University of Paderborn

Computer Networks Group

Extended Results on an Adaptive Resource/Performance Trade-Off for Resolving Complex Queries in P2P Networks

Thorsten Biermann, Christian Dannewitz, and Holger Karl

{thorsten.biermann|christian.dannewitz|holger.karl}@upb.de

October 2008

Technical Report TR-RI-08-294

Technical report

Abstract

Structured Peer-to-Peer (P2P) systems are increasingly important for scalable data dissemination and search. Current distributed approaches for resolving complex search queries, like multi-attribute and range queries, typically require multiple query messages to resolve a single search request. To reduce the message overhead and the search latency, some approaches like the Multi-Attribute Addressable Network (MAAN) use static replication. However, this results in high main memory requirements and large data transfers each time a device joins the P2P network. Those drawbacks can be tolerated for P2P networks that mainly consist of fixed, powerful nodes like PCs but are intolerable for resource-constrained nodes with high churn, like mobile devices. As mobile devices will play a significant role in accessing and distributing data in the future, we propose and evaluate an improved search mechanism for such a scenario. Compared to MAAN, our approach significantly reduces the memory footprint and bandwidth requirements (up to a factor of five in our sample scenario). At the same time, the good latency properties of MAAN are remained on average. This is achieved via a dynamic replication scheme which introduces an adjustable trade-off between memory footprint and search latency. Thereby, our approach makes efficient, distributed resolution of complex queries in resource-constrained P2P networks feasible.

1 Introduction

Peer-to-Peer (P2P) networks have become popular for data dissemination as they scale well and are self-configuring. At the same time, mobile devices like cellular phones and PDAs play an increasing role with respect to searching, accessing, and distributing content. Hence, mobile devices should be able to participate in P2P networks to enable users to search and access content anytime and anywhere.

In this paper, we focus on distributed search mechanisms in structured P2P networks in which mobile devices participate. In such a network, the user should be able to search for content based on attached attributes/metadata. We call the set of attributes accompanying content its *Information Object (IO).* The attribute values can change frequently and the system should enable complex queries on those attributes, including multi-attribute and range queries.

Mobile devices have some special properties that prevent them to be handled like a normal desktop computer in a P2P network. First of all, mobile devices have limited resources with respect to energy, memory, bandwidth, and processing power. Moreover, mobile devices produce high churn as they are typically used in a mobile, dynamic environment and only for short periods of time.

In such a scenario, realizing a fast distributed search efficiently is problematic. Existing solutions for distributed search primarily focus on scenarios with fixed, powerful nodes and, therefore, have one or multiple of the following characteristics that make them unsuitable for mobile devices:

- High memory requirements for each participating node
- Attribute updates generate a lot of traffic
- Joining/leaving nodes require large data transfers
- Multiple query messages are required for a single range/multi-attribute query, resulting in high traffic

Therefore, we developed a distributed search mechanism which is optimized for a mobile-devices scenario, i.e., it has a low memory footprint, reduces the traffic load generated by updates and joins/leaves, and reduces the number of required query messages. We achieve those goals via a dynamic replication mechanism (Section 3) that provides an adjustable trade-off between memory footprint and number of query messages generated per search request. The number of query messages directly correlates with the query latency in our system. Hence, we optimize latency and messaging overhead simultaneously. In the following, reducing the number of messages is implicitly included when we talk about reduced latency.

One would expect that the reduced resource consumption is paid for by a higher search latency compared to existing systems; however, our simulation results (Section 4) show that our approach can actually outperform other systems on both fronts at the same time. It supports *single-attribute searches*, *multi-attribute searches*, as well as *range queries.*

Although we use Chord [1] as underlying routing scheme in the course of this paper, our approach is not limited to Chord but can be applied to any distributed search mechanism where search results are combined from multiple nodes.

2 Related Work

This section will first describe related work and will then discuss the basics of MAAN [2] upon which our system builds.

2.1 Complex distributed query resolution

Today's P2P systems offer only very limited support for advanced, distributed search. While some systems, like BitTorrent or eDonkey2000, pursue a hybrid approach where search requests are processed on dedicated servers, other popular systems, like Kademlia, only support basic keyword-based search. Hence, multiple approaches [3, 4] have been developed that support distributed query resolution with better expressiveness. Solutions supporting range queries can be divided into two major groups: First, schemes based on space-partitioning schemes, either via locality-preserving hash functions [2, 5] or hypercuboids [6]. Second, schemes based on space-filling curves [7, 8]. All solutions generate considerable overhead by requiring multiple query messages to resolve a single range query, which limits their usability for mobile devices.

To enable multi-attribute queries and multi-dimensional range queries, current approaches use three fundamentally different solutions: Some approaches use multiple search structures like DHTs [7] or so called *hubs* [5], resulting in multiple query messages for each search structure and, hence, in additional overhead for the P2P network. Other approaches use a single search structure to manage all attributes, using a *fixed* multi-dimensional space [6, 8]. Leaving one or more dimensions undefined in a query again results in an increased number of additional query messages. Hence, some approaches like MAAN store several attribute replicas to reduce the number of query messages in exchange for higher memory requirements on the P2P nodes, which also limits the usability for resource-constrained devices.

To support complex queries on resource-constrained devices, we adjust and extent the MAAN approach. We add a dynamic replication scheme that introduces an adjustable trade-off between memory footprint and search latency. None of the other approaches offers such an adjustable trade-off. Thereby, we can reduce the latency of popular range queries and multi-attribute queries while significantly reducing the memory and bandwidth requirements compared to MAAN.

2.2 Multi-Attribute Addressable Network (MAAN)

MAAN enables searching for IOs based on their meta information. It uses the circular, structured P2P network Chord [1] as routing infrastructure and supports single-attribute, multi-attribute, and range queries. The processes of distributing data and resolving queries are described in the next sections.

2.2.1 Data distribution

Distributing attributes which do not require range queries is straightforward. Assuming that an IO consists of n_{A} attribute/value pairs (a_i, v_i), for each attribute value v_i the hash value $H_{\mathrm{con}}(v_i)$ is calculated using a consistent hash function. Such a value determines the next node clockwise in the P2P ring, *successor($H_{\mathrm{con}}(v_i)$)*, that is responsible for the attribute value v_i. This node receives a copy of all triples *(a_i, v_i, IO identifier)* describing the content of an IO. Hence, each attribute/value pair of an IO is stored n_{A} times within the system. The data structure which holds all these triples at each node is called *Search Table* in the following; the triples are called *Search Table Entries (STEs)*.

To give a rough idea about MAAN's memory requirements, we exemplarily calculate the mean Search Table size of each node in such a system. We assume an IO identifier length of 160 bits, an average attribute name size of 20 bytes, and an average attribute value size of 40 bytes. Furthermore, each IO contains $n_{\mathrm{A}} = 25$ attribute/value pairs which, e.g., corresponds to the ID3 tags and file properties of an MP3 file. Using a network of 2500 nodes to provide distributed search for 2.5 million IOs results in a Search Table size of 50 MB. This data has to be transferred on every node join and leave event.

2.2.2 Single- and multi-attribute queries

MAAN is able to resolve single-attribute queries at a single node. The required steps for processing multi-attribute queries depend on the logical operator linking the subqueries. OR-linked queries require input from up to $n_{\mathrm{A,Q}}$ nodes, where $n_{\mathrm{A,Q}}$ is the number of different attribute values within the query. AND-linked queries can be processed at a single node owing to all IO attributes being redundantly stored at each node.

2.2.3 Range attributes/queries

Range attributes have values that represent a whole interval of numerical values. To support such attributes and to also support searching for value ranges, *locality-preserving* hash functions are used. Such functions map attribute values to keys which have the same ordering as the input attribute values. If an IO contains a range attribute, all n_{A} attribute/value pairs are distributed to all nodes in the ring that cover this range. This number of nodes heavily depends on the total number of participating nodes in the ring and the size of the attribute value range. Both can become very large.

3 System Architecture

In the following, we will describe our architecture: First, Section 3.1 describes the basic search mechanism. It supports single-attribute, multi-attribute, and range queries with much smaller memory footprint compared to MAAN. On the downside, it results in increased latency for multi-attribute queries. To remedy this shortcoming, we introduce a dynamic replication strategy in Section 3.2 that reduces latency and offers a trade-off between latency and memory usage. Finally, Section 3.3 discusses the consequences of churn on the proposed techniques.

3.1 Basic search mechanism

The fundamental difference between MAAN and our approach is the way STEs are stored. MAAN stores the attribute/value pairs redundantly in the system, as described in Section 2.2.1. In our system, each attribute/value pair (a_i, v_i) is stored only once, i.e., only on the node *successor($H_{\mathrm{con}}(v_i)$)* which is responsible for this attribute value.

This approach has three main advantages: First, it significantly reduces the Search Table size of each participating node. Revisiting the example from Section 2.2.1, the same Search Table that requires 50 MB in MAAN would only require 2 MB in our system – reduced by a factor of 25, equaling the number of attributes per IO (n_{A}). Second, as the full Search Table has to be transferred to any new, joining node, the reduced memory footprint also results in significantly reduced data transfer for join operations. And third, changing an attribute value requires only one update on a single node, while MAAN has to update all $n_{\mathrm{A}} = 25$ replicas.

For single-attribute queries, the reduction in memory footprint comes without any cost, i.e., latency does not increase compared to MAAN and we can use the same query resolution algorithm: The hash value $H_{\mathrm{con}}(v_i)$ of the searched attribute value v_i is calculated and the query is routed to the node *successor($H_{\mathrm{con}}(v_i)$)* which can immediately answer the query. Range queries for a single attribute can also be processed in the same way as by MAAN without affecting latency.

OR-linked multi-attribute queries can also be resolved in the same way as by MAAN by querying all responsible nodes (i.e., responsible for one of the attribute values) in parallel and calculating the set union at the requesting node. For AND-linked queries, however, the reduced memory footprint causes an increased search latency for our basic search mechanism. While MAAN can answer AND-linked multi-attribute queries at a single node, our basic search has to visit all nodes responsible for a subquery one after the other. The same applies to multi-attribute *range* queries.

Figure 1 illustrates the AND-linked multi-attribute query `artist='Singer 1' AND song='My Song'`. First, the query message (1) is sent to node 11011 which can resolve the subquery `artist='Singer 1'`. The (remaining) query, including the preliminary search results, is forwarded (2) to node 10011, which can answer the subquery `song='My Song'`. This node calculates the intersection of the previous node's results with its own results and returns the final query result (3) to the requesting node 00011.

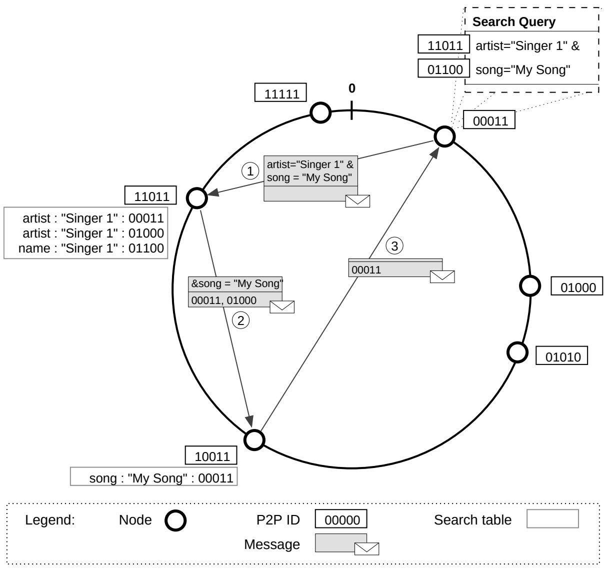


Figure 1: Resolving the multi-attribute query `artist='Singer 1' AND song='My Song'` from originating node *00011*.

Our approach is especially effective when storing range attributes in the P2P network. Range attributes have to be stored on all nodes falling into the value range. This is true for MAAN as well as for our system. In contrast to MAAN, however, we only store the single relevant STE on each of those nodes, whereas MAAN stores all attribute/value pairs of an IO on each node within the range. While this gives MAAN the same previously described latency advantages also when processing multi-attribute *range* queries, the redundant attribute storage results in large memory requirements. Hence, for range attributes, our approach achieves even higher *absolute* memory savings.

3.2 Dynamic replication

As mentioned in the previous section, the downside of our approach is an increased latency for processing multi-attribute queries. To overcome this drawback, we present an optimization scheme that dynamically replicates frequently requested STEs. These replicas only result in redundancy where useful, i.e., where they cause a reduced search latency. STEs can be replicated multiple times in sequence. This ensures that popular multi-attribute and range queries can finally be answered by a single node. This dynamic replication scheme is not limited to MAAN and can be applied to any distributed search mechanism where search results are combined of intermediate results from multiple nodes.

An intuitive example for search requests that can be optimized are queries for geographic coordinates. They consist of the attributes `geoLat` and `geoLong` and often ask for the same value combination when querying popular IOs. Such queries would require contacting two separate nodes. The proposed replication scheme, however, copies popular STEs which causes both search nodes (for latitude and longitude) to be able to answer the complete query at once.

The base for the replication decision are access statistics. They are collected for each STE during normal query processing. Therefore, in addition to the Search Table, each node holds an *Access Table* where the following information is stored:

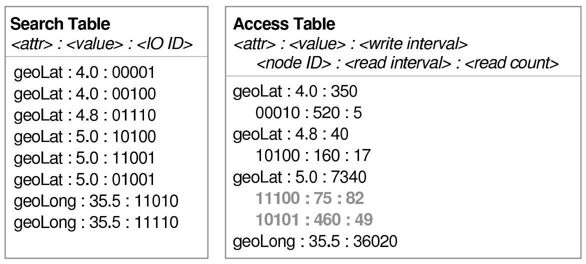
1. Mean inter-read time $\overline{t_{\mathrm{R},p,N}}$ per attribute/value pair p and accessing node N; this counter is reset to zero when a replica is sent out to N.

2. Mean inter-write time $\overline{t_{\mathrm{W},p}}$ per attribute/value pair p.

We propose to use an Exponential Weighted Moving Average (EWMA) for calculating both mean inter-access times. It can be implemented easily and has the advantage of absorbing intermittent request peaks while still reacting quickly to lasting changes in access frequencies. The agility can be adjusted via the smoothing factor α_{MA} used for EWMA calculation.

Every node N which requests an attribute/value pair p more frequently than it is updated (i.e., $\overline{t_{\mathrm{R},p,N}} < \overline{t_{\mathrm{W},p}}$) is potentially suitable for receiving a replica of this pair. Otherwise, replicas would require more effort to perform updates than could be saved by reduced requests.

Figure 2 shows a sample Search Table with a corresponding Access Table. The



Search Table
<attr> : <value> : <IO ID>

geoLat : 4.0 : 00001
geoLat : 4.0 : 00100
geoLat : 4.8 : 01110
geoLat : 5.0 : 10100
geoLat : 5.0 : 11001
geoLat : 5.0 : 01001
geoLong : 35.5 : 11010
geoLong : 35.5 : 11110

Access Table
<attr> : <value> : <write interval>
<node ID> : <read interval> : <read count>

geoLat : 4.0 : 350
00010 : 520 : 5
geoLat : 4.8 : 40
10100 : 160 : 17
geoLat : 5.0 : 7340
11100 : 75 : 82
10101 : 460 : 49
geoLong : 35.5 : 36020

Figure 2: Exemplary Search Table and corresponding Access Table. The attribute/value combination `geoLat=5.0` has two potential replication nodes, *11100* and *10101*, marked in gray.

Search Table contains 8 entries for which the node is responsible. On each search request for a given attribute/value pair, the inter-read time for the requesting node is updated. The same holds for the inter-write times for each write operation.

Two conditions have to be fulfilled prior to a replication. These conditions can be influenced via two parameters individually at each node:

$T_{\mathrm{R,t}}$ This threshold determines a STE's minimum required popularity in terms of inter-read time for being replicated. A replica is only created if $\overline{t_{\mathrm{R},p,N}} < T_{\mathrm{R,t}}$ is fulfilled. If $T_{\mathrm{R,t}}$ is small, only very popular entries are replicated; large values also permit replication of unpopular entries. This parameter determines the system's steady state, i.e., the required average memory and the achieved average search latency after the system has finished its initialization phase.

$T_{\mathrm{R,c}}$ This threshold defines the minimum required number of read operations before a replica is created. Hence, the number of requests $c_{\mathrm{R},p,N}$ by a node N since the previous replication must exceed the threshold $T_{\mathrm{R,c}}$. By varying $T_{\mathrm{R,c}}$, the speed of the replication process can be influenced. Low values foster a quick replication whereas high values prevent this.

To prevent old replicas which do not improve the search performance anymore, a third parameter is used by the search system. This parameter influences the replica lifetime:

α_{TTL} A replica is deleted from the Search Table after a time $t_{\mathrm{D},p}$ without read access. This time is calculated for each replica as the product $\overline{t_{\mathrm{R},p}} \cdot \alpha_{\mathrm{TTL}}$, where $\overline{t_{\mathrm{R},p}}$ is the mean inter-read time calculated before the replica was received and $\alpha_{\mathrm{TTL}} > 1$ is a configuration factor. The smaller α_{TTL} the earlier replicas deleted from the Search Table again when they become unpopular.

Updates to STEs have to be propagated to all nodes which currently hold a replica of this entry. This is done by forwarding the update from the primarily responsible node to all nodes which received a replica. To avoid reading outdated replicas, the ratio of inter-write and inter-read time $\frac{\overline{t_{\mathrm{W},p}}}{\overline{t_{\mathrm{R},p,N}}}$ should be kept small by limiting replication to cases where $\overline{t_{\mathrm{R},p,N}} \ll T_{\mathrm{R,t}}$.

Compared to the basic search mechanism, the described replication strategy increases the memory footprint but decreases the search latency. This introduces a powerful trade-off which permits to adapt the search system to the needs of many scenarios. A short guideline how to adjust the parameters $T_{\mathrm{R,t}}$, $T_{\mathrm{R,c}}$, and α_{TTL} to achieve the desired search latency and memory requirements will be given in Section 4.

The proposed replication and the use of locality-preserving hashing can cause unequal load distribution among P2P nodes. This problem can be circumvented by using virtual nodes [9] to balance load.

3.3 Handling churn

When users join or leave the P2P search system, STEs have to be migrated to joining nodes or from leaving nodes to neighbors. As leave events often do not occur in an announced manner, STEs are lost. To lower the loss probability, the underlying DHT can store entries not only on the responsible node but also create *k backup copies* on subsequent nodes in the ring. This static backup scheme is orthogonal to our dynamic replication method for popular STEs.

The question arises whether dynamically generated replicas should also be backed up. Creating only backups of primary STEs results in reduced memory requirements but raises the search latency on unheralded node leaves until a new replica is spawned again. To ensure a constantly low search latency, dynamically replicated entries have to be copied, too.

The major drawback of the backup scheme is the increase in memory footprint. However, compared to using this technique in MAAN, where it is required for the same reasons, the memory and data transfer increase of our approach is again lessened by the factor n_{A}. To further reduce the memory overhead, the number of backup copies k has to be reduced. Without increasing the loss probability, however, this is only possible by reducing the churn itself. This can be achieved by excluding mobile nodes from the P2P ring. To still provide these nodes with the ability to perform search requests, *proxy nodes* [10] can be introduced which perform the actual query resolution in the P2P ring for them. Figure 3 gives a sample topology showing this proxy functionality.

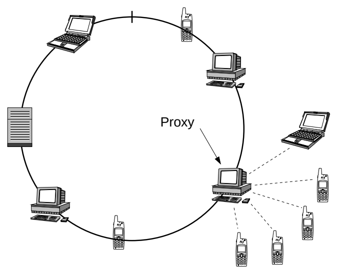


Figure 3: Sample search infrastructure: One *peer node* acts as *proxy* for highly mobile nodes. Nodes connected to a proxy are called *clients*.

Using proxies has additional advantages. First, it lowers memory and bandwidth requirements for users searching via proxies and, second, those users' search latency is decreased. This results from popular STEs being quickly replicated onto proxies as their search load is higher than for nodes only serving a single user. To handle the higher load, dedicated proxy nodes might be reasonable, e.g., installed by mobile ISPs, to specifically increase performance for customers.

4 Evaluation

The proposed search system is evaluated via simulation. Simulation assumptions are discussed in Section 4.1. Results are presented in the subsequent sections.

4.1 System model

All simulation runs are conducted using the discrete event simulator OMNeT++ and the two frameworks INET and OverSim. The topology used for the experiments consists of 4 core routers which are fully interconnected. Each of them provides connectivity to 3 access routers. All routers are connected via 10 Gbit/s links with a latency of 2 ms. Peer and client nodes are randomly assigned to the 12 access routers in equal shares and have 6 Mbit/s links with 35 ms latency. This corresponds to usual DSL connections.

The load model consists of the following components: the number of peers forming the search infrastructure, the number of clients using this infrastructure (via proxy peers), the number of IOs which can be searched for, and the clients' query behavior. We performed several experiments with different numbers of peer nodes. These nodes are sequentially added to the network at the beginning of each simulation run. Each of these nodes, again, is assigned a number of clients that pose queries to the system via this peer. The client number per peer node is varied for different experiments, too. The total amount (and hence the total query load), however, is kept constant at 2500 clients. Table 1 shows the parameters of the evaluated scenarios. In the first

Table 1: Evaluated proxy scenarios

Scenario	P(proxy)	μ_{clients}	σ_{clients}
Few dedicated peers are proxies	0.05	1000	500
All peers are proxies	1	10	5
No proxies	0	1	0

one, 5 % of the peer nodes act as proxies, whereas in the second one, all peer nodes are used as proxies by the clients. The third one does not use proxy nodes at all. The parameters μ_{clients} and σ_{clients} denote the mean number of clients per peer node and its standard deviation. The number of clients is normal distributed. P(proxy) denotes the probability of a peer node being a proxy.

After Chord has finished constructing its routing information, every peer node adds 30 IOs to the system which are not modified during the simulation; each IO contains 5 keyword attributes. Keywords are selected with Zipf-distributed popularity with parameter 1.01 [11] from a pool of 100 or 1000 keywords.

After all IOs have been added, clients are triggered to start posing queries. Each query contains two AND-linked keyword subqueries. The queried keywords are chosen from the keyword pool and are also selected with Zipf-distributed popularity. Each client's inter-query time is Weibull-distributed with a mean of approximately 115 s [12]. This parametrization corresponds to a Web browsing user behavior.

As described in Section 3.3, backup copies of STEs (incl. replicas) are stored on k subsequent nodes. Such copies are immediately transferred to joining nodes to maintain the current replication state at this position in the ring. In case of failures, a leaving node's duties will be immediately taken over by the next node in the P2P ring. Hence, our dynamic replication mechanism is not affected as long as at least one of the k

backup copies remains. For this reason, we omit the evaluation of churn scenarios in this paper.

Two main metrics are observed during the simulation. The number of *hops per search request* and the number of stored *STEs per peer node*. The number of hops per search request is the number of nodes which are involved in resolving one single search request. This metric gives information about the latency for answering search queries independent of the used P2P overlay network and the routing overhead therein. The number of STEs indicates the amount of memory each peer node must provide and represents the second trade-off component.

As $T_{\mathrm{R,t}}$ is an absolute threshold, its value has to be adapted to the actual load in the system to observe influences on value changes. This is done by measuring the global mean inter-read time per peer node and per STE $\overline{t_{\mathrm{R}}}$. The reference value $\overline{t_{\mathrm{R}}}$ is multiplied with a constant factor β to calculate $T_{\mathrm{R,t}}$.

4.2 Results

The following figures illustrate the trade-off between search latency and memory footprint introduced by our search mechanism. Each subfigure shows one of the two metrics varying over the simulated time and additionally contains MAAN's performance for comparison. Confidence intervals have been calculated for a confidence level of 95 % but are omitted due to their small size for the sake of clarity.

Figure 4 points out the influence of the first replication threshold $T_{\mathrm{R,c}}$. The number

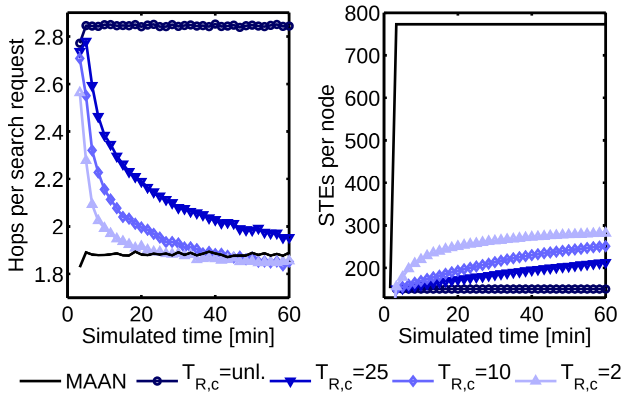


Figure 4: Hops per search request and STEs per peer node depending on $T_{\mathrm{R,c}}$; $\beta = 2$, $\alpha_{\mathrm{TTL}} = 10$, $P(\text{proxy}) = 0.05$, $C(\text{keywords}) = 1000$

of hops per search request is constant over time for MAAN and for our scheme with unlimited $T_{\mathrm{R,c}}$. While MAAN requires approx. 1.9 hops on average, our scheme without replication needs 2.85 hops. This corresponds to the expected 2 hops for MAAN (one to the proxy and one to the appropriate node) and 3 hops for our scheme (one additional to the second responsible node). The offset has two reasons: First, proxies can sometimes directly answer an attached client's search request when they are responsible for the queried attribute value themselves. Second, users being peers do not use proxies. In both cases, one hop is omitted. The other curves where replication is enabled nicely depict that small values for $T_{\mathrm{R,c}}$ accelerate the replication speed. The steady state, however, is unchanged. All three lines converge to a hop count slightly below MAAN (not completely shown). At the same time, the memory requirements of our scheme are still approx. 65 % lower compared to MAAN. Please note that this gain heavily depends on the average number of attributes n_{A} per IO and would even be higher in scenarios with $n_{\mathrm{A}} > 5$.

The influence of the parameter β, i.e., of the threshold $T_{\mathrm{R,t}}$, which is set on the replicating node to prevent replication of unpopular STEs, is shown in Figure 5. The

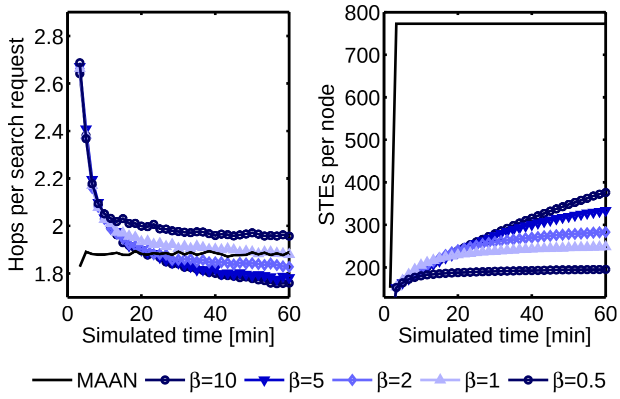


Figure 5: Hops per search request and STEs per peer node depending on β; $T_{\mathrm{R,c}} = 5$, $\alpha_{\mathrm{TTL}} = 10$, $P(\mathrm{proxy}) = 0.05$, $C(\mathrm{keywords}) = 1000$

figure nicely depicts the influence of $T_{\mathrm{R,t}}$ on the steady state latency and memory requirements. For all different $T_{\mathrm{R,t}}$ values, the steady state is reached at the same time. Corresponding to the expectations, large values for $T_{\mathrm{R,t}}$ lower the latency while raising the memory requirements as more STEs are replicated. Small values prevent unpopular keywords from being replicated. The measurements show that $T_{\mathrm{R,t}}$ enables nodes to adapt the resource/latency trade-off individually according to their device's abilities when they are not able to handle the query load they are exposed to.

Figure 6 shows the system behavior for various values of α_{TTL}. This parameter does not influence the active replication process but controls the lifetime of existing STE replicas. The parameter α_{TTL} influences the latency/memory trade-off on the replications' destination nodes in the same way as $T_{\mathrm{R,t}}$ on the replicating nodes' side. As expected, high values for α_{TTL} cause a low latency with high memory footprint and low values a high latency with low memory footprint. Hence, α_{TTL} gives also destination nodes of replicas the possibility to adjust the desired trade-off for themselves.

So far, we have shown our search technique's behavior for a proxy probability of $P(\mathrm{proxy}) = 0.05$, i.e., only a small fraction of all nodes act as proxies for nodes not participating in the P2P search structure. An alternative approach is to not use

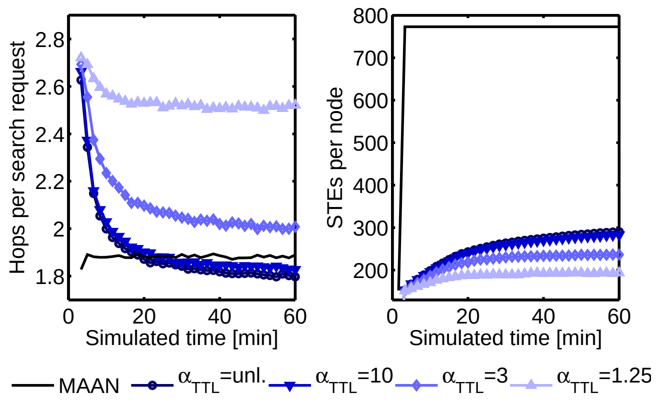


Figure 6: Hops per search request and STEs per peer node depending on α_{TTL}; $T_{\text{R,c}} = 5$, $\beta = 2$, $P(\text{proxy}) = 0.05$, $C(\text{keywords}) = 1000$

dedicated proxies but make all peer nodes perform proxying tasks. This causes the overall load to be spread among more nodes and, hence, to lower the load per individual proxy peer. The influence on the system's performance is shown in the following figures.

Figure 7 illustrates the influence of the replication threshold $T_{\text{R,c}}$. Compared to Figure 4, $P(\text{proxy})$ is changed from 0.05 to 1. The plot shows the expected behavior. As

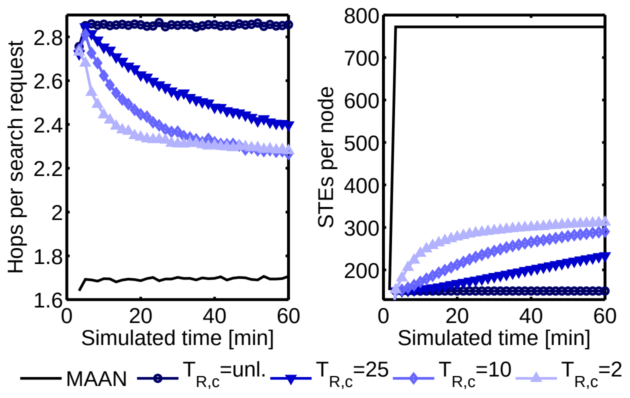


Figure 7: Hops per search request and STEs per peer node depending on $T_{\text{R,c}}$; $\beta = 2$, $\alpha_{\text{TTL}} = 10$, $P(\text{proxy}) = 1$, $C(\text{keywords}) = 1000$

the load is distributed among more proxy nodes, i.e., the load per proxy is lowered, less objects are replicated to each proxy compared to the scenario where $P(\text{proxy}) = 0.05$. On average, however, the number of IOs per node is nearly identical. This means that the number of available IO replicas at each proxy is noticeably lower than for the case where $P(\text{proxy}) = 0.05$. The result can clearly be seen at the required number of hops per search request. The performance decreases as less queries can be directly answered by proxies.

The lower hop count for MAAN results from the fact that now, where all peer nodes are proxies, more search requests can be directly answered by the proxies. The more peers are proxies, the higher the probability in the overall system to be able to answer requests directly at the first hop.

The counterpart to Figure 5 is given in Figure 8. It shows the two metrics depending on β, i.e., the replication threshold $T_{\mathrm{R,t}}$. The qualitative influence of the factor is

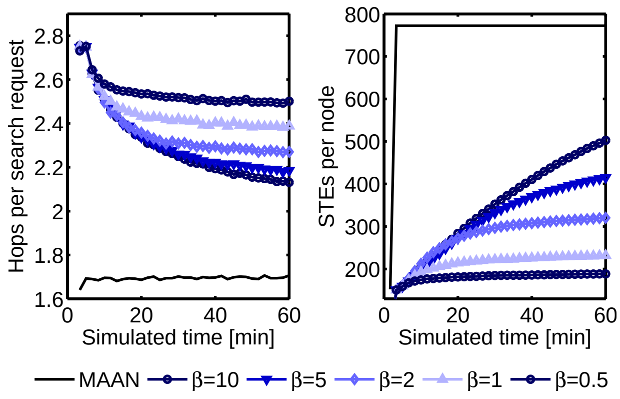


Figure 8: Hops per search request and STEs per peer node depending on β; $T_{\mathrm{R,c}} = 5$, $\alpha_{\mathrm{TTL}} = 10$, $P(\text{proxy}) = 1$, $C(\text{keywords}) = 1000$

identical in both scenarios. However, similar to the observations for $T_{\mathrm{R,c}}$, the achieved hop count at the same memory footprint is worse for $P(\text{proxy}) = 1$ than for 0.05. This behavior is caused by the same reasons that were alredy given for Figure 7.

A similar picture is given by Figure 9. It shows the measured system behavior for the configuration already examined in Figure 6 with a different proxy probability. The

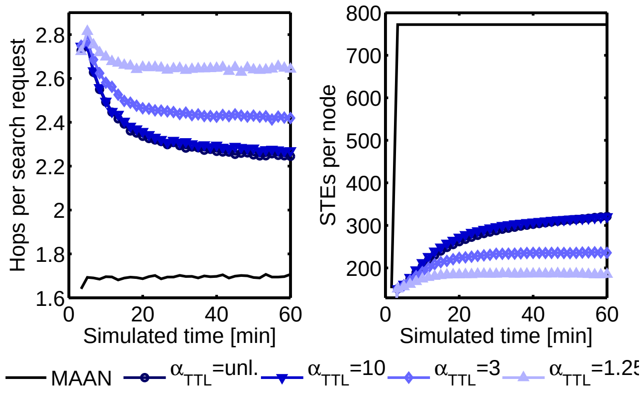


Figure 9: Hops per search request and STEs per peer node depending on α_{TTL}; $T_{\mathrm{R,c}} = 5$, $\beta = 2$, $P(\text{proxy}) = 1$, $C(\text{keywords}) = 1000$

plots confirm the behavior of α_{TTL} being able to adjust the required memory on the replication's destination side.

The last three figures (Figure 10, Figure 11, Figure 12) have the same parametrization as Figure 4, Figure 5, and Figure 6. The only difference is the pool size of keywords. Here, the pool size was reduced to 100 keywords instead of 1000 keywords. As the number of requests stays the same, this results in each keyword being requested more often and, hence, becoming more popular and being replicated more often. On the one hand, queries can, therefore, be answered in fewer hops, resulting in a significantly better performance compared to MAAN. On the other hand, the memory requirements also increase significantly, exceeding the MAAN memory requirements. This happens as STEs are replicated to many other nodes, including nodes that would not keep a copy of a certain STE in MAAN as they are not responsible for any of the included attributes. Nevertheless, those nodes send requests for the respective attribute and, hence, receive a replica in our scheme to improve latency. If the resulting memory requirements would be too high, memory needs and latency could be readjusted by choosing different parameter values for $T_{\mathrm{R,c}}$, β, and α_{TTL}.

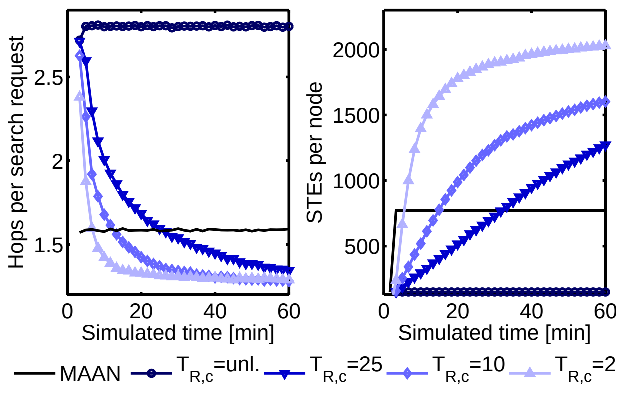


Figure 10: Hops per search request and STEs per peer node depending on $T_{\mathrm{R,c}}$; $\beta = 2$, $\alpha_{\mathrm{TTL}} = 10$, $P(\text{proxy}) = 0.05$, $C(\text{keywords}) = 100$

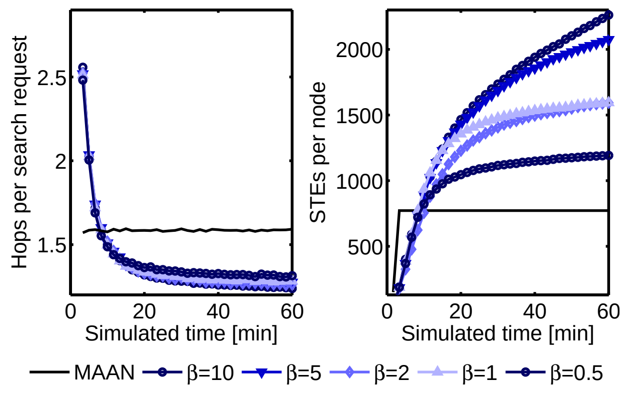


Figure 11: Hops per search request and STEs per peer node depending on β; $T_{\mathrm{R,c}} = 5$, $\alpha_{\mathrm{TTL}} = 10$, $P(\text{proxy}) = 0.05$, $C(\text{keywords}) = 100$

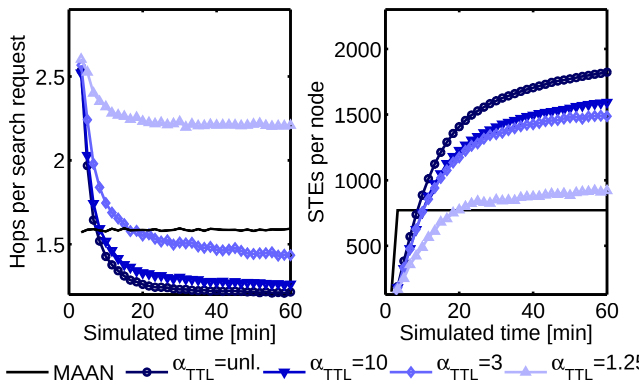


Figure 12: Hops per search request and STEs per peer node depending on α_{TTL}; $T_{\mathrm{R,c}} = 5$, $\beta = 2$, $P(\text{proxy}) = 0.05$, $C(\text{keywords}) = 100$

5 Conclusion

In this paper, we presented a distributed search mechanism that efficiently supports multi-attribute queries and multi-dimensional range queries on resource-constrained devices like PDAs or cellular phones. This is achieved by introducing an adjustable trade-off between search latency and resource requirements (memory, bandwidth). While we expected to pay the price for reduced resource requirements in terms of increased search latency compared to MAAN, simulations have shown that our system can even decrease it. Thereby, also in scenarios without resource constraints, users benefit from a low search latency and low message overhead.

Our simulations confirm the expected relation between chosen parametrization and resulting system behavior. Hence, adapting the scheme to the own needs is easily possible by adjusting the parameters according to their described influence.

In the future, we will evaluate the system's behavior to frequently changing attribute values and towards varying popularity of attribute values. Additionally, a closed-form representation of the relation between parametrization and resulting system properties would be useful.

Acknowledgment

We gratefully thank M. Dräxler for his implementation support and for the valuable discussions during our work.

References

[1] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *SIGCOMM '01: Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications.* New York, NY, USA: ACM, 2001, pp. 149–160.

[2] M. Cai, M. Frank, J. Chen, and P. Szekely, "MAAN: A multi-attribute addressable network for grid information services," in *GRID '03: Proceedings of the Fourth International Workshop on Grid Computing.* Washington, DC, USA: IEEE Computer Society, 2003, p. 184.

[3] J. Risson and T. Moors, "Survey of research towards robust peer-to-peer networks: search methods," *Comput. Netw.*, vol. 50, no. 17, pp. 3485–3521, 2006.

[4] R. Ranjan, A. Harwood, and R. Buyya, "A study on peer-to-peer based discovery of grid resource information," University of Melbourne, Australia, Technical Report GRIDS-TR-2006-17, Nov. 2006.

[5] A. R. Bharambe, M. Agrawal, and S. Seshan, "Mercury: Supporting scalable multi-attribute range queries," in *SIGCOMM '04: Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications.* New York, NY, USA: ACM Press, 2004, pp. 353–366.

[6] T. Schütt, F. Schintke, and A. Reinefeld, "Range queries on structured overlay networks," *Comput. Commun.*, vol. 31, no. 2, pp. 280–291, 2008.

[7] A. Andrzejak and Z. Xu, "Scalable, efficient range queries for grid information services," in *P2P '02: Proceedings of the Second International Conference on Peer-to-Peer Computing.* Washington, DC, USA: IEEE Computer Society, 2002, p. 33.

[8] C. Schmidt and M. Parashar, "Enabling flexible queries with guarantees in P2P systems," *IEEE Internet Comp.*, vol. 8, no. 3, pp. 19–26, 2004.

[9] J. Ledlie and M. Seltzer, "Distributed, secure load balancing with skew, heterogeneity and churn," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 2, Mar. 2005, pp. 1419–1430.

[10] T. Hsin-Ting Hu, B. Thai, and A. Seneviratne, "Supporting mobile devices in Gnutella file sharing network with mobile agents," *Proceedings of the 8th IEEE International Symposium on Computers and Communication (ISCC 2003)*, vol. 2, pp. 1035–1040, June 2003.

[11] K. Sripanidkulchai, “The popularity of gnutella queries and its implications on scalability,” 2001. [Online]. Available: http://www.cs.cmu.edu/~kunwadee/research/p2p/gnutella.html

[12] L. O. Walters, “A web browsing workload model for simulation,” Ph.D. dissertation, University of Cape Town, May 2004.