Complex Queries in P2P Networks with Resource-Constrained Devices

Christian Dannewitz, Thorsten Biermann, Martin Dräxler, and Holger Karl
University of Paderborn, Paderborn, Germany
{christian.dannewitz, thorsten.biermann, martin.draexler, holger.karl}@upb.de

***Abstract*— Structured Peer-to-Peer (P2P) systems are increasingly important for scalable data dissemination and search. At the same time, the importance of mobile devices like smartphones, PDAs, and netbooks for accessing and storing data is rapidly increasing.**

Current distributed approaches for resolving complex search queries, like multi-attribute and range queries, typically require multiple messages to resolve a single search request. This generates significant messaging overhead and increases the response latency. To reduce the messaging 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 present an improved search mechanism for distributed resolution of complex queries on resource-constrained devices.

Compared to MAAN, our approach significantly reduces the memory footprint and bandwidth requirements (up to a factor of three – five, depending on the load model and the type of query in our sample scenario). At the same time, the good latency properties and the low messaging overhead of MAAN are retained 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 on resource-constrained devices feasible, allowing such devices to live up to their important role in the future.

***Index Terms*— P2P, search, complex queries, multi-attribute queries, range queries, trade-off, resource-constrained devices, mobile devices**

I. 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 corresponding attributes/metadata. We call the set of attributes accompanying content its *Information Object (IO)*. The system should enable complex queries on these attributes, including multi-attribute queries, range queries, and a combination of both. The attribute values can change frequently and we especially include scenarios where the user is interested in always retrieving the latest results.

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

In such a scenario, realizing a distributed search with low latency efficiently is problematic. Existing solutions for distributed search primarily focus on scenarios with fixed, powerful nodes and, therefore, have one or several 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 have 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 with an adapted search architecture based on MAAN, combined with a new dynamic replication mechanism 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 reduce latency and messaging overhead simultaneously. In the following, reducing the number of

The research leading to these results has been conducted in the 4WARD project and received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 216041.

This paper is based on "An Adaptive Resource/Performance Trade-Off for Resolving Complex Queries in P2P Networks" by T. Biermann, C. Dannewitz, and H. Karl which appeared in the Proceedings of the IEEE International Conference on Communications (ICC), Dresden, Germany, June 2009. [1] © 2009 IEEE.

messages is implied when we talk about reduced latency and vice versa.

Our search mechanism supports *single-attribute searches*, *multi-attribute searches*, as well as *range queries*, e.g., searching for a range of GPS coordinates. We have performed simulations of the attribute searches and range queries to evaluate and illustrate the advantages of our approach. Those results are discussed in Sec. V. One would expect that the reduced resource consumption is paid for by a higher search latency compared to existing systems. However, the simulation results show that our approach can actually outperform other approaches on both fronts. Before explaining our approach in Sec. IV, we will first illustrate some usage scenarios for distributed complex query computation on resource-constrained devices and will subsequently discuss related work (Sec. III).

This paper is based on work previously published at the 2009 IEEE International Conference on Communications [1]. It adds a significantly extended evaluation of the system, including an evaluation of range queries and the evaluation of the system with respect to different kinds of popularity changes of requested items. Furthermore, we have implemented and evaluated a simulation that compares our approach to a standard LRU caching scheme under the same requirements. We have also added a discussion of possible scenarios for the presented algorithms and a more thorough discussion of the system architecture and the simulation model.

II. Scenarios

There are generally two different ways to perform complex queries on mobile devices. First, resource-constrained devices only pose queries while the queries themselves are resolved elsewhere, e.g., by a centralized server or some distributed infrastructure in the Internet. Second, resource-constrained devices pose queries and also resolve those queries autonomously in a distributed fashion, i.e., without requiring access to an external infrastructure that performs query resolution. The search approach presented in this paper is relevant for the second case. If the resource-constrained devices resolve queries autonomously, then the search algorithm has to be adapted to the constrains of the devices.

Examples of resource-constrained devices include mobile devices like cellphones, PDAs, and netbooks. Those devices have to perform the query resolution themselves in two cases: (1) no external infrastructure is available or reachable and (2) it is not desirable to rely on such an infrastructure. A typical example of scenario (1) includes a disaster scenario where mobile devices are connected in an ad hoc mode without connectivity to the Internet or an external infrastructure. In such a disaster scenario, users heavily rely on always retrieving the latest data. Solutions that rely on, e.g., passive caching that might contain outdated data are not acceptable. In addition, such a scenario is typically characterized by heterogeneous devices with different capabilities and mobility characteristics, including semi-mobile devices like laptops and netbooks, and highly mobile devices like PDAs and cellphones. Here, more powerful devices could directly participate in the P2P system and could in addition serve as proxies for very constrained or extremely mobile devices as described in Sec. IV-C.

Even if Internet connectivity is available, it is desirable in many cases to perform the query resolution autonomously in a distributed fashion. Besides reducing (potentially expensive) Internet traffic, using an external search service that indexes all available information might not be desirable for privacy reasons. In addition, an external search service always requires a service provider which demands some form of compensation and might not always be available.

When talking about complex queries, we refer to multi-attribute and range queries in this paper. Multi-attribute queries are required to perform queries based on more than one attribute. The attributes can be combined in a disjunctive or conjunctive way, i.e., enabling OR and AND connections of attributes. This is useful to, e.g., search for information on resource-constrained devices based on multiple different attributes, e.g., searching for data based on a geographical longitude and latitude value. Range queries are required if the exact attribute value is not known or a range of values is relevant. For example, context awareness applications [2] are mainly performed using mobile devices and often require range queries as well as multi-attribute queries. An integration of both multi-attribute and range queries are multi-dimensional range queries. An example of a multi-dimensional range query is the search for all data in a certain area, involving a range of latitude and longitude values.

III. Related Work

This section first describes related work and will then discuss the basics of MAAN [3] upon which our system builds.

A. Complex distributed query resolution

Today's P2P systems offer only very limited support for advanced, distributed search. While some systems, like BitTorrent [4] or eDonkey2000 [5], 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 [6], [7] 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 [3], [8] or hypercuboids [9]. Second, schemes based on space-filling curves [10], [11]. 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 [10] or so called *hubs* [8], 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 [9], [11]. 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 attribute replicas on several nodes. This has significant advantages compared to the other approaches as it reduces the number of query messages and the response latency.

In summary, none of the existing algorithms fulfills the requirements posed by a universal, distributed search system based on resource-constrained devices. Most approaches typically have higher response latencies or generate significant messaging overhead and/or have high memory requirements. The MAAN approach is a very promising approach for distributed computation of complex queries with respect to low messaging overhead and response latency. In addition, MAAN is one of the few approaches that supports single-attribute, multi-attribute, and range queries. However, the decreased messaging overhead and latency is paid for by significantly increased memory requirements on the participating P2P nodes, which limits the usability of the original MAAN algorithm for resource-constrained devices.

Based on our goal to find an efficient, distributed search algorithm for low latency complex query resolution on resource-constrained devices that supports a wide variety of complex queries, we chose the MAAN algorithm as basis for our new approach because of its universality as well as its good latency and messaging overhead properties. To overcome the high memory requirements, we have performed modifications and extensions to the original algorithm. Specifically, we have modified the search algorithm significantly to reduce its memory footprint and we have added a dynamic replication scheme that introduces an adjustable trade-off between memory footprint and search latency. This trade-off, which none of the other approaches offers, is instrumental for realizing complex queries on resource-constrained devices. It provides low response latencies for popular range and multi-attribute queries while significantly reducing the memory and bandwidth requirements compared to MAAN.

B. Multi-Attribute Addressable Network (MAAN)

MAAN enables searching for IOs based on their meta information. It uses the circular, structured P2P network Chord [12] 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.

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 [13]. This class of hash functions ensures that the addition or removal of nodes does not significantly change the mapping of keys to nodes. The calculated hash value determines the node that is responsible for the attribute value v_i, which is the next node clockwise in the P2P ring with an ID larger than the calculated hash value (node IDs and hash values share the same value range). This node receives a copy of the entire IO, i.e., a copy of all triples *(a_i, v_i, IO identifier)* describing the content of an IO and including the IO's identifier. 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 length of 20 bytes, and an average attribute value size of 40 bytes. Furthermore, each IO contains $n_{\mathrm{A}} = 25$ attribute/value pairs. Those values correspond, e.g., 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) 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. Disjunctions, i.e., 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. Conjunctions, i.e., AND-linked queries, can be processed at a single node owing to all IO attributes being redundantly stored at each node.

3) Range attributes/queries: Range attributes have values that represent a whole interval of numerical values. To support such attributes and to support searching for value ranges, *locality-preserving* hash functions are used. Such functions map attribute values to hash values so that the ordering of the output hash values is identical to the ordering of 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. Correspondingly, all nodes covering a certain range have to be queried to perform a range query for this value range. The number of nodes that are involved in storing range attributes/performing range queries 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.

IV. System Architecture

In the following, we will describe our architecture. Sec. IV-A describes the basic search mechanism for multi-attribute queries; Sec. IV-B for range queries. The mechanisms support single-attribute, multi-attribute, and range queries with much smaller memory footprint compared

to MAAN. On the downside, the latency is increased for multi-attribute queries. To remedy this shortcoming, we introduce a dynamic replication strategy in Sec. IV-A.2 that reduces latency and offers a trade-off between latency and memory usage. Finally, Sec. IV-C discusses the impact of churn.

We use Chord as underlying routing scheme in the course of architecture descriptions and in our simulations. However, our approach is not limited to Chord but can be applied to most distributed search mechanisms where search results are combined from multiple nodes. This is because our system is independent of the overlay routing scheme but only relies on the fact that multiple nodes contribute to solving a search query. This property is given for most networks.

A. Multi-attribute queries

1) Data distribution and basic search mechanisms: The fundamental difference between MAAN and our approach is the way Search Table Entrys (STEs) are stored. MAAN stores the attribute/value pairs redundantly in the system, as described in Sec. III-B.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 Sec. III-B.1, the same Search Table that requires 50 MB in MAAN would only require 2 MB in our system. The reduction by the factor of 25 equals 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.

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 (excluding the dynamic replication mechanism described subsequently) has to visit all nodes responsible for a subquery one after the other. The same applies to multi-attribute *range* queries.

Fig. 1 illustrates the AND-linked multi-attribute query `artist='My Band' AND song='My Song'`. First, the query message (1) is sent to node 11011 which can resolve the subquery `artist='My Band'`. 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.

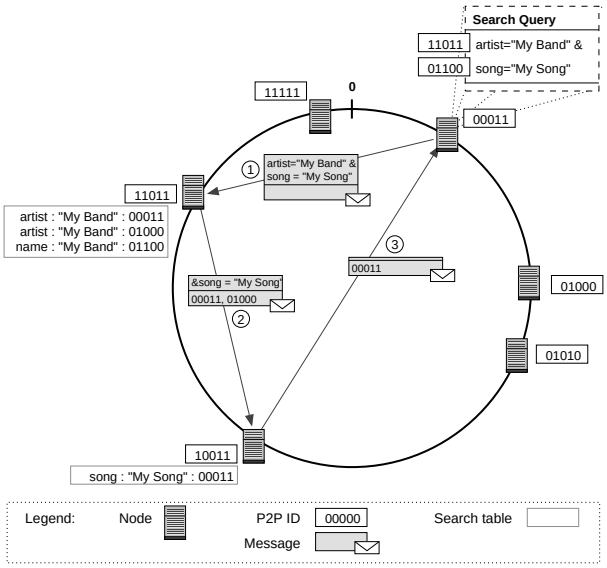


Fig. 1: Resolving the multi-attribute query `artist='My Band' AND song='My Song'` from originating node *00011*.

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, i.e., the scheme actively generates cached copies of popular STEs. Those generated replicas are continuously updated as required via a push mechanism that ensures that caches always contain the latest data as described as a requirement in Sec. II.

Replicas are generated based on dynamically collected usage statistics. Therefore, these replicas only result in redundancy where useful, i.e., where they reduce search latency with high probability. STEs can be replicated multiple times in sequence, i.e., caching nodes can re-replicate their cached STEs to other nodes. This ensures that popular multi-attribute and range queries can finally be answered by a single node.

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 related to this geographic location. Such queries would require contacting two separate nodes in MAAN. The proposed replication scheme, copies popular STEs which enables both search nodes (for latitude and longitude) to answer the complete query at once.

The basis 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 updates on replicas would require more effort compared to the benefit of the replication as each update requires this updated attribute/value pair to be pushed to the caching node.

Fig. 2 shows sample Search and Access Tables of one node. The 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.

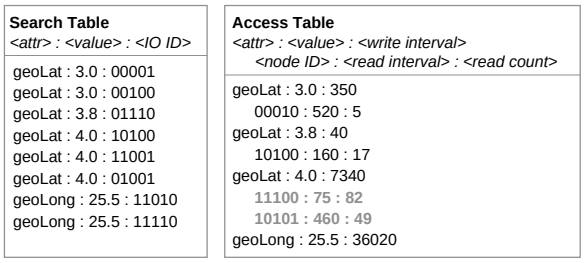


Fig. 2: Exemplary Search Table and corresponding Access Table. The attribute/value combination `geoLat=4.0` has two potential replication nodes, *11100* and *10101*, marked in gray.

Our active replication scheme is push-based, i.e., the pushing node has to decide *when* it is useful to create a new replica. This replication is controlled and triggered by the following two threshold values. Both can be influenced 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 $\overline{t_{\mathrm{R},p,N}} < T_{\mathrm{R,t}}$. If $T_{\mathrm{R,t}}$ is small, only (potentially temporarily) 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.

Our system is tailored for resource-constrained devices, i.e., it is desirable to reduce memory requirements as much as possible. Therefore, instead of passively managing the caches, e.g., via an established cache replacement strategy like Least Recently Used (LRU), we actively eliminate replicas that do not improve the search performance anymore via a Time To Live (TTL) parameter. It influences the replicas' lifetime in the following way:

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. α_{TTL} allows to influence to overall memory requirements. The smaller α_{TTL}, the faster replicas are again removed from the Search Table when they become unpopular. A configuration with $\alpha_{\mathrm{TTL}} < 1$ would mean that the replica is already deleted before it is queried again, which would make no sense.

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.

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 by adapting the parameters $T_{\mathrm{R,t}}$, $T_{\mathrm{R,c}}$, and α_{TTL}.

There are multiple ways to determine an appropriate value for $T_{\mathrm{R,t}}$. For example, the parameter can be adapted to the global inter-read time $\overline{t_{\mathrm{R}}}$ of all requests in the system. Calculating $\overline{t_{\mathrm{R}}}$ can be done using the arithmetic mean or a quantile of the measured distribution of inter-read times in the neighborhood. The arithmetic mean can be computed more easily and robustly in a distributed way [14]–[16]. Therefore we use the arithmetic mean and multiply the measured reference value $\overline{t_{\mathrm{R}}}$ with a constant factor β to calculate $T_{\mathrm{R,t}}$.

Additional details on 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 are given in Sec. V.

Using replication and locality-preserving hashing can cause unequally distributed load among P2P nodes. This can be circumvented by introducing virtual nodes to perform load balancing as described by Ledlie and Seltzer [17].

3) LRU cache replacement scheme: For comparison, we have also implemented a standard replication scheme with least-recently-used (LRU) cache replacement for multi-attribute queries. This replication scheme ignores both parameters $T_{\mathrm{R,t}}$ and $T_{\mathrm{R,c}}$ and directly replicates an STE as soon as a node queries for it. Furthermore, there is no α_{TTL}, instead the cache on each node has a fixed size T_{LRU} and the least recently queried entry is removed if the cache is full.

B. Range queries

1) Data distribution and basic search mechanisms: The mechanisms for multi-attribute queries, described in Sec. IV-A.1, are also valid for value ranges. 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 high memory requirements. Hence, for range attributes, our approach is expected to achieve even higher *absolute* memory savings.

2) Dynamic replication: The dynamic replication process for range queries is essentially similar to the one for multi-attribute queries. It, however, differs in some details.

For making the replication decision, instead of collecting access statistics per attribute/value pair *and* accessing node (cf. Sec. IV-A.2), here the statistics are *only* collected per accessing nodes. The reason is that range queries are usually not done on a discrete set of values; they are continuous. Consequently, without knowing/collecting further statistical details about the query behavior, a node that is responsible for (parts of) a popular value range can only replicate the whole value range for which it is responsible. Thus, in case a node detects range queries for a certain attribute coming frequently from another node, all STEs belonging to this attribute (including those that are already replicas) are sent to the querying node.

Compared to the non-range case where each node keeps track of all *attribute values* in its search table, now each node keeps track of all *range segments* it has stored in its search table. For each of these segments that are replicas a TTL is maintained to be able to remove unneeded replicas again after a while.

C. 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 occur unannounced, 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 for popular STEs.

The question arises whether dynamically generated replicas should also be backed up. Creating backups only 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}, i.e., the number of attributes per IO. 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 alternatively excluding extremely mobile nodes from the P2P ring. To still provide these nodes with the ability to perform search requests, *proxy nodes* [18] can be introduced which perform the actual query resolution in the P2P ring for them. Fig. 3 gives a sample topology showing this proxy functionality.

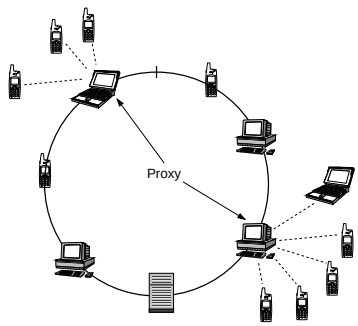


Fig. 3: Sample search infrastructure: Two *peer nodes* act as *proxies* for highly mobile nodes. Nodes connected to a proxy are called *clients*.

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.

V. Evaluation

The proposed search system is evaluated via simulation. Simulation assumptions are discussed in Sec. V-A. Results are presented in the subsequent sections, including a comparison to MAAN's performance.

A. System model

All simulation runs are conducted using the discrete event simulator OMNeT++ [19] and the frameworks INET [20] and OverSim [21].

1) Topology: The topology used for our 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 2 ms latency. 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 (DSL-like connections).

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 (cf. Fig. 3). The client number per peer node is varied for different experiments. The total amount (and hence the total query load), however, is kept constant at 1000 clients. Tab. I shows the parameters of the evaluated scenarios.

TABLE I: Evaluated proxy scenarios

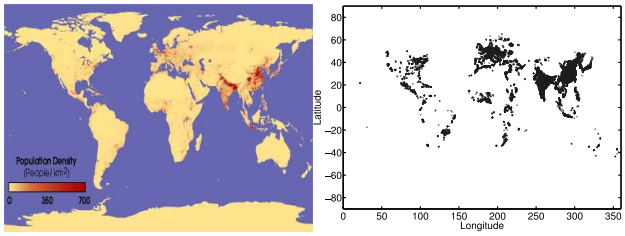
Scenario	$P(\text{proxy})$	μ_{clients}	σ_{clients}
Few dedicated peers are proxies	0.05	200	100
All peers are proxies	1	10	5

In the first scenario, only a few powerful peer nodes act as proxies (5 %). Other nodes either use these proxies for searching, or participate in the P2P search infrastructure only for resolving their own queries. In the second scenario, all peer nodes are proxies, i.e., every peer node can be used by other clients to use the search infrastructure. 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. $P(\text{proxy})$ denotes the probability of a peer node being a proxy. The values in Tab. I all result in a total number of clients that is 10 times the number of peers forming the P2P search infrastructure.

2) Information model: After Chord has finished constructing its routing information for the underlying topology, every peer node adds 30 IOs to the system. These IOs are not modified during the simulation.

In the simulation, each IO contains 5 keyword attributes (a higher number was infeasible due to limited simulation resources). All keywords are selected with Zipf-distributed popularity with parameter 1.01 [22] from a pool of 1000 keywords.

To simulate range queries, each IO also contains a GPS coordinate with a longitude and latitude value. Under the assumption that IOs represent objects in the real world, the associated locations in a real system will probably accumulate to places on earth where many humans are concentrated. Therefore, the coordinates are generated based on the world's population density. Fig. 4a shows a visualization of this density generated from data from the year 1994 [23]. This figure is used to extract the population density and create a set of available coordinate pairs whose popularity is proportional to the population density at that specific location. From this set of locations, concrete values for the location attributes are randomly selected. Fig. 4b shows a plot of 25000 randomly chosen values.



(a) Population density [23] (b) Sample GPS locations

Fig. 4: GPS locations based on population density

3) Multi-attribute search queries: 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 [24]. This corresponds to a Web browsing user behavior.

We also investigate the system behavior on spontaneous keyword popularity changes, like flash crowds. A flash crowd, where few unpopular keywords suddenly become very popular, is simulated by shifting the keyword popularity by 2. This means that the two least popular keywords become the two most popular keywords. Furthermore, we evaluate the worst case where in our pool of 1000 keywords a shift of 500 is done. This results in the maximum difference in popularity.

For the LRU scheme the cache size T_{LRU} is set to 60 IOs for normal peers and 600 IOs for proxy peers. This means a normal peer can store twice as much data as it initially adds to the system and a proxy peer can store 20 times as much.

4) Range search queries: A range query always searches for a GPS location, consisting of a longitude range and a latitude range. The coordinate value of an IO has to be included in both ranges to be avalid answer for the query. The center of each range is picked uniformly from the pool of locations generated from the population density data as described before. Additionally, the length of each queried range is selected according to a normal distribution with a mean of λ. The inter-query time is identical to the multi-attribute query scenario.

To allow simulating popularity changes in GPS coordinates, a certain percentage of queried coordinates is not picked according to the population density but to a random peak. The proportion of these queries is set up by the parameter ρ. Such a peak may occur, e.g., in case of a natural disaster or a flash crowd event.

5) Metrics: Two main metrics are observed during the simulations. 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.

B. Multi-attribute queries

The Figures 5 to 7 illustrate the trade-off between search latency and memory footprint introduced by our search mechanism for basic multi-attribute queries. For space reasons, only results from the scenario with $P(\text{proxy}) = 0.05$ are shown (cf. Tab. I). Further evaluations can be found in [25]. Each subfigure shows one of the two metrics varying over the simulated time and contains MAAN's performance for comparison. Confidence intervals (95 % confidence level) for all plots have been calculated but are omitted due to their small size and for the sake of clarity.

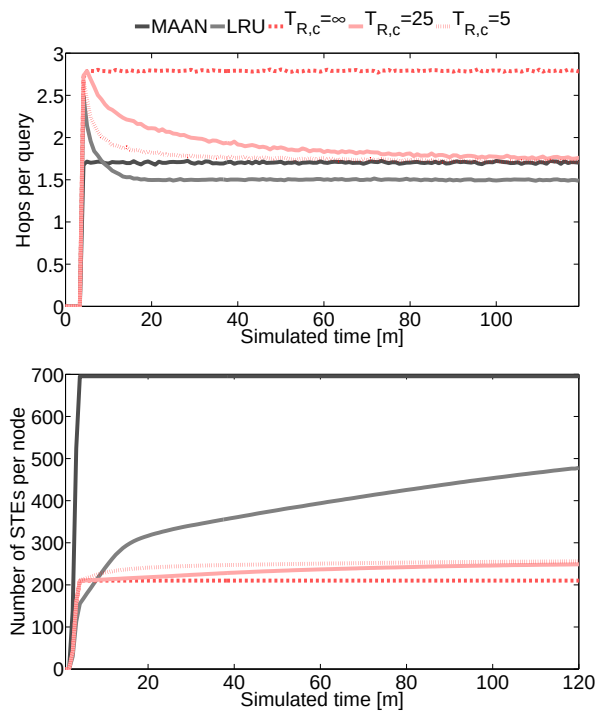


Fig. 5: Hops per search request and STEs per peer node depending on $T_{\text{R,c}}$ (multi-attribute queries); $T_{\text{R,t}} = 5$, $\alpha_{\text{TTL}} = 15$, $P(\text{proxy}) = 0.05$

Fig. 5 points out the influence of the first replication threshold $T_{\text{R,c}}$. The number of hops per search request is constant over time for MAAN and for our scheme with unlimited $T_{\text{R,c}}$. While MAAN requires 1.75 hops on average, our scheme without replication needs 2.75 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 slight deviation 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, direct queries from peers which do not use proxies. In both cases, one hop is omitted. The curves with replication enabled nicely depict that small values for $T_{\text{R,c}}$ accelerate the replication. The steady state, however, is the same. Both two lines with $T_{\text{R,c}} \neq \infty$ converge to a hop count equally to MAAN. At the same time, the memory requirements of our scheme are still approximately 70 % 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 a higher n_{A}.

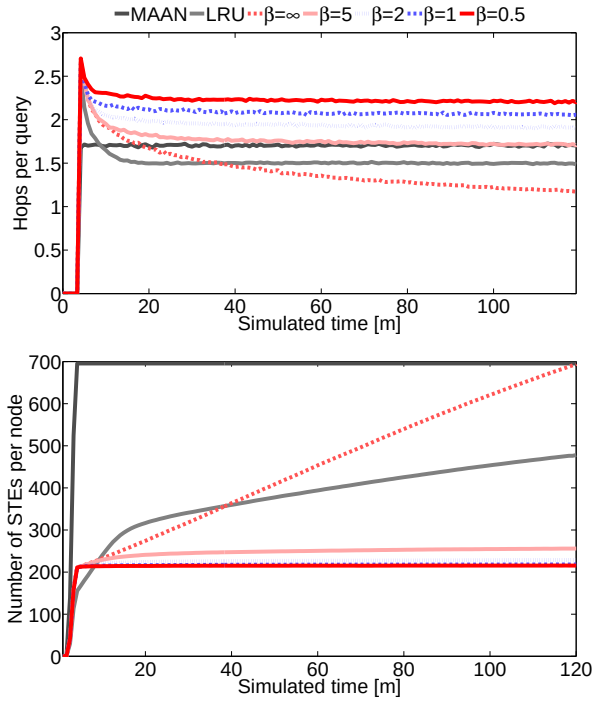


Fig. 6: Hops per search request and STEs per peer node depending on β (multi-attribute queries); $T_{\text{R,c}} = 5$, $\alpha_{\text{TTL}} = 15$, $P(\text{proxy}) = 0.05$

The influence of the threshold $T_{\text{R,t}}$, which is set on the replicating node to prevent copying unpopular STEs, is shown in Fig. 6. The figure depicts $T_{\text{R,t}}$'s influence on the steady state latency and memory requirements. For all $T_{\text{R,t}}$ values, the steady state is reached at the same time. Large values for $T_{\text{R,t}}$ reduce the latency while raising the memory requirements; small values prevent unpopular keywords from being replicated. The measurements show that $T_{\text{R,t}}$ enables nodes to adapt the resource/latency trade-off individually according to their device's abilities. The unbounded memory requirements for an unlimited $T_{\text{R,t}}$ also highlight the importance of the parameter.

Fig. 7 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 $T_{\text{R,t}}$ on the replicating nodes' side. As expected, high values for α_{TTL} cause a low

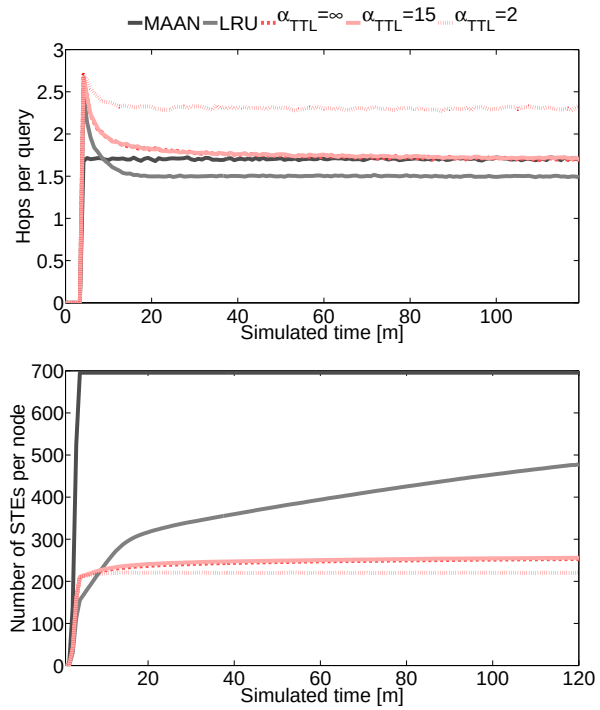


Fig. 7: Hops per search request and STEs per peer node depending on α_{TTL} (multi-attribute queries); $T_{\mathrm{R,c}} = 5$, $T_{\mathrm{R,t}} = 5$, $P(\mathrm{proxy}) = 0.05$

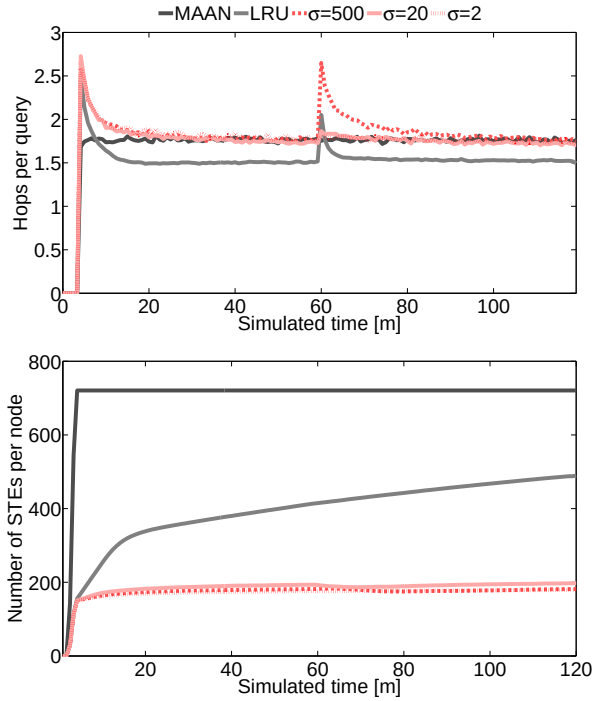


Fig. 8: Hops per search request and STEs per peer node depending on σ (multi-attribute queries, changing popularity); $\alpha_{\mathrm{TTL}} = 15$, $T_{\mathrm{R,c}} = 5$, $T_{\mathrm{R,t}} = 5$, $P(\mathrm{proxy}) = 0.05$

latency with higher memory footprint and low values a high latency with lower memory footprint. Hence, α_{TTL} gives also destination nodes of replicas the possibility to adjust the desired trade-off for themselves.

Figures 5 to 7 also illustrate the performance of our scheme in comparison to the simple LRU scheme. LRU replicates faster, resulting in a slightly smaller latency than our scheme. At the same time, LRU starts to use memory up to a fixed limit and is not able to reduce the memory consumption after that. In contrast, our scheme removes unused STEs after a while, which results in the lower memory consumption shown in the figures.

C. Multi-attribute queries with changing popularity

Fig. 8 to 11 illustrate the trade-off between search latency and memory footprint introduced by our search mechanism under a spontaneous change in the query popularity distribution.

Fig. 8 shows the behavior of our system for different severities of the popularity change after 60 minutes of simulation. Only a high value of 500 for σ, which corresponds to a complete shift of popularity, has a notable influence on the latency. Lower values have only minor influence on the systems performance. The memory footprint is nearly unaffected by the popularity change. Thus, we will only show results for $\sigma = 500$ here.

For all three parameters $T_{\mathrm{R,c}}$, $T_{\mathrm{R,t}}$ and α_{TTL}, as shown in Figs. 9, 10, and 11, the system behaves similarly to simple multi-attribute queries. After the popularity

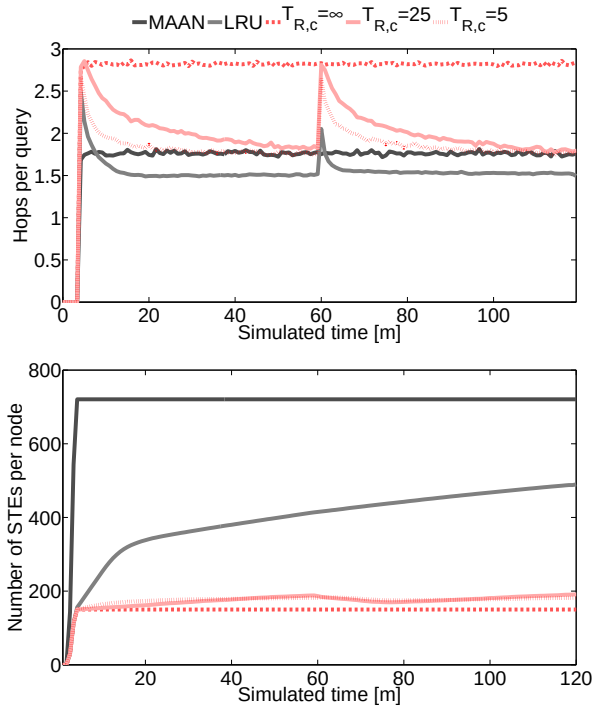


Fig. 9: Hops per search request and STEs per peer node depending on $T_{\mathrm{R,c}}$ (multi-attribute queries, changing popularity); $T_{\mathrm{R,t}} = 5$, $\alpha_{\mathrm{TTL}} = 15$, $P(\mathrm{proxy}) = 0.05$, $\sigma = 500$

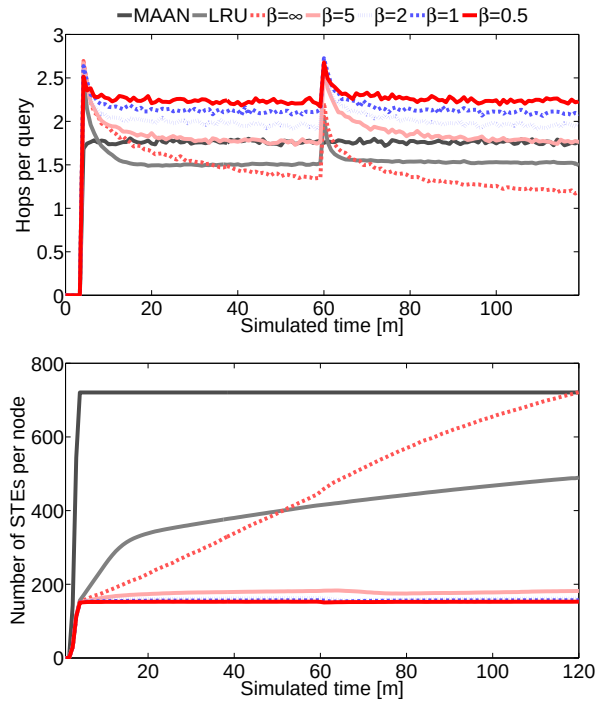


Fig. 10: Hops per search request and STEs per peer node depending on β (multi-attribute queries, changing popularity); $T_{\mathrm{R,c}} = 5$, $\alpha_{\mathrm{TTL}} = 15$, $P(\mathrm{proxy}) = 0.05$, $\sigma = 500$

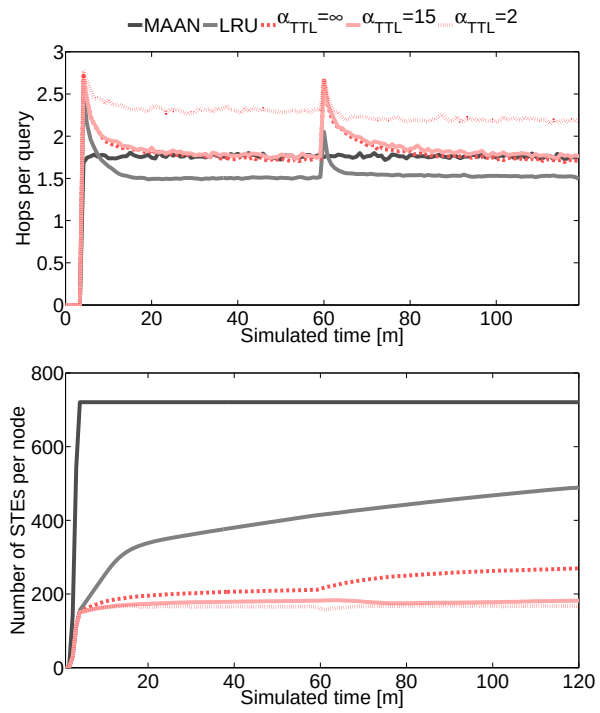


Fig. 11: Hops per search request and STEs per peer node depending on α_{TTL} (multi-attribute queries, changing popularity); $T_{\mathrm{R,c}} = 5$, $T_{\mathrm{R,t}} = 5$, $P(\mathrm{proxy}) = 0.05$, $\sigma = 500$

change the systems adapts itself and reaches its steady stage again. Thus, the parameters can not only be used to establish a fair trade-off between search latency and memory footprint but also to adapt the system behavior to quickly react to flash crowds.

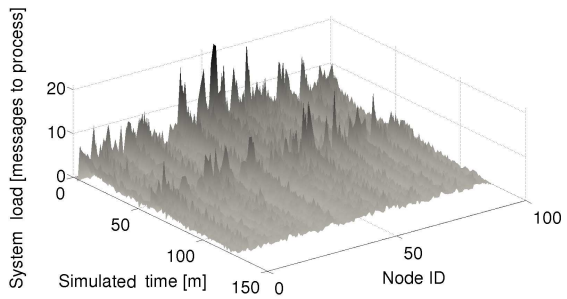


Fig. 12: Load distribution per peer

Figs. 8 to 11 also include plots for the standard LRU scheme. The behaviour is identical to normal multi-attribute queries. While LRU offers a smaller latency, the memory usage is higher.

In addition to the averaged metrics shown in the previous figures, Fig. 12 shows the distribution of load among all peers for parameters $T_{\mathrm{R,c}} = 5$, $T_{\mathrm{R,t}} = \infty$, $\alpha_{\mathrm{TTL}} = \infty$, $P(\mathrm{proxy}) = 0.05$, $\sigma = 500$. The per peer load is calculated as the absolute number of messages to process per measuring window (50 seconds). The figure nicely depicts that the overall load as well as the per-node load decreases over time. It also shows that the initial high load on some nodes is moved to different nodes after the popularity has changed at a simulated time of approximately 50 seconds.

D. Range queries

Range queries incorporate two additional parameters: the mean length of query ranges λ and the fraction of queries related to the random peak ρ (cf. Sec. V-A.4).

In Fig. 13, the influence of the query range length is shown. When using MAAN, a longer range results in a higher latency because more nodes have to be queried for their IOs. Our system is able to adapt to all simulated query range lengths and quickly reaches a steady state with low latency by replicating popular range segments. The memory footprint directly depicts how our system handles unpopular range segments which have been replicated unnecessarily: After a short increase of the memory footprint at the beginning, the TTL mechanism removes unpopular range segments and a steady state with lower memory footprint than MAAN is reached for short query range lengths. For long ranges the memory footprint remains slightly higher than with MAAN, but the latency is still remarkably lower.

The parameter ρ (Fig. 14) also affects how well MAAN is able to handle the range queries. If a large fraction of queries asks for ranges around the random peak value, MAAN has no mean to replicate that value to lower the latency. In contrast to that our approach is able to adapt

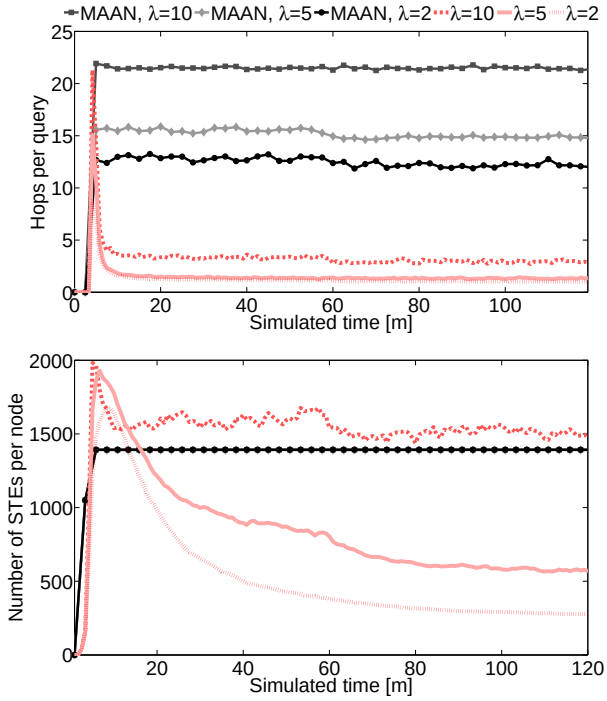


Fig. 13: Hops per search request and STEs per peer node depending on λ (range queries); $\alpha_{\mathrm{TTL}} = 15$, $T_{\mathrm{R,c}} = 5$, $T_{\mathrm{R,t}} = 5$, $P(\mathrm{proxy}) = 0.05$, $\rho = 10$

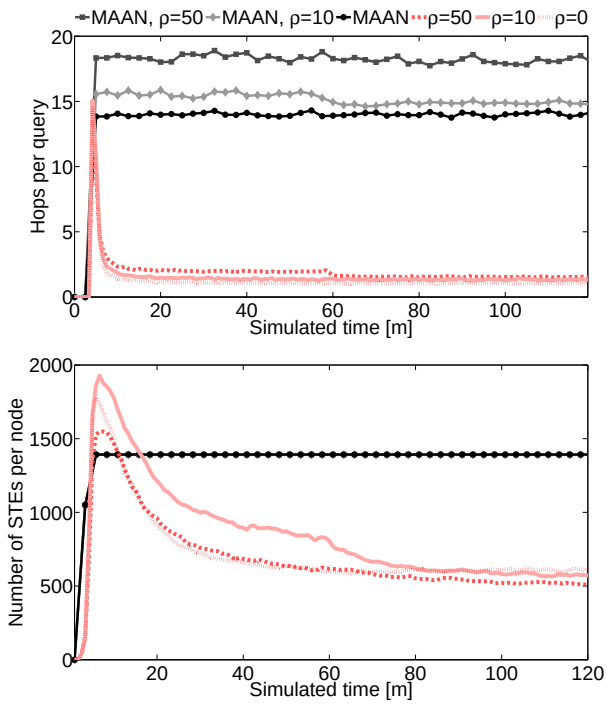


Fig. 14: Hops per search request and STEs per peer node depending on ρ (range queries); $\alpha_{\mathrm{TTL}} = 15$, $T_{\mathrm{R,c}} = 5$, $T_{\mathrm{R,t}} = 5$, $P(\mathrm{proxy}) = 0.05$, $\lambda = 5$

to the peak by replication. Furthermore, a spontaneous change of the peak value (again after 60 minutes of simulation) has no effect on the latency because the system quickly reacts by replicating the new range segments. The peak change's influence on the memory footprint can be neglected.

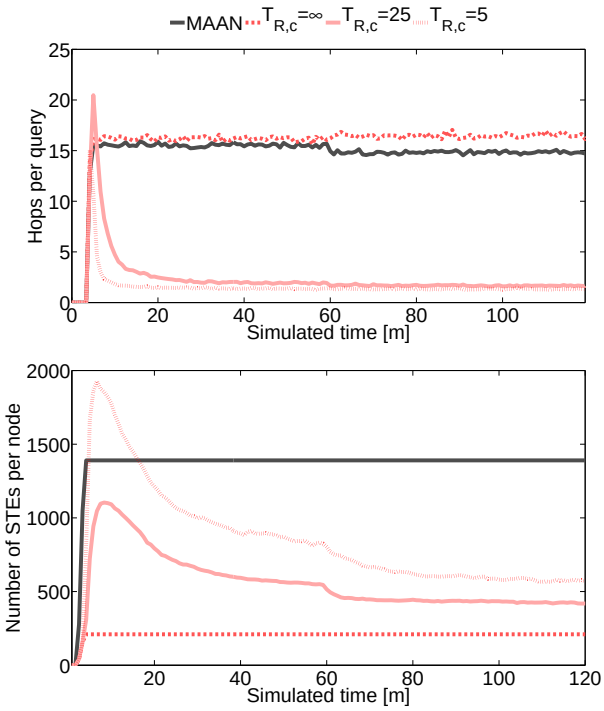


Fig. 15: Hops per search request and STEs per peer node depending on $T_{\mathrm{R,c}}$ (range queries); $T_{\mathrm{R,t}} = 5$, $\alpha_{\mathrm{TTL}} = 15$, $P(\mathrm{proxy}) = 0.05$, $\rho = 10$, $\lambda = 5$

Figs. 15 and 16 show the system's behavior for range queries with the parameters $T_{\mathrm{R,c}}$ and α_{TTL}. When $T_{\mathrm{R,c}} = \infty$, no replication takes place and our system performs similarly to MAAN. Otherwise $T_{\mathrm{R,c}}$ can be used to tweak how quickly the latency increases and how much memory the initial replication phase requires. The memory footprint can be further improved by tweaking α_{TTL}. Fig. 16 also shows that an unlimited TTL value would result in an excessive use of memory.

VI. 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. This is achieved by introducing an adjustable trade-off between search latency and resource requirements (memory, bandwidth). Adapting our scheme to the needs of a specific system is easily possible by adjusting three configuration parameters.

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 search latency. Thereby, our

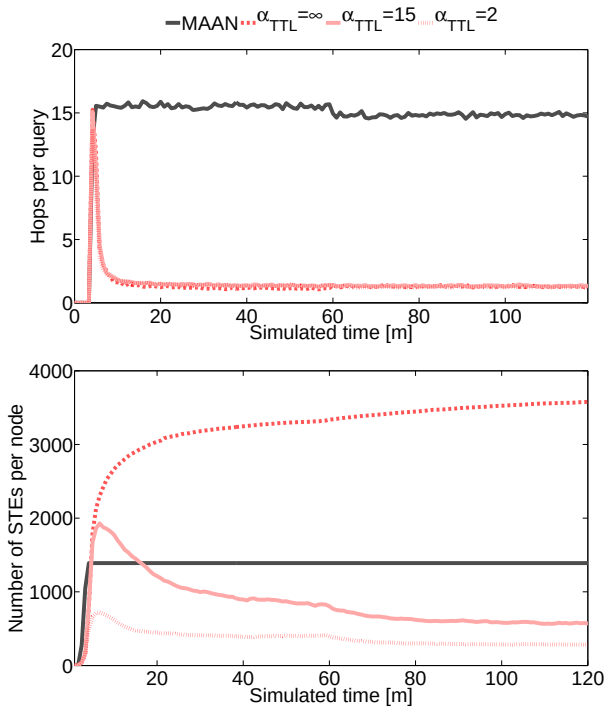


Fig. 16: Hops per search request and STEs per peer node depending on α_{TTL} (range queries); $T_{\mathrm{R,c}} = 5$, $T_{\mathrm{R,t}} = 5$, $P(\mathrm{proxy}) = 0.05$, $\rho = 10$, $\lambda = 5$

system offers advantages in scenarios with resource-constrained devices in terms of high performance in spite of the resource constraints but also in scenarios without resource constraints. Here, users also benefit from a low search latency and low message overhead.

Our dynamic replication algorithm improves the overall system performance and efficiency and, compared to standard mechanisms like LRU, allows immediate distribution of data updates to caches in the network.

Simulations confirm that our mechanisms can handle popularity changes like flash crowds. This emphasizes the benefits for mobile scenarios where frequently changing contexts can result in changing search popularities.

Our distributed search algorithm empowers mobile devices like smartphones, PDAs, and netbooks to become full participants in a P2P network for searching and sharing information. This eliminates the need for additional infrastructure that is in many cases not desirable or simply not available at all.

References

[1] T. Biermann, C. Dannewitz, and H. Karl, "An adaptive resource/performance trade-off for resolving complex queries in P2P networks," in *Proc. IEEE International Conference on Communications (ICC)*, June 2009.

[2] C. Dannewitz, "Augmented Internet: An information-centric approach for real-world / Internet integration," in *Proc. Int. Workshop on the Network of the Future, colocated with IEEE ICC*, June 2009.

[3] 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.

[4] B. Cohen, "The BitTorrent protocol specification," Standard, January 2008. [Online]. Available: http://www.bittorrent.org/beps/bep_0003.html

[5] O. Heckmann and A. Bock, "The eDonkey 2000 protocol," Multimedia Communications Lab, Darmstadt University of Technology, Tech. Rep. KOM-TR-08-2002, Dec. 2002.

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

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

[12] 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.

[13] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web," in *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 1997, pp. 654–663.

[14] L. Xiao and S. Boyd, "Fast linear iterations for distributed averaging," in *Proc. 42nd IEEE Conference on Decision and Control*, December 2003.

[15] C. C. Moallemi and B. V. Roy, "Consensus propagation," *IEEE Transactions on Information Theory*, vol. 52, pp. 4753–4766, 2006.

[16] Q. Zhang and W. Wang, "An efficient algorithm for approximate biased quantile computation in data streams," in *Proceedings of the sixteenth ACM conference on information and knowledge management - CIKM '07*. New York, New York, USA: ACM Press, 2007, p. 1023.

[17] 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.

[18] 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.

[19] OMNeT++ Community, "OMNeT++ discrete event simulation environment," http://www.omnetpp.org/.

[20] INET Community, "INET framework," http://inet.omnetpp.org/.

[21] OverSim Community, "OverSim: The overlay simulation framework," http://www.oversim.org/.

[22] 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

[23] NASA, "Visible Earth: Population Density," http://visibleearth.nasa.gov/view_rec.php?id=116, 03 2008.

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

[25] T. Biermann, C. Dannewitz, and H. Karl, "Extended results on an adaptive resource/performance trade-off for resolving complex queries in P2P networks," University of Paderborn, Tech. Rep. TR-RI-08-294, Oct. 2008.

Christian Dannewitz studied computer science and electrical engineering at the University of Paderborn, Paderborn, Germany, and the Carleton University, Ottawa, Canada. He is currently driving the research of ubiquitous and information-centric networking at the Research Group Computer Networks of the University of Paderborn. Prior to his work at the University of Paderborn, he led the research & development unit of an international software company, focusing on information management.

His research interests include Future Internet technologies with a focus on network architecture and information-centric networking.

Thorsten Biermann is currently a Ph.D. candidate at the Research Group Computer Networks of the University of Paderborn, Paderborn, Germany. He received his Diploma and BS degrees in computer science from the University of Paderborn, in 2008 and 2006, respectively.

His research interests include Future Internet technologies with focus on practical network coding and wireless cooperation techniques. Currently, he is working in the European Union research project 4WARD.

Martin Dräxler received his BS degree in computer science from the University of Paderborn, Paderborn, Germany, in 2009.

Currently, he is a student assistant at the Research Group Computer Networks, University of Paderborn.

Holger Karl studied computer science in Karlsruhe, Germany, and at the University of Massachusetts, Amherst, USA, and received his Ph.D. degree from the Humboldt University of Berlin, Berlin, Germany, in 1999. He was an Assistant Professor at the Technical University Berlin, Berlin, Germany.

He is a Full Professor of computer science at the University of Paderborn, Paderborn, Germany and head of the Research Group Computer Networks, which concentrates on Internet and wireless communication. His main research interests are architectural questions for future mobile communication systems, cross-layer optimization, and wireless sensor networks.