Server Selection with Arbitrary Distribution

Xinjie Li and Monica Brockmeyer*
Department of Computer Science
Wayne State University, Detroit, MI 48202
xinjieli@wayne.edu mab@cs.wayne.edu

Abstract

Many applications need to pick servers with some desired distribution. For example, in probabilistic quorum systems, one method to generate quorums with high probability of intersection is to randomly pick $k\sqrt{n}$ nodes with a fixed probability distribution. Load balancing applications may need to take several samples of the servers with some desired distribution. Existing approaches realize a fixed stationary distribution by controlling the topology of the overlay graph and conducting random walks on it. In particular, existing approaches focus on achieving a uniform distribution. This paper proposes using the distributed Hastings-Metropolis algorithm to achieve any *desired stationary distribution* without *control or global knowledge of the overlay graph. The new method facilitates good load balancing, since heterogeneous server capacity or other factors can be considered in deciding the appropriate distribution by which to pick servers.*

1 Introduction

We will take *probabilistic quorum systems* (PQS) [1, 7] as the motivating example application of picking servers with desired distribution. This is mainly because this method was first proposed when we were studying PQS [5]. However, the understanding of PQS is not necessary in understanding our proposed method. Any application that needs to pick servers with some desired distribution could use our method. In many cases, the task of selecting a quorum is just the same as that of selecting a group of servers. In the following text, the term *quorum* and *a group of servers* are used interchangeably. The biggest challenge in selecting a group of servers (a quorum) is how to find them *without* global knowledge.

In *probabilistic quorum systems* (PQS) by Malkhi *et al*. [7], the quorums are chosen such that any two quorums intersect with high probability. In one approach [7], a client can pick a quorum by choosing $k\sqrt{n}$ servers with uniform probability from the total n servers, where the k is a safety parameter. Any two quorums picked by this method intersect with probability $1 - (e^{-k^2})$, which is close to 1 with only a moderate k.

Abraham and Malkhi [1] describe an approach that picks quorums from a dynamic system without global knowledge. They first prove that when the client is selecting $k\sqrt{n}$ out of n servers, quorum members do not have to be picked uniformly at random. As long as the members are chosen according to a fixed probability distribution p, any two quorums of size $k\sqrt{n}$ will intersect with high probability. To realize a fixed p in a dynamic system, they propose careful emulation of a dynamic De Brujin graph structure, which rapidly mixes to a stationary distribution in logarithmic steps. To pick a quorum, the client starts $k\sqrt{n}$ number of random walk messages in parallel, each of which picks a server after logarithmic number of steps.

Note that even though any realized fixed probability distribution p guarantees sufficient likelihood of quorum intersection, it may not also be an appropriate probability distribution for load balancing purposes. For example, if all the servers have the same capability, the realized p should be uniform to achieve load balancing. The probability realized in [1] is fairly close to uniform. The probability of node i being picked, $p(i)$, is $\frac{1}{2^{l(i)}}$, where $l(i)$ is the *level* of node i in the dynamic graph. The level gap, i.e., the difference between the levels, among all the nodes in the graph could be maintained with high probability within a constant bound with a $O(log(n))$ message overhead or within two with a $O(n)$ message overhead. For any particular server node, the realized likelihood of being selected depends on the relative order in joining the overlay graph and also on the *random* choices in the algorithm. As a result, the server selection probability is not controllable using this approach.

Our research addresses the more common situation, where complete control over the system topol-

*This material is based upon work supported by the National Science Foundation under grant CAREER-0347222.

1-4244-0030-9/05/$20.00 ©2005 IEEE

ogy is not present. This situation in not unusual, because security concerns or administrative constraints limit the channels, where other factors determine the overlay topology, such as Distributed Hash Tables[10], or where maintaining a specific overlay incurs undue cost. Moreover, our approach permits control over the server picking distribution, without depending on an estimation of the network size n.

Generally speaking, for any random walk, its stationary distribution is determined by two factors, the graph topology and the forwarding probability on each edge. Since we are interested in the situations where no control over the graph topology can be assumed, the only mechanism under our control is to adjust the forwarding probabilities on the edges. To achieve any desired distribution only by tuning forwarding probability, we use the Hastings-Metropolis [3] algorithm, which generates a Markov chain with a desired stationary distribution. When the H-M algorithm is used, knowing the mixing time[1] is important, we will present our work-in-progress on this issue.

The rest of this paper is structured as following, Section 2 introduces the distributed H-M algorithm; Section 3 discusses our work-in-progress on some situations under which the H-M algorithm could be fast mixing and the mixing speed could be estimated; Section 4 discusses a store-and-retrieve method to lower the client perceived quorum obtaining delay when the mixing time is hard to estimate.

2 The Distributed Hastings-Metropolis Algorithm

The Hastings-Metropolis Algorithm The Hastings-Metropolis (H-M) algorithm [3] was first developed in the context of the Markov chain Monte Carlo (MCMC) method to generate a Markov chain with a desired stationary distribution specified up to a multiplicative constant. The Hastings-Metropolis algorithm uses a *candidate-generating distribution* $q(i,j)$ for each state i and j such that $\sum_{j=1}^{n} q(i,j) = 1$, for each i. All the $q(i,j)$'s form a Markov transition probability matrix $\mathbf{Q}$. There are few restrictions on the selecting of $\mathbf{Q}$ as far as the correctness of the H-M algorithm is concerned. The only requirement is that the matrix represents an ergodic [2] Markov chain. Different choices of $\mathbf{Q}$, however, do make a difference in the execution speed of H-M algorithm. The desired distribution is specified as unnormalized $b(j)$, where the normalizing factor $B = \sum_{j=1}^{\infty} b(j)$ is finite but unknown. The desired distribution, after normalization, will be

$$\pi(j) = b(j)/B$$

The H-M algorithm creates a time-reversible Markov chain with stationary distribution π by using $\mathbf{Q}$ and the unnormalized distribution b. From a current state i, a tentative next state j is selected according to $\mathbf{Q}$, however this tentative move is accepted with a probability determined by :

$$\alpha(i,j) = min\left(\frac{b(j)q(j,i)}{b(i)q(i,j)}, 1\right) \quad (1)$$

So the new Markov chain has the following transition matrix $\mathbf{P}$ given by:

$$P(i,j) = q(i,j)\alpha(i,j), \text{ if } i \neq j$$

$$P(i,i) = q(i,i) + \sum_{k \neq i} q(i,k)(1 - \alpha(i,k))$$

As we can see from Equation 1, the desired distribution can be specified in an unnormalized fashion because it appears both in the numerator and denominator. As a consequence, the unnormalized probability distribution can be specified *locally* and without the knowledge of system size n, which is the key to the ability to achieve a distributed implementation.

As we mentioned earlier, in choosing the candidate generating distributions $\mathbf{Q}$, cautions must be taken so it is ergodic. If we do not worry about periodicity, a simple random walk

$$q(i,j) = \frac{1}{|N(i)|} \text{ if j} \in N(i)$$

would suffice. To handle the periodicity issues, a standard technique[9] is to add a self link for each node and assign probability 0.5 to the self transition. The modified random walk is given by:

$$q(i,j) = \frac{1}{2|N(i)|} \text{ if j} \in N(i)$$

$$q(i,i) = 1/2$$

Without the use of H-M algorithm, the stationary distribution of this modified random walk is [9]:

$$\mu(i) = degree(i)/2m,$$

where m is the total number of edges. We can see here that with simple random walk, the stationary distribution is completely determined by the topology of the graph.

[1] The number of steps for the Markov chain to stabilize to its stationary distribution

[2] Informally, it means starting from any state, every other state is reachable within finite steps (positive recurrent) and the number of steps needed does not have to be a multiple of some number (aperiodic).

When using the H-M algorithm, the desired stationary distribution is controlled by setting different b's. For example, if the desired accessing probability for a node is proportional to its capability, $b(j)$ is simply set to be the capability of server j. For another example, if the desired distribution is uniform, $b(j)$ will simply be set to some constant C for all servers j. Note that it is possible to control the server selection probability without knowledge of the size of the underlying graph.

The Distributed Hastings-Metropolis Algorithm We see that the H-M algorithm needs only *local* information. As a result, it is very easy to distribute. The distributed H-M algorithm works as follows:

1. Each node i is initialized with a local unnormalized probability $b(i)$.
2. Each node sends its outward degree information to its neighbors.
3. Each node calculates $\alpha(i, j)$ according to equation 1 and hence determines the new forwarding probability $p(i, j)$.
4. A random walk message is forwarded to the next node with the probability given by $p(i, j)$. Note that $p(i, i) > 0$, so with non-zero probability, the next node is the current node itself. No matter if the next node is the a new node or not, it is considered a *step*. A counter in a random walk message records the number of steps it has taken.
5. After a *certain* number of steps, when the Markov chain *mixes* to the desired distribution, the random walk stops and the node at which the message stops is picked.

Note that step 1 needs to be done only once, and step 2 and 3 need to be done only upon topology change. Further, when a node joins or an leaves the overlay, only nodes that are within one hop of the change need to be notified. The H-M algorithm guarantees that when the Markov chain mixes, the probability that the servers get picked follows the desired distribution by the user.

3 Mixing speed of the H-M algorithm

The distributed algorithm requires that the random walk stops after a pre-determined number of steps. To implement the distributed H-M algorithm, we need to know the steps required for the Markov chain to mix up. It is known that the mixing speed of H-M algorithm depends on the eigenvalues of the transition matrix $\mathbf{P}$ [2]. The Markov chain approaches its stationary distribution geometrically with respect to the its second largest eigenvalue λ_2. However, we do not have a global view of the graph so we do not know $\mathbf{P}$ nor λ_2.

Fortunately, some research results suggest that, we may be able to estimate the mixing speed without global information. The Faloutsos *et. al.* [4] showed that the eigenvalues of the Internet graph may follow the power-law distribution. For overlay networks, Lv *et. al.* showed that the node degree of the Gnutella networks roughly follows a two-segment power-law distribution [6]. Please note that the origin of power-law property is still under research so the assumption that the eigenvalues of unstructured peer-to-peer networks also follows a power-law distribution is subject to future verifications. If such assumption holds, it may enable us to estimate the mixing speed of the pure random walk on that graph. For the example given in [4], for the three Internet graph observed in the year 1997 and 1998, the eigenvalues roughly follow a power-law distribution with exponent parameter -0.48 (Please refer to Fig 11(a) in [4]). More specifically,

$$\lambda_i \propto i^{-0.48} \tag{2}$$

Given the power-law property, there are still two gaps to be filled before we can estimate the mixing speed of the random walk. The first gap is that the observed power-law property is on the eigenvalues of the adjacency matrix $\mathbf{A}$ of the underlying graph. However, to estimate the mixing time of the random walk, we need to know about the eigen-gaps on the induced stochastic matrix $\mathbf{Q}$. It is easy to see that $\mathbf{Q} = \mathbf{DA}$, where $\mathbf{D}$ is the diagonal matrix in which the ith diagnoal entry is $1/d(i)$. For a power-law graph, the eigenvalues $\mathbf{A}$ and $\mathbf{Q}$ are generally different. We are currently working on finding the relationship between the eigen-gaps of the two matrices.

The second gap is that when the H-M algorithm is employed, fast mixing on the original chain ($\mathbf{Q}$) does not necessarily imply fast mixing on the new chain induced by H-M algorithm ($\mathbf{P}$). The new chain generally takes more steps to mix up, as the H-M algorithm could be thought of as a filter, and depending on the candidate chain $\mathbf{Q}$ and the desired distribution π, a lot of tentative moves could be filtered out.

We have yet to determine that the number of steps that the random walk message should take. It depends on the relationship between the candidate generating distributions $\mathbf{Q}$ and the desired distribution π, Mengersen and Tweedie [8] showed that geometric

convergence of the H-M algorithm happens "if and only if the candidate density is bounded below by a multiple of π". We are currently exploring on how to check or estimate if this condition holds on the distributed graph.

4 Efficiency in Selecting a Group of Servers for General Graph Structure

Not all overlay graphs follow the power-law distribution, so in general, it is very hard to determine the mixing speed of an overlay graph as we do not have the global knowledge. As a result, we have to make a very conservative decision in the number of steps a random walk message takes. As a consequence, starting the random walks upon the client request will result in unacceptably high delay.

To reduce the latency felt by clients in selecting a group of servers (a quorum in PQS), we propose using a background quorum selecting process to alleviate this problem. This quorum selecting process is a random walk, which picks servers, forms quorums and *stores* the quorums in the system. Specifically, it does the following:

1. One or more quorum generating random walks are started in the system.
2. After a certain number of hops, H, when the Markov chain has become stationary, each quorum generating random walk picks the current node, appends the information in the random walk packet and continues the random walk.
3. When enough number of nodes ($k\sqrt{n}$ for example) have been picked by the random walk, which means a quorum has been identified, the quorum generating process stores the quorum information at *some* node.

Since the quorums are to be cached by the quorum gathering process and later retrieved by the clients, an efficient store-and-retrieve mechanism needs to be employed. In particular, the quorum information can be stored right at the last server that was picked as in that quorum; and the clients' request can be routed according to the random walk. We are currently investigating more efficient mechanisms on storing and retrieving the quorum information.

5 Conclusion and Future Work

This paper proposes a new method based on Hastings-Metropolis algorithm to control the server selection distribution without knowledge or control of the overlay topology. We presented our in progress on some graphs where the speed of mixing could be fast. For the general case, we presented a store-and-retrieve mechanism to alleviate the client-perceived delay. We are currently working on better estimation of the mixing speed of this method and the storage mechanism of quorums.

References

[1] I. Abraham and D. Malkhi. Probabilistic quorums for dynamic systems. In *Proc. of the 16th International Symposium on Distributed Computing (DISC'03)*, 2003.

[2] P. Bremaud. *Markov Chains*. Springer, 1999.

[3] Siddhartha Chib and Edward Greenberg. Understanding the metropolis-hastings algorithm. *The American Statistician*, 49(4), November 1995.

[4] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM*, pages 251–262, 1999.

[5] Xinjie Li and Monica Brockmeyer. Brief announcement: controlled quorum selection in arbitrary topologies. In *PODC*, page 321, 2005.

[6] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and replication in unstructured peer-to-peer networks. In *ICS '02: Proceedings of the 16th international conference on Supercomputing*, pages 84–95, New York, NY, USA, 2002. ACM Press.

[7] Dahlia Malkhi, Michael K. Reiter, and Rebecca N. Wright. Probabilistic quorum systems. In *Symposium on Principles of Distributed Computing*, pages 267–273, 1997.

[8] K. Mengersen and R. Tweedie. Rates of convergence of the hastings and metropolis algorithms, 1996.

[9] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[10] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM 2001*, 2001.