SplitQuest: Controlled and Exhaustive Search in Peer-to-Peer Networks

Pericles Lopes
pericles@facom.ufms.br

Ronaldo A. Ferreira
raf@facom.ufms.br

College of Computing, Federal University of Mato Grosso do Sul, Brazil

Abstract

This paper presents SplitQuest, a controlled and exhaustive search protocol that relies on a hybrid network structure to avoid unnecessary query replication and to speed up query propagation in P2P networks. In SplitQuest, peers are organized in replication groups, in which each peer shares its contents with all members, and queries are propagated only once to a group. By avoiding query duplication, directing queries to disjoint groups, and exploiting peers' heterogeneity, SplitQuest is able to achieve high levels of recalls and low response times, while incurring very low overhead. We simulate SplitQuest using synthetic and traces of real-world topologies and show that it outperforms the best known solution in number of messages, response time, number of hops, and success rate for query resolution, while being resilient to high churn rates. We also derive upper bounds on query routing for SplitQuest.

1 Introduction

In recent years, peer-to-peer (P2P) systems have emerged as a powerful networking paradigm that allows sharing of a multitude of resources at the edge of the Internet in a completely distributed manner. More important, this new paradigm gave life to several new and exciting applications, such as instant messaging, internet phone (Skype), video distribution, distributed computation (SETI@Home), and many others. Concomitantly, end users have become enthusiastic producers of information that needs to be shared with the rest of the world. This last fact is particularly evident if we consider the increase in the number of personal blogs and Wikis in the Internet. To maintain these sources of information, users still rely on centralized infrastructure. One of the reasons is because, despite numerous research efforts, efficient complex queries in P2P networks still remain an open and challenging problem.

Search in structured P2P networks has elegantly and efficiently been solved for hash-based queries [6, 5]. However, the overhead of keeping a well defined topology, in which peers have to maintain many pointers to other peers in a dynamic population, is often a source of criticism. Moreover, the query semantic is very limited, since a user must know exactly the name of the object to determine the rendezvous peer responsible for keeping a replica of the object or its reference. Complex queries, composed by multiple keywords, can be implemented, but at higher, and sometimes prohibitive, costs.

Exhaustive search has been recently introduced for unstructured P2P networks [3, 8] as an effective and reliable search method. In this type of search, a query is evaluated against all data in the network and each peer is free to choose a local search algorithm. The query semantic is greatly improved, in comparison with hash-based queries, since sophisticated algorithms can be run locally in each peer. However, the most efficient solutions rely on random walks [3], or their variants with multiple branching factors [8], to provide probabilistic guarantees. As it is well known, random walk is a blind search method that may visit the same peer more than once, generating unnecessary overhead, and that does not guarantee network coverage, unless a significant number of messages is generated. Moreover, both solutions replicate large amounts of data on random peers in order to reduce query response time and improve query success.

In this paper, we propose SplitQuest, an exhaustive search protocol that relies on a hybrid structure to avoid unnecessary query duplication and to guarantee network coverage in P2P networks. In SplitQuest, peers are uniquely identified and placed uniformly on a virtual ring. This lightweight structure is embedded on a random graph in which peers also keep connections to other randomly selected peers, as in a pure unstructured P2P network. The number of random connections of each peer varies depending on its available resources. This heterogeneity of peer connections is intelligently explored by SplitQuest, which aggressively propagates queries on highly connected peers. Moreover, the simple structure allows SplitQuest to direct queries to specific parts of the network, avoiding the overhead of query du-

plication that is present in blind search methods, as random walks.

SplitQuest also relies on replication to achieve high levels of recall and to speed up query resolution. Peers are organized in replication groups, in which each peer shares its contents with all members of the group. Groups are formed by peers on a contiguous segment of the virtual ring. The length of the segment determines the size of the group and, consequently, the tradeoff between index replication and query overhead.

2 The SplitQuest Protocol

A replication strategy in a P2P network, coupled with a search protocol, must ensure that a query is evaluated on every data. In structured P2P networks, a query is directed exactly to a peer where the sought object is supposed to be installed. As it is well discussed in the literature, the query semantic in this case is very limited, because objects can be queried only on their hash values. Unstructured P2P networks, on the other hand, offer a richer semantic at a higher cost of replicating meta information on several peers or flooding a query to a large fraction of peers. Sublinear solutions in unstructured networks [3][8] replicate meta information on a random set of peers and, similarly, query another set of randomly selected peers. These solutions rely on random walks to offer probabilistic guarantees. Random walk, however, is a blind search method that does not guarantee network coverage and does not prevent a query to be evaluated at the same peer more than once, generating unnecessary overhead.

SplitQuest avoids the overhead of previous search methods by imposing a lightweight structure on a mostly unstructured network that allows queries to be directed to specific and distinct parts of the network. In addition to random connections, normally present in unstructured networks, SplitQuest places peers on a virtual ring, i.e., each peer has a predecessor and a successor, and uniquely identifies them with identifiers in the interval $[0, 1]$. Moreover, peers store the IDs of their neighbors locally. This might look similar to structured networks, but as it will be discussed in the next sections, our approaches to peer join, data replication, and query propagation are rather different. Peer connections are still random, there is no well defined routing substrate, and queries are loosely and randomly propagated in the network. Most importantly, SplitQuest allows richer queries, not limited to hash values, since each peer is free to execute the most convenient local search algorithm.

2.1 Index Replication

The basic assumptions in SplitQuest are that peers are uniformly distributed in the interval $[0, 1]$ and that each peer can estimate the network size (n). Network size estimation is a well-studied problem and can be solved by different approaches, as described in [8] and references therein.

Index replication in SplitQuest is performed on a set of random peers located on a contiguous subinterval of the $[0, 1]$ interval. When peers are uniformly distributed in the whole interval, the expected number of peers in a contiguous subinterval is proportional to the subinterval length. Consequently, the basic problem in this case is to determine the subinterval length, or more specifically the replication group size, that minimizes the overhead of index replication and query propagation. When n is known, the subinterval length with expected number of peers equal to d is given by d/n. Let d be the group size, and q be the number of groups in the network. The total number of messages to replicate and query an object, $M = q + d$, is given by

$$M = \frac{n}{d} + d$$

It is easy to see that d equals to $\sqrt{n}$ minimizes M. When query and data messages have different sizes, for instance b_1 and b_2 bytes respectively, the value of d that minimizes the total byte overhead is given by $\sqrt{\frac{b_1}{b_2}n}$. Figure 1 illustrates how peers are organized in replication groups in the interval $[0, 1]$.

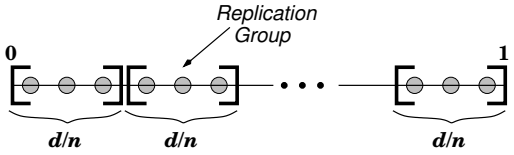


Figure 1: Replication groups in SplitQuest.

Groups in SplitQuest are numbered from 1 to $\lceil \frac{n}{d} \rceil$, and a peer's group can be computed by $\lceil \frac{ID \times n}{d} \rceil$, where ID is the peer's identifier in the interval $[0, 1]$. To install a reference to an object, a peer i sends installation messages to its immediate neighbors, peers with IDs smaller and greater than i, if they belong to the same group. The immediate neighbors of i check and forward the received messages if their neighbors are also in the same group, messages are not forwarded to peers already visited. The process is repeated until all peers in the group receive a copy of the reference.

2.2 Search Algorithm

Peers in SplitQuest replicate their contents in other peers in the same group. As a result, queries related to the content of a group can be answered by any group member.

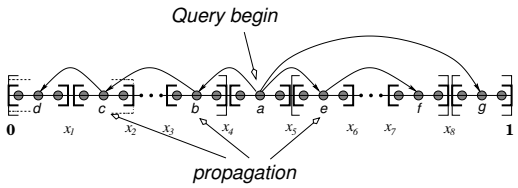


Figure 2: Query propagation in SplitQuest.

.

This replication strategy greatly simplifies query propagation, inasmuch as a query will be answered with certainty, positively or negatively, if it reaches any peer in each group. Therefore, the search algorithm must ensure that each group is visited at least once, and preferably no more than once. To guarantee that each group is visited only once, SplitQuest uses two different artifacts: search by interval and group shortcuts.

Search by interval means that each query message carries an interval $[X, Y]$ the query has to cover. When a peer i receives a message with the interval $[X, Y]$, it inspects its connections and determines which neighbors are within the received interval. These neighbors are possible candidates for query forwarding. Next, peer i computes the group IDs of the possible candidates and keeps only one peer for each different group, the choice to which peer to keep is random. Depending on the selected groups, peer i breaks down the initial interval and sends subintervals to each one of the selected peers. Figure 2 shows a simple example of a query propagation, only a small set of peer connections are shown. The query starts at peer a that is connected to peers b, e, and g and that has to cover the whole $[0, 1]$ interval. Peer a builds three messages with intervals $[0, x_4]$, $[x_5, x_8]$, and $[x_8, 1]$ and sends them to peers b, e, and g respectively. Peer b sends a message to c with interval $[0, x_2]$, and peer e sends a message to f with interval $[x_6, x_8]$. Finally, peer c sends a message to d with interval $[0, x_1]$, and peers b and f have to cover the remaining subintervals ($[x_2, x_3]$ and $[x_6, x_7]$).

In the example above, peer a has only three connections. If the number of connections were greater, the query could have been propagated to many other intervals, and, consequently, the query response would be faster. It is important to emphasize that the other peers in the figure may have several other random connections, we chose to show only the connections used to propagate the query. The hidden connections could be to other peers in intervals already covered in the current query, to groups not shown, or to peers in the same groups as the peers shown in the figure. Moreover, the split process always tries to build subintervals with the same sizes. This split process avoids long paths caused by uneven distributions and, consequently, speeds up query resolution.

Another artifact used by SplitQuest to improve query performance is group shortcuts. Peers build shortcuts (direct connection to other peers) to the two adjacent groups to the group to which they belong. These shortcuts guarantee that there is always a group path, i.e., a path in which a new hop covers a new group, between two different groups, and avoid that a query bounce several times inside a group. A peer with ID w can build shortcuts to adjacent groups by connecting to peers with IDs $w + 1/d$ and $w - 1/d$, or to peers with IDs closest to these values if the peers with the exact computed IDs are not present in the network. In these cases, the IDs must be appropriately fixed.

2.3 Analysis

In SplitQuest, groups are visited only once. If we consider the group connections, built as random connections at the peers visited during a query, we can model query propagation to the different groups as a broadcast in a random tree. At each step of the algorithm, the groups are partitioned based on a peer's connection and on the interval it has to cover.

Theorem 1. *Any query in SplitQuest can be solved in* $O(\log n)$ *steps.*

Proof. Consider G as the number of groups to be visited. Initially, G is equal to the total number of groups. Each g_k represents the cardinality of a set of groups that covers non overlapping intervals. When a query starts at a peer i, it has to visit a set of G disjoint groups. This set is randomly split into R subsets, which depend on the connections of i, with distribution given by $P(R = r) = p_r$, where p_r is a probability distribution on $\{1, 2, \ldots\}$. Conditionally on the event $\{R = r\}$, for $1 \leq k \leq r$, a group is in the k-th subset with probability $V_{k,r}$, where $V_r = (V_{k,r}; 1 \leq k \leq r)$ is a random probability vector on $\{1, \ldots, r\}$. If N_k is the cardinality of the k-th subset, then, conditionally on the event $\{R = r\}$ and on the random variables $V_{1,r}, \ldots, V_{r,r}$, the distribution of the vector $(N_1, \ldots, N_r)$ is multinomial with parameters $G-1$ and $(V_{1,r}, \ldots, V_{r,r})$ [4]. The analysis of the height (H_G) of the induced random tree for a general distribution of R is very intricate and is generally obtained by complex analysis techniques [4]. When r is in $\{1, 2\}$, which means that there are at most two branches at each broadcast step, we can obtain a more conservative upper bound. Devroye [2] shows that when r is in $\{1, 2\}$ the height of the tree is bounded by $O(\log G)$, more specifically Devroye [2] shows that

$$\lim_{G \to \infty} P\{H_G > c \log G\} = 0$$

where c is a constant. A stronger result is derived in [1], where it is shown that $H_G / \log G \to 4.31106\ldots$ in

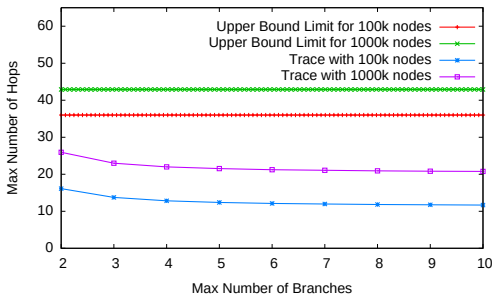


Figure 3: Maximum distance in hops a query travels from the starting peer in SplitQuest for different branch factors. The top lines are the theoretical bounds for 1000k and 100k peers respectively.

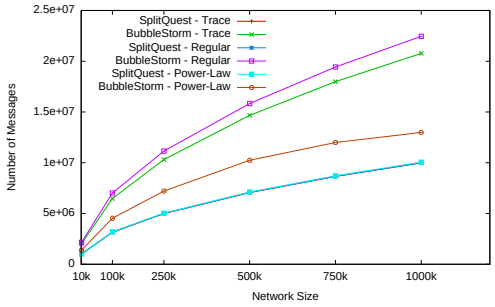


Figure 4: Number of messages with churn.

probability. Due to space limitation, we refer the reader to [1] for more details on the analysis. □

It is easy to see the above derived upper bound is also an upper bound for the farthest (number of hops) a query travels in SplitQuest. Since SplitQuest can forward a query to more than two groups, the upper bound is, in fact, very conservative. Figure 3 shows results to the maximum number of hops a query travels from the starting peer in networks with 100k and 1000k peers and when the maximum number of branches is increased from 2 to 10.

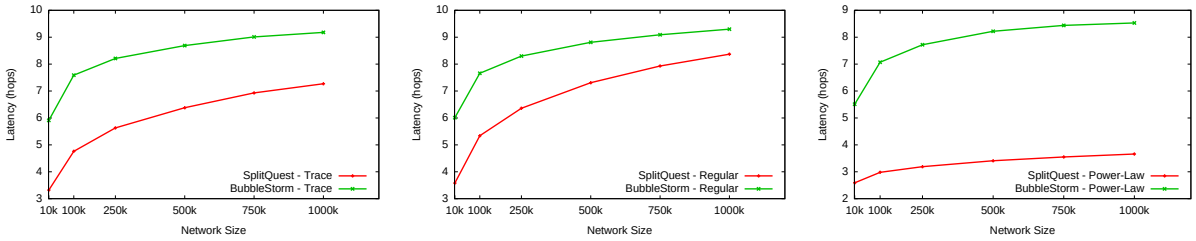
3 Preliminary Results

In this section, we present numerical results for the comparisons between SplitQuest and Bubblestorm. We show results for three metrics (number of messages, success rate and latency) under three different topologies (trace [7], regular and power-law) and under different churn scenarios (static and churn). In all figures below, the x-axis represents different network sizes and the y-axis represents a specific measure computed as the average of 11 independent runs. To be fair in our comparison, we borrowed most of the simulation parameters from [8], including churn rates and network sizes.

In Figure 4, we show results for the number of messages when 5000 articles are inserted in the network. For each article, a query is started 100 seconds after the article is inserted. The total number of messages presented is the sum of data and query messages during the whole simulation time, we observed similar behavior for the total traffic. We can see that, in all three topologies and network sizes, the number of messages for SplitQuest is almost the same. We can in fact see the three lines as the single line at the bottom of the figure. This behavior is easy to understand, since SplitQuest uses only the network size as its parameter to determine the number of replicas and query messages. An important conclusion we can infer from the figure is that the number of messages SplitQuest generates is independent of topology (at least for the three cases we studied) and independent of particular parameters of peers, such as degree distribution (as BubbleStorm which uses a protocol based on degree distribution and on a certainty factor). We can also observe in Figure 4 that SplitQuest outperforms BubbleStorm in all three different topologies. The gains vary from 50% up to 52% in the regular topology, from 53.5% up to 55.5% in the trace topology, and from 22.5% up to 30.5% in the power-law topology.

The success rate in a dynamic scenario (with churn) can be observed in Figure 5. We observe that, for SplitQuest, the success rate is over 99.8%, which represents a decrease of approximately 0.17% in comparison with the static scenario (not shown and which yielded 100% of success rate). This decrease can be explained by peers leaving the network while they are still processing data or query messages. A peer that receives a data replication message has to distribute the data to other peers in the group. If the peer leaves, peers not yet covered may not receive the message. Furthermore, a peer that leaves the network may not forward a query message to other peers in the interval it has to cover. Even though BubbleStorm replicates more data and query messages than SplitQuest, it does not achieve success rate higher than 98.5%. This result is mainly due to the blind search method it uses that does not explore all possible peers (the success rate in BubbleStorm can be increased, but with an additional increase in query or data messages). The results also show that SplitQuest is resilient to peer joins and departures, and that it keeps high success rate even under churn.

One of the most important characteristic of BubbleStorm is the low latency in query resolution, it improved significantly the response times for exhaustive search in comparison with the first approach based on sequential random walks [3]. SplitQuest is able to achieve even lower times, as can be seen in Figure 6. This



(a) Trace (b) Regular (c) Power-law

Figure 6: First-match latencies in hops for the three different topologies and under churn.

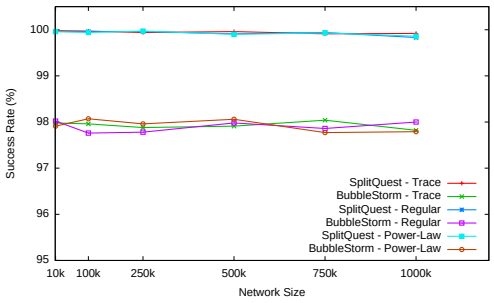


Figure 5: Success rates with peer joins and departures.

reduction in latency is possible because SplitQuest aggressively exploits the heterogeneity of peers and spread query messages much more quickly than BubbleStorm. High-degree peers are able to send a query message to several different groups in one single step. Moreover, the directed search avoids loops and rapidly cover all the groups in the network. The ability to explore the heterogeneity of peers is more evident when we compare the latencies in the regular topology with the latencies in the power-law topology. Since the power-law topology contains peers with up to 800 connections, SplitQuest is able to solve a query with up to 4 fewer hops than in the regular case.

For all topologies and network sizes, SplitQuest presents better results in terms of times and hops than BubbleStorm. The best improvement is observed in the power-law topology, in which the latency is reduced by approximately 59%. The worst case happens in the regular topology, but SplitQuest still outperforms BubbleStorm by approximately 11%.

4 Conclusion and Future Work

An initial implementation of SplitQuest appears promising, but there are still several questions we need to explore. Our future work includes a more detailed analysis of SplitQuest's replication strategy. We plan to extend our replication strategy and analyze the impact of replicating data in more groups in the network and also in groups of different sizes. The metrics evaluated in this work will most likely be affected, as higher number of replicas should offer more options for search messages to be spread in the network. Moreover, object popularity should offer another important dimension to be explored, since popular objects do not need to be replicated at the same rate as rare objects.

Acknowledgment

This research has been partially funded by CNPq (Proc. 483929/2007-7 and Proc. 304974/2008-0) and Fundect (Proc. 23/200.134/2008).

References

[1] DEVROYE, L. A Note on the Height of Binary Search Trees. *Journal of ACM 33*, 3 (1986), 489–498.

[2] DEVROYE, L. Universal Limit Laws for Depths in Random Trees. *SIAM Journal on Computing 28*, 2 (1998), 409–432.

[3] FERREIRA, R. A., RAMANATHAN, M. K., AWAN, A., GRAMA, A., AND JAGANNATHAN, S. Search with Probabilistic Guarantees in Unstructured Peer-to-Peer Networks. In *P2P 2005* (Konstanz, Germany, August 2005), pp. 165–172.

[4] MOHAMED, H., AND ROBERT, P. A Probabilistic Analysis of Some Tree Algorithms. *The Annals of Applied Probability 15*, 4 (2005), 2445–2471.

[5] RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., AND SHENKER, S. A Scalable Content-Addressable Network. In *ACM SIGCOMM 2001* (San Diego, CA, August 2001), pp. 247–254.

[6] STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, F., AND BALAKRISHNAN, H. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *ACM SIGCOMM 2001* (San Diego, CA, August 2001), pp. 149–160.

[7] STUTZBACH, D., REJAIE, R., AND SEN, S. Characterizing Unstructured Overlay Topologies in Modern P2P File-Sharing Systems. *IEEE/ACM Transactions on Networks (*Eletronic Edition*) 16*, 2 (April 2008).

[8] TERPSTRA, W. W., KANGASHARJU, J., LENG, C., AND BUCHMANN, A. P. BubbleStorm: Resilient, Probabilistic, and Exhaustive Peer-to-Peer Search. In *ACM SIGCOMM 2007* (Tokyo, Japan, August 2007), pp. 49–60.