59	0.45	0.67

For this more realistic workload, preliminary results show that GAB has a 41% lower FRT than a simple hybrid (compared to 30% in simulations); 55% lower LRT (43% in simulations), and 33% lower bandwidth cost (49% in simulations). Both systems have a lower recall than in simulations. We attribute these discrepancies to the fact that trace-based queries are partial-keyword queries, and therefore require fewer lookups than the full-keyword queries in the simulations, reducing the response time. The recall is lower because the number of distinct documents is much larger than with our simulations. Nevertheless, overall trends in synthetic and trace-based simulations agree.

IV. Related Work

Numerous DHT-based search systems have been proposed in the literature; an overview of these can be found in [4]. Extensions to DHTs to allow searches using only a subset of document title's keywords have been proposed in [3] and [14], among others. Hybrid systems combining DHT and flooding networks are described in [1,2]. GAB builds on and extends this work by proposing gossip-based algorithms for search selection.

Gossip systems are well known in the literature, where they are also called epidemic algorithms [7, 8, 15]. We refer interested readers to [16] for an overview and survey of recent work in this area.

V. Conclusions and Future Work

Our work makes two main contributions. First, we show how gossip-based computation of global statistics improves search efficiency, reducing both response time and bandwidth costs. Second, we show how to adapt a critical tuning parameter, the flood threshold, to changes in the operating environment. The use of a decentralized gossip-style state computation, combined with a DHT removes all centralized elements from our system, which permits good scalability. The adaptation process uses user utilities, and this allows system behavior to be controlled by intuitive 'control knobs'. We believe that the use of gossip to compute global state and the explicit use of utility functions to modify system behavior, are applicable to any large-scale distributed system.

We quantified gains from GAB using simulation on both synthetic and trace-based workloads. We found that, compared to a simple hybrid approach, our search algorithm can roughly halve the last response time and bandwidth use, with no loss in recall. Our algorithm scales well, with only a 7% degradation in performance with a 3x increase in system size.

We have implemented our system by modifying the Phex Gnutella client to use the OpenDHT framework. In current and future work, we plan to quantify the benefits from our algorithms for more realistic workloads.

VI. Acknowledgement

We would like to gratefully acknowledge Boon Loo at UC Berkeley for his traces of the Gnutella workload.

VII. References

[1] B.T. Loo, R. Huebsch, I. Stoica, and J.M. Hellerstein, "The Case for a Hybrid P2P Search Infrastructure," *Proc. IPTPS,* 2004.

[2] B.T. Loo, J. M. Hellerstein, R. Huebsch, S. Shenker and I. Stoica, "Enhancing P2P File-Sharing with an Internet-Scale Query Processor," *Proc. 30th VLDB Conference,* 2004.

[3] P. Reynolds, and A. Vahdat, "Efficient Peer-to-Peer Keyword Searching," *Proc. Middleware*, 2003.

[4] H. Balakrishnan, M.F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Looking Up Data in P2P Systems," *Comm. ACM*, Vol. 46, No. 2, Feb, 2003.

[5] Gnutella, `http://www.gnutella.com`

[6] S. Nath, P. Gibbons, S. Seshan, and Z. Anderson, "Synopsis Diffusion for Robust Aggregation in Sensor Networks," *Proc. SenSys*, Nov. 2004.

[7] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-Based Computation of Aggregation Information," *Proc. IEEE FOCS*, 2003.

[8] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Gossip Algorithms: Design, Analysis, and Applications," *Proc. INFOCOM 2005*, March 2005.

[9] P. Flajolet and G.N. Martin, "Probabilistic Counting Algorithms for Database Applications," *J. Computer and System Sciences*, Vol. 31, 1985.

[10] D. Carta, "Two fast implementations of the "minimal standard" random number generator," *Comm. ACM*, Vol. 33, No. 1, pp.87-88, 1990.

[11] M. A. Zaharia and S. Keshav, "Efficient and Adaptive Search in Peer to Peer Networks," U. Waterloo Technical Report 2004-55, 2004.

[12] S. Saroiu, K.P. Gummadi, S.D. Gribble, "Measuring and Analyzing the Characteristics of Napster and Gnutella Hosts," *Multimedia Systems Journal*, Vol. 9, No. 2, pp. 170-184, August 2003.

[13] K.P. Gummadi, R.J. Dunn, S. Saroiu, S.D. Gribble, H.M. Levy, and J. Zahorjan, "Measuring, Modeling and Analysis of a Peer-to-Peer File-Sharing Workload," *Proc. 19th SOSP*, October 2003.

[14] S. Dwarkadas, and C. Tang, "Hybrid Global-Local Indexing for Efficient Peer-to-Peer Information Retrieval," *Proc. NSDI*, 2004.

[15] A. Demers et al "Epidemic algorithms for replicated database maintenance," *PODC*, 1987.

[16] S. Keshav, "Efficient and Approximate Computation of Global State," *Manuscript under submission*, http://www.cs.uwaterloo.ca/~keshav