Range query optimization leveraging peer heterogeneity in DHT data networks

Nikos Ntarmos, Theoni Pitoura, and Peter Triantafillou

R.A. Computer Technology Institute and
Computer Engineering & Informatics Dept.,
University of Patras, Rio, Greece.
{ntarmos, pitoura, peter}@ceid.upatras.gr

Abstract. In this work we address the issue of efficient processing of range queries in DHT-based P2P data networks. The novelty of the proposed approach lies on architectures, algorithms, and mechanisms for identifying and appropriately exploiting powerful nodes in such networks. The existence of such nodes has been well documented in the literature and plays a key role in the architecture of most successful real-world P2P applications. However, till now, this heterogeneity has not been taken into account when architecting solutions for complex query processing, especially in DHT networks. With this work we attempt to fill this gap for optimizing the processing of range queries. Significant performance improvements are achieved due to (i) ensuring a much smaller hop count performance for range queries, and (ii) avoiding the dangers and inefficiencies of relying for range query processing on weak nodes, with respect to processing, storage, and communication capacities, and with intermittent connectivity. We present detailed experimental results validating our performance claims.

1 Introduction

Structured P2P systems have provided the P2P community with efficient and combined routing/location primitives. This goal is accomplished by maintaining a structure in the system, emerging by the way that peers define their neighbors. These systems are usually referred to as Distributed Hash Tables (DHTs)[1–3]. DHTs have managed to take routing and location of data items in P2P systems to the next level; from the nondeterministic, flood-based techniques used in unstructured P2P overlays, DHTs provide us with strong probabilistic (under node failures and skewed data and access distributions) guarantees on the worst-case number of hops required to route a message from a node to any other node in the system or, equivalently, for a node in the system to locate data items published therein. Unfortunately, traditional DHT overlays were designed to only support exact-match queries. This has led researchers to investigate how they could enhance P2P systems to support more complex queries[4–15]. On another axis, one of the main characteristics of widely deployed P2P networks (e.g. Gnutella, Kazaa, etc.) is that participating peers are largely heterogeneous, with regard to their processing power, available main memory and disk storage, network bandwidth, and internet connection uptime. Relevant studies of P2P networks [16, 17] have shown that this large heterogeneity is also depicted in the distribution of the query processing chores across

the node population; in the Gnutella network, about 70% of nodes share no files at all with the community, while 5% of nodes serve almost 95% of the queries posed.

Recognizing heterogeneity among peers and harnessing it to speed-up complex query processing has not been done before in the structured P2P world, although most workable real-world P2P applications utilize it, by building multi-level hybrid networks. Our philosophy is based on this very observation. We wish to bring this "hybrid" design into the DHT world and utilize it for complex query processing. The question now is how to do this efficiently. Therefore, we believe that harnessing the power of powerful and "altruistic" nodes is the key to providing an efficient way to expedite complex query processing in a P2P setting. In this work we discuss the range query case. Note that we do not propose another range queriable P2P overlay. We leverage the functionality, scalability, and performance of such network overlays, and present a two-layered architecture were powerful nodes are identified and assigned extra tasks. We do so in an efficient and low-overhead way, using the functionality already provided by the underlying structured P2P network, with significant gains in range query processing costs. To our knowledge, this is the first work to look into this issue.

2 Range Queries over DHTs

Traditional DHTs DHTs use an m-bit circular identifier space for nodes and objects/documents, and modulo-N arithmetic (with $N = 2^m$ being the maximum number of nodes/documents in the system). Both node and document IDs are usually based on some randomizing (usually cryptographic) hashing (e.g. SHA-1) of a node/document specific piece of information. Nodes maintain links to other nodes in the overlay, according to the DHT's geometry and distance function[18]. Using these links, DHTs can route between any two nodes in the overlay in $O(\log N)$ hops, while maintaining $O(\log N)$ links. A document d inserted into a DHT is stored on the node whose ID is closer to the document's ID, according to the DHT's distance function. This node is called the document's "successor" (or "root"). We assume that data stored in the P2P network are structured in a $(k + l)$-attribute relation $R(a_1,..., a_k,b_1,...,b_l)$, where a_i, b_i are the attributes of R, with every tuple t in R being uniquely identified by a primary key $t.key$. This key can be either one of the attributes of the tuple, or can be calculated otherwise (e.g. based on the values of one or more of the attributes of t). Furthermore, attributes a_i are used as single-attribute indices of t, with each a_i being characterized by the domain $t.a_i.\mathcal{D} : \{t.a_i.v_{min}, t.a_i.v_{step}, t.a_i.v_{max}\}$ of its values.

Now suppose that every index tuple is added to the DHT, using an ID generated by (SHA-1) hashing the attribute's value. This would result in tuples being spread across all participating nodes in a uniform manner, but would also lead to tuples with successive index values being stored on completely unrelated nodes. This fact renders traditional DHTs highly inefficient for range query processing; given a range on the domain of the index values, a traditional DHT has to execute queries for each and every value in the range interval! A range query for r consecutive values would require on average r queries to be executed, for a total of $O(r \times log(N))$ hops. In a non-densely populated value domain, most of these queries would return no data items. If all nodes had global knowledge on every value stored in the P2P overlay (we'll call this system the *Enhanced*

DHT), they could then skip queries for non-existing values. Thus, if only r' out of r values exist in the system, it would take on average r' queries, or $O(r' \times log(N))$ hops. Although better than $O(r \times log(N))$, this still is too expensive for a real-world system.

Range-queriable DHTs After the first wave of DHTs, the peer-to-peer research community started investigating structured overlays that would allow for more complex queries than simple equality, while achieving the same performance and scalability figures of early DHTs. This has lead to the design and implementation of several specially crafted DHTs, capable of efficient complex query processing: SkipNet[4], Skip Graphs[5], OP-Chord[11], PIER[13], Mercury[14], P-Trees[6], as well as the works by Ganesan et al.[7, 8], Gupta et al.[9], and Sahin et al.[10], are examples of such systems.

The common idea behind these overlays with regard to range query processing, is that traditional DHTs destroy the locality of content, due to the randomizing (cryptographic) hash functions used to construct document IDs prior to insertion, but locality is a desired property when sequential access is sought, as is the case in range queries. Due to this, these overlays use the actual content (e.g. attribute values in a P2P-based RDBMS environment, file names in a file-sharing system, etc.) rather than the outcome of a (cryptographic) hash function to sort and store documents on the overlay. Thus document locality is preserved and range query processing consists of: (i) locating the node responsible for the start of the range, and (ii) following one-hop "successor" pointers until we reach the node responsible for the end of the range, gathering results in the meantime. If the desired range spans q nodes on the overlay, the above lead to a hop-count complexity of $O(\log N + q)$; $O(\log N)$ hops for phase (i), plus q more hops for phase (ii). We will use the term *LP-DHT ring* to refer to such a locality-preserving, range-queriable overlay in the rest of this paper.

3 The RangeGuard

Our intention is to form a second LP-DHT ring, the RangeGuard ring, above the LP-DHT ring (fig. 1), composed of powerful nodes – the RangeGuards or RGs – burdened with extra functionality chores. Each such node is responsible for storing the index tuples placed in nodes between its predecessor RangeGuard and itself. Thus, if there are M RGs in the system, they partition the normal LP-DHT ring into M continuous and disjoint ranges. Each RG maintains routing information for both the lower-level ring and the RangeGuard ring. Additionally, there is a direct link from each peer to the next RG in the upper-level ring (i.e. to the RG responsible for the peer). Nodes in the lower-level ring probe their RG (e.g. as part of the standard LP-DHT stabilization process), and automatically update the index tuples it stores.

RGs must be (i) powerful enough and willing to withstand the extra (storage, processing, communication) load, and (ii) connected most of the time, to provide hop-count guarantees for range queries and to avoid large transfers due to their joining/leaving. This, in turn, calls for a mechanism to identify and exploit candidate RGs in an effi-

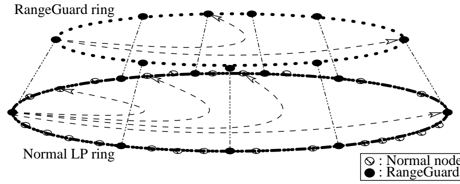

Fig. 1. The RangeGuard architecture. *RG*s form a second LP-DHT ring of their own, taking responsibility (using consistent hashing) for ranges of nodes on the lower-level ring.

cient and transparent manner[1]. Given the functionality offered by our initial (without RangeGuards) infrastructure, RGs are identified and located as follows.

3.1 Node Performance Counters and the Node Performance Relation (NPR)

The administrator of each node selects whether she wants her node to be a candidate for RangeGuard membership or not (much like it is done now with super-peers in most unstructured P2P sharing applications). A candidate node n, with id $n.id$, keeps track of the amount $n.\alpha$ of retrieval and/or just routing requests it serves. This information is updated periodically, every $\mathcal{E}$ seconds (also called an *epoch*). Thus, it keeps two *node performance counters* (or NPCs) – $n.\alpha_c$ and $n.\alpha_p$ – of requests served during the current and the previous epoch respectively.

This information is stored in the system as a four-attribute node performance relation $\mathcal{NPR} : \{n.id, n.\alpha_p, \mathcal{U}, status\}$, with primary key $n.id$, and indexed by $n.\alpha_p$. $status$ is a boolean variable, set to $true$ if the node is a member of the RangeGuard. $\mathcal{U}$ is a counter, incremented on every update of the tuple, and left-shifted (assuming a little-endian architecture) (i) on every update or (ii) every $\mathcal{E}$+δ seconds, with the timer being reset on every update. δ is a quantity depending on measurable characteristics (e.g. round-trip / ping time) of the end-to-end connection between node n and the node storing the index tuple with n's metadata. $\mathcal{U}$ encapsulates the amount of time a peer stays connected to the network. We believe that the network uptime of a peer and the number of requests it has served during this time, are enough evidence of a node's power and fitness for the RG ring. More elaborate metrics may be used instead, under the same intuition of storing these tuples on the lower LP-DHT ring; investigation of such metrics is an open issue and a subject of ongoing and future work. If a node wishes to cease being a candidate RG, it suffices to set a low value (e.g. 0) for its α_p and stop updating this information (so that its $\mathcal{U}$ value decays with inactivity). Also note that the $\mathcal{NPR}$ tuple of a node n is stored on this very node, since the primary key of the relation is the $n.id$ attribute.

Cost of Maintaining NPR Candidate RG nodes must update their $\mathcal{NPR}$ index tuple – remember that $\mathcal{NPR}$ tuples are also indexed by their $n.\alpha_p$ field. This operation requires 2 LP-DHT ring lookups every epoch $\mathcal{E}$; one lookup to delete the index tuple for the old

[1] We assume that peers will not act maliciously or selfishly (leaving countermeasures for such behavior as a possibility for future work), as is the case with most DHT-based research.

value of $n.\alpha_p$, and one to insert the index tuple for the new value of $n.\alpha_p$. The overall cost, in terms of hops, is $O(log(N))$ (since every lookup needs $O(log(N))$ hops), while the bandwidth consumption is minimal given the very small size of these index tuples. Alternatively, we can either keep a link to the node last seen storing the relevant index tuple and start the lookup from there, or follow a soft-state approach. Moreover, the overall cost is tunable via $\mathcal{E}$, so we can trade-off $\mathcal{NPR}$ index freshness for bandwidth and hops.

3.2 Joining the RangeGuard

A node that is to join the RangeGuard uses its RG as the "bootstrap" node for the RangeGuard ring. The RG is responsible for retrieving the metadata of the candidate node and checking whether it is powerful enough (i.e. has served more requests than a predefined threshold) and has stayed online for long enough (based on the corresponding $\mathcal{U}$ value) to be allowed into the RangeGuard. If all prerequisites[2] are met, the standard LP-DHT join protocol is executed and the candidate node is promoted to the RG ring, otherwise the protocol terminates. After that, it updates the $status$ field in its entry in the $\mathcal{NPR}$ relation on the lower-level ring to reflect its promotion to RG status and notifies nodes in its arc of responsibility of its existence. Alternatively, this step may be left as part of the lower-level ring stabilization/maintenance process. The cost of joining the RangeGuard ring consists of: (i) the cost to contact and send the relevant NPR tuple to the RG responsible for the joining node (1 hop), and (ii) the cost of the standard LP-DHT ring *join* protocol, for the RG ring. Thus, the hop-count cost for joining the RG ring is in $O(log(M))$, while the extra bandwidth consumption is minimal (given the expected small size of the RG ring and the very small size of $\mathcal{NPR}$ tuples).

Admission into the RangeGuard There are two ways for a node to be admitted into the RangeGuard: either (i) be promoted by a node already in the RangeGuard who wishes to shed some of its load, or (ii) volunteering to take up some region in the address space for which there exists no RG.

i. Promotion Due to irregularities in the data or access distribution, a RangeGuard may get overloaded with incoming requests. Moreover, it is possible for a region in the RangeGuard to be underpopulated (e.g. imagine the RangeGuard ring in its setup phases). In such cases, a member of the RangeGuard can ask for support from candidate RangeGuards by promoting them to RG status.

With the infrastructure described earlier, when a RG wants to promote a node to RangeGuard status for a region around a point p – i.e. the id of a node in a distance of at most ϵ from a point p in the lower-level ring, with access count greater than a in the previous epoch, and a $\mathcal{U}$ value above u – it merely executes the range query:

select id **from** $\mathcal{NPR}$ **where** $\mathcal{U} > u$ **and** $\alpha_p > a$ **and** $0 \leq id - p < \epsilon$

The result set of this query will contain the IDs of candidate RGs in the region of interest. It is then up to the RG who originated the query to select the best candidate, inform it of its promotion, and initiate the *join* protocol to add it to the RG ring.

[2] These thresholds will probably vary depending on the semantics sought from the RangeGuard. Calculating crisp theoretical thresholds is an orthogonal issue and left as future work.

ii. Volunteering Candidate RGs may lie in any region on the lower-level ring. For data/access distribution irregularity reasons similar to the ones urging RangeGuards to ask for support, it is possible for some regions to have such low data/access loads that the RangeGuard responsible for them has never been in need of support. This could result in large arcs/ranges on the lower-level ring being mapped to a single RangeGuard node, located many hops away on the lower-level ring from the first nodes on this arc. Although this is not an issue in the steady state, it may increase the time needed by a node on this arc to find a new RG, should the current RG leave the system abnormally.

We, thus, allow candidate RG nodes on the lower-level ring to volunteer for an RG position; if a candidate RG detects a situation as the one described earlier (i.e. a large distance between itself and its RG), it can contact the latter and ask to be promoted to RG status. The RG is responsible for going through the α_p and $\mathcal{U}$ checks and admitting the candidate to the RangeGuard or not.

3.3 Leaving the RangeGuard

Similarly, a RG may decide to leave the RangeGuard if it finds itself in a situation where its arc of responsibility becomes very small (due to candidate RGs being promoted to RG status in its vicinity), or the load it faces as an RG drops below some predefined threshold (e.g. an estimate of the load it should have, based on uniform data/access load). A RG that wishes to leave the RangeGuard ring, goes through the following steps: (1) it follows the LP-DHT ring "leave" protocol, transferring its RG-related stored data to the appropriate node(s) on the RG ring, (2) it updates the $status$ field in its entry in the $\mathcal{NPR}$ relation on the lower-level ring, to denote that it is no longer a RangeGuard, and (3) optionally, it notifies the nodes that link to it on the RG ring to update their links, or leaves this to be done as a part of the RG ring stabilization/maintenance process.

Note that our approach uses standard LP-DHT ring operations to set up and maintain the RangeGuard ring. With the exception of the second step, the procedure described above is the standard LP-DHT *leave* protocol. Also note that the leaving RG does not need to notify nodes on its arc of responsibility of their new RG, since that will be achieved during the lower-level ring stabilization process. Consequently, the cost for a node to leave the RangeGuard is equal to the cost of executing the standard LP-DHT *leave* protocol for the RG ring, while data transfer is minimal due to the very small size of $\mathcal{NPR}$ tuples and the size of the RG ring. On the other hand, there is the requirement for several RangeGuard peers with enhanced capabilities. This is not unrealistic, since many peers in real-life applications have proved to be more powerful. With the notion and exploitation of RangeGuards we can harness this power heterogeneity to achieve higher efficiency in range query processing.

3.4 Range Query Processing Using RGs

With RangeGuards in the scene, a query $(a_i.v_{low}, a_i.v_{high})$ on attribute a_i will be sent from the requesting node directly (1 hop) to the RG responsible for the requesting node's data. After this point the RangeGuards assume responsibility to gather the requested information, using the LP-DHT algorithm described earlier, except that now all

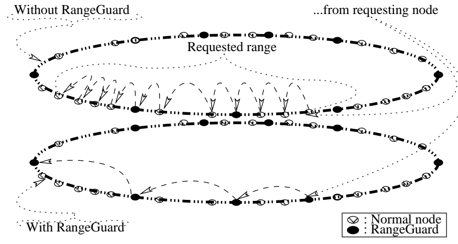

Fig. 2. Range query processing with and without *RG*s.

operations take place on the RangeGuard ring (fig. 2). With data placement on the lower ring being reflected on the RangeGuard ring, the requested index tuples will reside between RG_l responsible for $a_i.v_{low}$ and RG_h responsible for $a_i.v_{high}$. This algorithm requires 1 routing hop to reach the RangeGuard ring, another $O(log(M))$ hops in the RG ring to reach RG_l, and as many routing hops as there are RangeGuards between RG_l and RG_h. Moreover, the $O(log(M))$ term can be further improved to $O(1)$, by using techniques similar to those presented in [19] or [20].

Since, there will probably be much fewer RGs in the system than there are nodes, and RangeGuards are more powerful (with respect to computing capacities, network bandwidth, and network uptime) than the average node in the system, this architecture is significantly more efficient that the one presented earlier. Specifically, for $5\% \times N$ RGs, although the worst-case hop-count efficiency remains in $O(N)$, it now has a rather significantly lower constant modifier (i.e. a 5% – or 20 times – lower hop-count). Note that we require a mere $5\% \times N$ nodes to be powerful and altruistic in our setting; as relevant research has pointed out [16, 17], we can expect an average 5% of the node population in wide-scale peer-to-peer data sharing networks, such as Gnutella and Kazaa, to be powerful, altruistic nodes. Thus, by harnessing the full power of all these nodes, we can achieve even higher performance gains than those outlined above.

3.5 Modifications to the LP-DHT Overlay

We have extended the underlying LP-DHT system in the following fields. First we have provided appropriate protocols to allow nodes to join/leave the RangeGuard ring, while guaranteeing correct operation of the overall system, as well as a method to discover and use candidate RGs (described in detail in sect. 3.2 and 3.3). We have also altered the query processing protocol, to utilize and harness the extra functionality offered by the RangeGuard (described in sect. 3.4). As far as routing state is concerned, we have added 1 more entry to each node to point to the RG responsible for it. For fault-tolerance reasons and faster recovery from failing/leaving RGs, we may choose to maintain links to the next k RGs. Note that routing state size is still in $O(\log N)$.

With the extra information in the nodes' routing tables, we also need to tweak the stabilization process to include the RG entry in the set of links to probe. Much like the standard stabilization process, a node issues a query for its ID on the RG ring. The relevant information is by design stored on the responsible RG; thus, the response to this query will originate from the RG currently responsible for the arc in which the

querying node is located. If the RG responsible for the node has changed (e.g. due to more candidate RGs joining the RangeGuard), the node will get back a response from a different RG and will thus update its RG link. Finally, if all of a node's k RG links have failed simultaneously, then the node can fall-back to querying the lower-level ring for a RangeGuard (i.e. a node whose $status$ field is set to $true$) in its vicinity.

4 Load Distribution on the RG Ring

In order to emulate the 95%-5% observation of [16] (i.e. 5% of all nodes serve 95% of all requests in the system), we have nodes on the LP-DHT ring flip a biased coin and dispatch queries to the RG ring with a 0.95 probability, while processing them solely on the LP-DHT ring with probability 0.05. Apart from relevant provisions by the LP-DHT, the load is further balanced on the RG ring by the load-aware join/leave protocols.

As far as join/leave is concerned, remember that RGs may call for support from candidate RG nodes when overloaded, and may decide to leave the RG ring if their load is too low or their arc of responsibility is too narrow. In the former case, the RG calling for help can choose among several candidate RGs, as returned from the relevant query. Since this node knows which part of its arc of responsibility causes it the more load, it can choose an appropriate candidate RG to shed this very load. The above algorithm is able to provide us with the basics for having a balanced access load. As already mentioned, we expect candidate RangeGuards to be uniformly distributed on the lower-level ring. Thus, RangeGuards calling for help will have a good probability of finding a candidate RangeGuard in their arc of interest.

With respect to data placement, if the popularity of a value does not depend on its position in the attributes domain (e.g. value v is not the most popular for all attributes in the system) then having multiple attributes mapped on the same ring translates to having multiple popular items distributed to all nodes on the system. In the opposite case, a random but easily computable offset value (e.g. the cryptographic hash of the the attribute's name) can be added (mod the maximum document ID) to all values in the attribute's domain. This provides us with enough randomization in the data placement to guarantee similar results, as we shall see in sect. 5.2. Note that, for an N-node system and the worst-case skewed distribution (i.e. one value being selected with probability 1 and the rest with probability 0), then N attributes are required to have a balanced load, under a best-case distribution of load based solely on the above facts. However, since the RG ring is much smaller than the LP-DHT ring, the required number of attributes is much smaller (i.e. $M{<<}N$ attributes for an M-node RG ring)

Furthermore, due to this difference in the sizes of the RG and the LP-DHT rings, every RG node is responsible for the values assigned to multiple nodes on the LP-DHT ring, which leads to an even smoother distribution of the load on the RG ring. Moreover, we can easily apply load balancing techniques developed for the underlying LP-DHT, to further balance the load on both lower-level and RG nodes (e.g. virtual nodes[21], or load-aware node migration[22], when using OP-Chord[11]; rely on the load balancing effects of the overlay itself, when using SkipNets[4] or Skip Graphs[5], etc.)

5 Performance Evaluation

We will be using our home-brewed LP-DHT, OP-Chord[11], as our overlay of choice. OP-Chord is based on Chord, with an order-preserving hash function used instead of SHA-1 for document ID generation, and the same range query processing principles discussed in sect. 2. We have extended the basic Chord simulator (available through http://www.pdos.lcs.mit.edu/chord/), adding support for index tuples and range queries, and implementing our OP-Chord and RangeGuard architectures. We have chosen to test two aspects of the system: (i) the hop count efficiency of our range query processing algorithm, and (ii) the distribution of storage requirements and accesses on participating RG nodes during range query processing, under realistically skewed distributions.

5.1 Hop Count

The experiments used a single-index-attribute relation, with the index attribute taking $5,000$ integer values, following a Zipf distribution with θ=0.7 over D[17]. Range queries are generated using a separate Zipf distribution over the domain D (again with θ=0.7) for their lower bound, and a uniformly distributed range span S, ranging from 1% to 50% of the attribute domain. We report on a series of $50,000$-queries experiments for a system with N=1, 000 nodes, M=50 ($\approx 5\% \times N$) range guards, and $50,000$ tuples (the reported results are not sensitive to these values).

Performance Reference Points We have compared the hop-count efficiency of the RG architecture against (i) plain Chord (PC), as a representative of traditional DHTs, (ii) an imaginary, enhanced Chord (EC), where for each range $\mathcal{R}$ the system knows the IDs of the n' nodes storing values in $\mathcal{R}a$, (iii) our OP-Chord architecture, and (iv) a hybrid system where 95% of queries are processed on the RG ring and the remaining 5% are dealt with on the OP-Chord ring (see sect. 4). Assume we have an integer range query $r = <v_{low}, v_{high}>$. Further assume that the requested index tuples are stored on n' nodes, under Chord's hashing scheme, and on k nodes, managed by k' RangeGuards, under our OPHF scheme. Then, in order to gather all possible results:

- PC: $|r|$ queries, or $O(|r| \log(N))$ hops, are needed.
- EC: $|n'|$ queries, or $O(|n'| \log(N))$ hops, must be executed.
- OP: we must first route to the node holding v_{low} and then follow k-1 successor pointers, for an $O(\log(N){+}k)$ overall hop count.
- RG: we must route to the closest RG (1 hop), then to the RG holding v_{low} ($O(\log(M)$ hops) and follow $k'-1$ successor pointers, for an $O(\log(M){+}k')$ overall hop count.
- $OP + RG$: we flip a biased coin and choose among OP or RG processing, with the hop count complexities outlined above.

Fig. 3 summarizes the measured hop-counts per range query. Traditional DHTs were not designed with range queries in mind thus Chord performs poorly. The (unrealistically) enhanced Chord brings the required hop count down to ≈20% of the Chord figure. However, total global knowledge is required to implement this approach. On the other hand, by using the order-preserving hashing scheme and the RangeGuard architecture, the hop count is decreased by a factor of approx. 50 to 500 compared to PC,

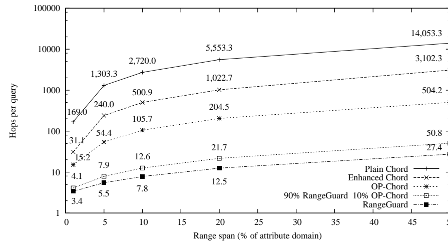

Fig. 3. Hop count per range query (log-plot).

10 to 110 compared to EC, and 5 to 20 compared to OP for different range spans, with the performance of $OP + RG$ following closely behind.

5.2 Load Distribution

The effect of random offsets and of overlapping multiple attributes in the access/storage load balancing is beneficial in our setting. To showcase this claim, we have performed the following experiment: assume we have an OP-Chord ring; we add nodes to the system, at random positions on the OP-Chord ring (simulating the quasi-uniform placement resulting from the use of SHA-1); we let the system stabilize and add $20,000$ multi-attribute tuples in the system. The values of the index attributes are drawn from the $[1,\ 40,000]$ integer interval according to a Zipf distribution with $\theta = 0.7$. If, on the other hand, we assume a uniform value occurrence distribution (as opposed to the above Zipfian distribution), the following results carry on to a Zipf load access distribution. We vary (i) the number of index attributes per tuple, from 1 (the classic single-attribute case of the currently available Chord system) up to 400 attributes, and (ii) the number of nodes in the network from $1,000$ to $5,000$, $10,000$, and $20,000$. Note that, e.g. in the $20,000$-node case, should these nodes be RG nodes, they would be enough to administer a $400,000$-node network, under the 5%-intuition described earlier.

Figure 4 shows the ratio of the highest to the lowest load in the system. Naturally, the optimal load ratio is 1, in which case all nodes in the system will have the same load. With a $\theta = 0.7$ Zipfian value occurrence distribution in an 1,000-node network, the highest-to-lowest single-attribute node access/storage load ratio load is 7.5, dropping to 1.97 for 8, and 1.06 for 400 attributes. We have noted on the figures the load ratio for the single-attribute case (denoted by the "load = " points) and the number of attributes required for this load to drop below 2 (denoted by the "# of attributes" points). With nodes being placed on the lower-level ring using Chord's SHA-1, we can expect RangeGuards to be uniformly distributed on the ring. Thus, the above situation holds for both the lower-level ring and the RangeGuard ring of our architecture. Note, however, the increase in the latter with the number of nodes in the network. As we expect RG nodes to be a small percentage of all network nodes, the above results show that, for the RG ring, load distribution will be within acceptable bounds (even without the other relevant mechanisms discussed in sect. 3). For much larger networks, we shall either need a very large number of attributes to achieve a good load distribution and/or more elaborate load balancing mechanisms[23].

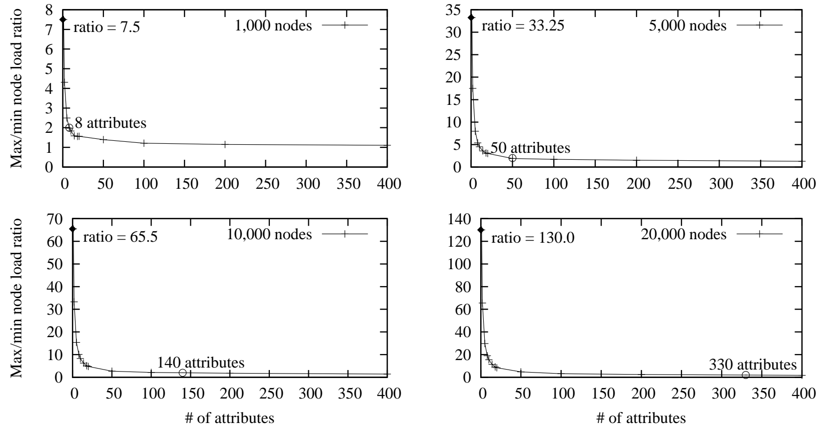

Fig. 4. Highest-to-lowest node load ratio.

6 Related Work

All earlier research efforts focusing on complex query processing over DHTs[4–15] failed to recognize and exploit the key fact that the appropriate utilization of powerful nodes can speed up query processing significantly. Viewed from a complementary angle, earlier research failed to recognize that in large scale data sharing networks, there exist nodes which are weak, with respect to their processing, storage, and communication capacity, and that there also exist nodes with orders of magnitude more horsepower[17]. Our proposal avoids the pitfall of relying upon weak nodes for query processing. Furthermore, we follow a data management approach to discovering and harnessing powerful nodes: keeping metadata for participating nodes as a relation over the LP-DHT ring allows us to swiftly and efficiently locate such nodes and, by promoting them to RangeGuard status, to use them in the core of routing and query processing.

7 Conclusions

With this work we address the problem of efficient range query processing in structured P2P networks. Our approach leverages existing DHT-based P2P research. Our approach is centered on a new architecture that facilitates the exploitation of powerful nodes, coined RangeGuards, in the network, assigning to them specific tasks for further significant speedups during range query processing. This architecture is based on: (i) a way to efficiently identify and collect RangeGuards, and (ii) mechanisms to utilize them during range query processing. Our performance results have shown that significant savings can be achieved by the proposed architecture. Perhaps most importantly, a key advantage of the proposed architecture is that the dangers and inefficiencies of relying on weak nodes for range query processing, with respect to their processing, storage, and communication capacities, and their intermittent connectivity are avoided.

Acknowledgments

Peter Triantafillou was partly funded by FP6 of the EU through IST DELIS (#001907). Nikos Ntarmos was funded by the PENED 2003 Programme of the EU and the General Secretariat for Research and Technology of the Hellenic State.

References

1. Stoica, et al., I.: Chord: A scalable peer-to-peer lookup service for internet applications. In: Proc. ACM SIGCOMM. (2001)
2. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content-addressable network. In: Proc. ACM SIGCOMM. (2001)
3. Rowstron, A., Druschel, P.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: Proc. Middleware. (2001)
4. Harvey, N., Jones, M., Saroiu, S., Theimer, M., Wolman, A.: Skipnet: A scalable overlay network with practical locality properties. In: Proc. USITS. (2003)
5. Aspnes, J., Shah, G.: Skip Graphs. In: Proc. SODA. (2003)
6. Crainiceanu, A., Linga, P., Gehrke, J., Shanmugasundaram, J.: Querying peer-to-peer networks using P-trees. In: Proc. WebDB. (2004)
7. Ganesan, P., Bawa, M., Garcia-Molina, H.: Online balancing of range-partitioned data with applications to peer-to-peer systems. In: Proc. VLDB. (2004)
8. Ganesan, P., Yang, B., Garcia-Molina, H.: Multi-dimensional indexing in peer-to-peer systems. In: Proc. WebDB. (2004)
9. Gupta, A., Agrawal, D., Abbadi, A.: Approximate range selection queries in peer-to-peer systems. In: Proc. CIDR. (2003)
10. Sahin, O., Gupta, A., Agrawal, D., Abbadi, A.: Query processing over peer-to-peer data sharing systems. Technical Report UCSB/CSD-2002-28, UC Santa Barbara (2002)
11. Triantafillou, P., Pitoura, T.: Towards a unifying framework for complex query processing over structured peer-to-peer data networks. In: Proc. DBISP2P. (2003)
12. Gribble, et al., S.: What Can Peer-to-Peer Do for Databases, and Vice Versa? In: Proc WebDB. (2001)
13. Huebsch, et al., R.: Querying the internet with PIER. In: Proc. VLDB. (2003)
14. Bharambe, A., Agrawal, M., Seshan, S.: Mercury: Supporting scalable multi-attribute range queries. In: Proc. SIGCOMM. (2004)
15. Andrzejak, A., Xu, Z.: Scalable, efficient range queries for grid information services. In: Proc. P2P. (2002)
16. Adar, E., Huberman, B.: Free Riding on Gnutella. First Monday (2000)
17. Saroiu, S., Gummadi, K., Gribble, S.: A measurement study of peer-to-peer file sharing systems. In: Proc. MMCN. (2002)
18. Gummadi, K., Gummadi, R., Gribble, S., Ratnasamy, S., Shenker, S., Stoica, I.: The impact of DHT routing geometry on resilience and proximity. In: Proc. SIGCOMM. (2003)
19. Gupta, A., Liskov, B., Rodrigues, R.: One hop lookups for peer-to-peer overlays. In: Proc. HotOS IX. (2003)
20. Ntarmos, N., Triantafillou, P.: AESOP: Altruism-Endowed Self-Organizing Peers. In: Proc. DBISP2P. (2004)
21. Rao, et al., A.: Load balancing in structured P2P systems. In: Proc. IPTPS. (2003)
22. Karger, D., Ruhl, M.: New algorithms for load balancing in P2P systems. In: Proc. IPTPS. (2004)
23. Pitoura, T., Ntarmos, N., Triantafillou, P.: HotRod: Range query processing and load balancing in peer-to-peer data networks. Technical Report TR 2004/12/05, R.A. Computer Technology Institute (2004)