Optimized Information Discovery using Self-adapting Indices over Distributed Hash Tables

Faraz Memon, Daniel Tiebler, Frank Dürr, Kurt Rothermel
IPVS – Distributed Systems Department, Universität Stuttgart
Universitätsstraße 38, 70569 Stuttgart, Germany
Email: {faraz.memon, tiebledl, frank.duerr, kurt.rothermel}@ipvs.uni-stuttgart.de

Abstract

Distributed Hash Table (DHT)-based peer-to-peer information discovery systems have emerged as highly scalable systems for information storage and discovery in massively distributed networks. Originally DHTs supported only point queries. However, recently they have been extended to support more complex queries, such as multi-attribute range (MAR) queries. Generally, the support for MAR queries over DHTs has been provided either by creating an individual index for each data attribute or by creating a single index using the combination of all data attributes. In contrast to these approaches, we propose to create and modify indices using the attribute combinations that dynamically appear in MAR queries in the system.

In this paper, we present an adaptive information discovery system that adapts the set of indices according to the dynamic set of MAR queries in the system. The main contribution of this paper is a four-phase index adaptation process. Our evaluations show that the adaptive information discovery system continuously optimizes the overall system performance for MAR queries. Moreover, compared to a non-adaptive system, our system achieves several orders of magnitude improved performance.

1. Introduction

During the past decade, DHTs have led the way for distributed, scalable and fault-tolerant information discovery systems. DHTs have been extended from their original form, where they supported only point queries, to meet modern application demand of supporting multi-attribute range (MAR) queries. Queries such as, "*find all computers with RAM from 2 to 6 GB and CPU speed from 1.0 to 4.0 GHz*" or "*find all restaurants open from 10 to 11 PM and with seating capacity for 8 to 10 people*", are typical examples of MAR queries.

DHTs have been extended using three different indexing approaches to provide the support for MAR queries. The first approach maps the value ranges of individual data attributes to a network of peers [4, 5, 19, 21]. A MAR query is resolved by dividing the query into multiple single-attribute range queries and then by joining the results at the query initiator. The second approach indexes the combination of all data attributes [6, 10, 18]. Data attributes that are not included in a MAR query are considered to be wild-cards in this approach. The third type of approach, employed by our Optimized Information Discovery (OID) system [16], indexes several attribute combinations with each combination different from the other. A MAR query is resolved by selecting the most efficient index for performing the query.

Although the third type of indexing approach outperforms the other two approaches in terms of individual query efficiency [16], the overall system performance still depends on the attribute combinations used for defining each index. The efficiency of the overall system increases with increasing number of queries being able to find a closely matching index, in terms of the used attribute combination.

In [15], we presented a tool that assists the designer of a distributed application in defining a useful set of indices for the third type of DHT indexing approach. Given a limit for the maximum number of indices and a representative set of MAR queries (*workload*), our tool recommends a set of indices that produces close-to-optimal system performance for the workload within the given limit.

The index recommendation tool is an offline tool, i.e., it is assumed that the workload provided to the tool is somehow collected from an already existing DHT-based information discovery system. Further, it is assumed that the recommended set of indices is manually installed over the DHT by the designer of the distributed application. In order to carry out such an installation, the information discovery system would have to be taken offline, which is highly undesirable for large-scale peer-to-peer (P2P) systems. In this paper, we relax these assumptions to present an adaptive OID system. The adaptive OID system performs the task of

Published in Proceedings of 29th International Performance Computing and Communications Conference (IPCCC'10), pages 1-9, December 2010, Albuquerque, New Mexico, USA.

index recommendation and index installation online, eliminating the need for manually updating the set of indices.

The main contribution of this paper is the index adaptation process. The index adaptation process in a DHT, including online index recommendation and index installation, is carried out in four phases. During the first phase, a workload of MAR queries is collected from several peers in the network using uniform random sampling. The second phase involves execution of the index recommendation tool to determine an optimal set of indices for the collected workload. During the third phase, the cost and the benefit of installing the recommended set of indices is calculated. If it is beneficial to install the recommended set of indices, the installation is carried out during the fourth phase.

The rest of the paper is organized as follows: in Section 2 we give an overview of the related work, the architecture of the adaptive OID system is discussed in Section 3, in Section 4 we describe the index adaptation process in detail, evaluation results are presented in Section 5, and finally we conclude the paper with an overview of our future work in Section 6.

2. Related Work

A number of adaptive P2P information discovery systems have been proposed in the past. In this section, we discuss some of them in relation to our system.

2.1. Unstructured P2P Systems

Several unstructured P2P information discovery systems have been suggested that improve the efficiency of future queries based on the past query workload in the system [3, 13, 14, 17]. The major difference between these systems and the structured P2P systems such as ours is that, each peer in these systems tries to optimize the performance of queries individually by modifying local data index. This does not necessarily lead to the optimization of overall system performance. Moreover, given a query, these systems perform only a best-effort search in the network, i.e., not all matching data objects are always retrieved.

2.2. Structured P2P Systems

In order to improve the search efficiency of queries in structured P2P information discovery systems, several DHT extensions have been proposed [7, 8, 20].

Deng et al. [7] introduce learning-aware blind search for range queries in DHTs. Each peer in their system stores information about previously retrieved results from each link of the DHT using a local index structure. Queries are forward to regions of the DHT that had previously returned the highest number of results. Unlike our system, their system performs only best-effort search since each peer tries to optimize the query performance individually.

Skobeltsyn et al. [20] present a system that stores the results of frequently issued queries at certain peers in the DHT. The choice of queries whose results are cached is based on the dynamic workload of queries in the system. A query is resolved first by looking up the results in the local cache. If no results are found, the peer tries to find a neighboring cache with results. If still no results are found, the query is sent to all peer using broadcast. In our system, queries are never resolved using broadcast since it is highly unscalable to resolve queries in such manner. Instead, we optimize indices for efficient query processing.

The HiPPIS system [8] indexes the data in a DHT using hierarchical indices. Each peer in the system logs each query that it issues. If the granularity of a queried attribute changes locally at a peer, e.g., more queries contain "`city`" attribute instead of the "`state`" attribute, the peer checks if the index has to be adapted accordingly. The peer performs the adaptation check by asking every peer in the system for the query statistics on the attribute using flooding. If adaptation is needed, the peer locks all the peers in the system by flooding a lock message. During this period, queries are answered also using flooding. Finally, the adaptation message is sent to all peers in the system using flooding as well. Unlike the HiPPIS system, our system has a flooding-free scalable index adaptation process.

3. System Architecture

The adaptive OID system has a layered architecture (see Fig. 1(a)). The top layer consists of distributed applications that require support for MAR queries. The bottom layer is the DHT layer that provides the service for looking up a key, broadcasting a message, and aggregating a value. The middle layer, known as the OID framework layer, consists of four major components: data index space, data placement controller, query engine, and adaptation engine.

The data index space of OID framework layer consists of several indices. The data placement controller uses these indices to route each data object to the peer responsible for hosting it. The query engine is responsible for distributed query resolution, while the adaptation engine participates in the index adaptation process.

Each data index in the OID framework layer is a Hilbert Space-filling curve (SFC) [11]. Due to locality preserving properties of Hilbert SFC, data objects that are close in a multi-dimensional attribute space tend to be mapped to sets of neighboring peers in a DHT. This enables efficient processing of MAR queries. A Hilbert SFC is defined as:

Definition 1 *A continuous function* $h : (a_1, a_2, \ldots, a_d) \mapsto x \in \mathbb{N}$*, where* $(a_1, a_2, \ldots, a_d)$ *is a point in a d-dimensional Euclidean space and* $\mathbb{N}$ *is the set of natural numbers.*

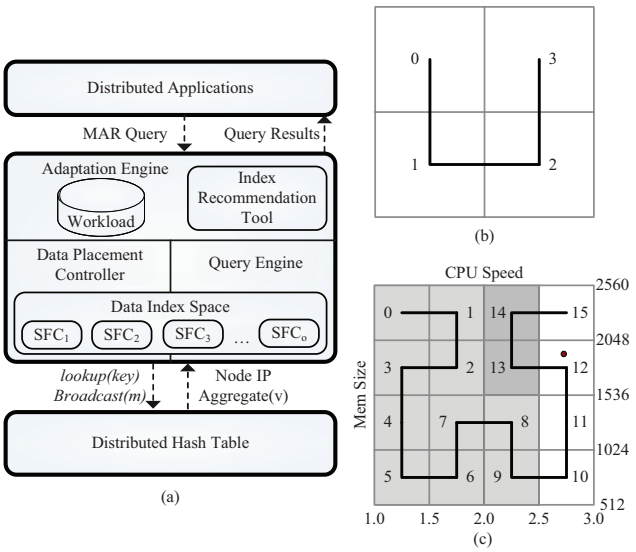

Figure 1. System Architecture

A Hilbert SFC divides a d-dimensional euclidean space into $2^{k \cdot d}$ cubes, called zones. A line then passes through all zones defining an order among them. The result is a k^{th} order SFC, where k, known as the approximation level, defines the granularity of the space sub-division. Figure 1(b)-(c) show a 2^{nd} and a 3^{rd} order Hilbert SFC respectively.

A data object in our system is indexed using each SFC defined in the data index space of the OID framework layer. If the SFC shown in Fig. 1(c) is one such index, then a data object defined as ($\texttt{CPU Speed} = 2.7$ *GHz*, $\texttt{Mem Size} = 1792$ *MB*) would receive an identifier 12 from this index. After a data object receives an identifier from each SFC-based index, a copy of the data object is routed to the DHT peers responsible for the object identifiers. For a detailed description of the data indexing process, see [16].

A MAR query is resolved in two steps. First, the query is mapped to each SFC defined in the data index space. For example, a MAR query defined as "($\texttt{CPU Speed} >= 1.3$ *GHz*) $\wedge$ ($\texttt{CPU Speed} <= 2.3$ *GHz*) $\wedge$ ($\texttt{Mem Size} >= 640$ *MB*) $\wedge$ ($\texttt{Mem Size} <= 2304$ *MB*)" can be mapped to 11 zones on the SFC shown in Fig. 1(c). In the second step, the query is routed to the peers responsible for the zone identifiers obtained using the least expensive index [16].

4. Index Adaptation

The goal of the index adaptation process is to update the set of indices in the OID framework layer of each peer according to the dynamic workload of MAR queries in the system. In order to achieve this goal, we introduce a four-phase index adaptation process that is periodically executed in the system. The four phases of the index adaptation process are: distributed workload collection, index recommendation, adaptation decision, and index installation.

We define following three types of peer roles to carry out the index adaptation process:

Adaptation Peer – An adaptation peer is a peer that periodically initiates the index adaptation process. The length of a period is set by the designer of the distributed application. In order to avoid conflicting index updates, there can only be a single adaptation peer at a time in the network. We assume that the location of the adaptation peer is pre-selected by the designer of the distributed application. This could be done by deciding that the peer that is the successors of a certain key would be the adaptation peer in the network.

If a new peer joins at the location of the adaptation peer, the state of the adaptation peer is transferred to it, making it the new adaptation peer. Moreover, if the adaptation peer fails during the first three phases of the adaptation process, the process is restarted by the new adaptation peer. We assume a correctly functioning DHT where any peer that fails or leaves the network is automatically replaced.

Monitoring Peer – Each peer in our system is a monitoring peer. Monitoring peers are involved in the local collection of the query workload, i.e., each monitoring peer logs each query that it resolves. This log is emptied when a new set of indices is installed in the data index space of the peer.

Sampling Peer – A sampling peer is a peer that is involved in distributed workload collection discussed in the next section. Any peer can take the role of a sampling peer.

4.1. Distributed Workload Collection

Ideally, if the complete set of past queries were collected from all peers in the network, an optimal set of indices could be obtained. However, collecting queries from all peers is neither efficient nor scalable. Therefore, the goal of distributed workload collection is to collect a subset of the complete set of queries by sampling some random peers. The idea is to sample a sufficiently large subset of peers at different locations in the network to get an approximation of the complete set of queries.

The adaptation peer could directly collect a workload of MAR queries by randomly sampling some monitoring peers in the network. However, in this case, the adaptation peer will have to issue a large number of sampling requests and handle a large number of sampling responses, making the sampling process unscalable. Therefore, in order to limit the fanout of the adaptation peer and to make the sampling process scalable, we use a two-level sampling process.

The adaptation peer initiates the first level of the sampling process by generating β random keys from the identifier space of the DHT, i.e., $[0, 2^m)$, where m is the number of identifier bits. A DHT lookup is then performed for each random key in order to identify the peer responsible

for it. Here, we assume a basic DHT lookup functionality that, given a key, returns the identity of the peer responsible for the key. Once the identity of a random peers is learned, a sampling request with parameter γ is sent to it, where γ indicates the number of peers to be sampled at the second level of the sampling process.

Upon receiving a sampling request from the adaptation peer, a peer assumes the role of a sampling peer. The sampling peer then forwards the sampling request to γ random monitoring peers in the same manner as the adaptation peer. After receiving a sampling request from a sampling peer, a monitoring peer responds with the local query workload.

A sampling peer accumulates all the workloads received from γ random monitoring peers into a single workload. Since the same query could have been resolved by several monitoring peers, it could appear multiple times in the accumulated workload. Therefore, duplicates are eliminated during the accumulation process. Note that the same query issued twice is not considered as a duplicate query since each query has a globally unique identifier. Finally, the accumulated workload including the workload of the sampling peer is sent to the adaptation peer where the accumulation process is repeated.

In order to detect the failures of the monitoring or the sampling peers, the process of distributed workload collection includes timeouts at each level. At the level of a sampling peer, if a response is not received from a monitoring peer before the timeout, the sampling request is re-issued assuming that the faulty monitoring peer has been replaced by the DHT. Similarly, at the level of the adaptation peer, if a response is not received from a sampling peer before the timeout, the sampling request is re-issued.

The distributed workload collection phase requires $O\left((\beta \cdot \gamma) \cdot (log_2 N + 2)\right)$ messages in the worst-case to collect a workload of MAR queries. N is the total number of peers in the network and $log_2 N$ is the maximum number of messages required for a DHT lookup. Two additional messages are needed to send a sampling request to a peer and receive a sampling response from it.

4.2. Index Recommendation

Once a workload of MAR queries has been collected at the adaptation peer, the next step in the adaptation process is to search for an optimal set of indices for the collected workload. For this purpose, we utilized the index recommendation tool previously introduced by us.

Given a workload of MAR queries and a limit o for the maximum number of indices, the index recommendation tool recommends a close-to-optimal set of indices I_r for the given workload. For a detailed description of the index recommendation tool and the index recommendation algorithms, see [15].

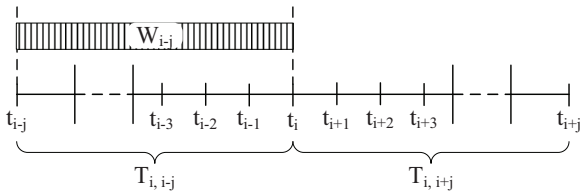

Figure 2. Adaptation Decision

4.3. Adaptation Decision

After obtaining a recommended set of indices from the index recommendation tool, a naïve approach would be to directly install this set of indices in the network. However, it is possible that the cost of installing the recommended set of indices outweighs the benefit of installing it. Therefore, the goal of the adaptation decision phase is to determine whether the installation of the recommended set of indices is beneficial or not. This is done by comparing the estimated cost of the workload over the current set of indices with the estimated cost of the workload over the recommended set of indices. The installation cost of the recommended set of indices is also taken into account.

Let t_i mark the current periodic execution of the index adaptation process, I_c be the current set of indices, and I_r be the recommended set of indices. Then, we define the following quantities in our system (see Fig. 2):

$\mathbf{T_{i,i-j}}$ – Time interval between t_i and t_{i-j} $\forall j \in \mathbb{N}^+$ where, t_{i-j} marks the index adaptation process where I_c was installed. Note that this time interval is dynamic since a new set of indices is not installed during each periodic execution of the index adaptation process.

$\mathbf{W_{i-j}}$ – Complete set of MAR queries during the time interval $T_{i,i-j}$.

$\mathbf{SW_{i-j}}$ – Sampled workload, from the complete set of MAR queries during the time interval $T_{i,i-j}$.

$\mathbf{cost_{in}}$ – Estimated cost of installing the recommended set of indices I_r.

The adaptation peer considers the installation of the recommended set I_r beneficial, if the following condition holds:

$$cost(SW_{i-j}, I_c) > cost(SW_{i-j}, I_r) + cost_{in} \quad (1)$$

i.e., if the cost of the sampled workload SW_{i-j} over the current set of indices I_c is greater than the cost of the same workload over the recommended set of indices I_r plus the installation cost of the recommended set of indices. The assumption behind Condition 1 is that the complete set of MAR queries W_{i-j} would be repeated for a similar interval of time in the future, i.e., for $T_{i,i+j}$. This is the most general assumption for predicting the cost of future queries. If

Condition 1 is satisfied, the next phase of index adaptation is carried out. Otherwise, the index adaptation process is halted until the next periodic execution.

The cost functions $cost(SW_{i-j}, I_c)$ and $cost(SW_{i-j}, I_r)$ in Condition 1 can be generalized as a cost function $cost(Q, I)$. If $Q = \{q_1, q_2, q_3, \ldots, q_l\}$ is a set of queries and $I = \{SFC_1, SFC_2, SFC_3, \ldots, SFC_o\}$ is a set of indices, then $cost(Q, I)$ is calculated as:

$$\begin{aligned} &cost(Q, I) = \sum_{i=1}^{|Q|} cost(q_i, SFC_j) \text{ such that} \\ &cost(q_i, SFC_j) < cost(q_i, SFC_k) \text{ where} \\ &\forall j, k : 1 \leq (j, k) \leq |I| \text{ and } j \neq k \end{aligned} \tag{2}$$

i.e., the cost of a set of queries Q over a set of indices I is a sum of the cost of each query in Q over the least expensive index in I.

In order to determine the least expensive index for a query, the network cost of the query over each index needs to be calculated. Due to highly dynamic nature of P2P systems, this cost cannot be accurately anticipated. However, if the cost of routing a message in the network is known, the maximum cost for resolving a query can be calculated.

Let z be the total number of zones a query maps to, on an SFC-based index. In order to resolve this query, the peer responsible for each zone has to be queried. If a basic query routing strategy is considered, where first a lookup is performed to determine the peer responsible for each zone, then the maximum cost of a query q on an index SFC is calculated as:

$$cost(q, SFC) = z \cdot (log_2 N + 2) \text{ [messages]} \tag{3}$$

where N is the total number of peers in the network and $log_2 N$ is the maximum number of messages needed for looking up a peer responsible for a zone. Two additional messages are needed to send a query request to a peer and receive a query response from it.

In order to check if Condition 1 holds, the cost of installing the recommended set of indices $cost_{in}$ has to be calculated. Similar to the cost calculation above, only the maximum cost of installation can be calculated. Let λ be the total number of unique data objects in the system, then the maximum cost for installing the recommended set of indices I_r is calculated as:

$$\begin{aligned} cost_{in} = 3 \cdot (N - 1) + (|I_r| - |I_c \cap I_r|) \cdot \\ \lambda \cdot (log_2 N + 2) \text{ [messages]} \end{aligned} \tag{4}$$

where $3 \cdot (N - 1)$ is the cost of broadcasting the recommended set of indices I_r, $(|I_r| - |I_c \cap I_r|)$ is the total number of new indices, and $\lambda \cdot (log_2 N + 2)$ is the cost of re-indexing the data. The reason for the broadcast cost being almost three-times the network size is discussed in the next section.

Note that Equations 3 and 4 require the knowledge of global parameters such as N and λ, which are generally not known to a peer in a DHT network. However, an estimate for these parameters could be obtained by installing a reliable broadcast/aggregation tree in the network. The root of this tree will be the adaptation peer in our system. Such a broadcast/aggregation tree could be installed and maintained using the approaches discussed in [9] and [12].

4.4. Index Installation

Once it is determined that installing a recommended set of indices is beneficial, the adaptation peer initiates the index installation phase. The goal of the index installation phase is to broadcast the new set of indices I_r, and to re-index the data on each peer accordingly.

A naïve way of carrying out the index installation phase is to broadcast the recommended set of indices using the DHT broadcast/aggregation tree, and let each peer re-index the data according to the new set of indices. However, queries issued in the system during the re-indexing of the data may not be able to recall the matching data objects completely. This could happen in cases where, e.g., a query issued using the new set of indices searches for matching data objects at a peer where the data has not been placed yet using the new set of indices. In order to avoid this shortcoming, we introduce a 3-step index installation phase.

During the first step, a broadcast message containing the new set of indices I_r is sent by the adaptation peer to each peer in the system. For this purpose, the DHT broadcast/aggregation tree is used. Upon receiving the broadcast message, each peer begins to re-index its data. Note that the old set of indices I_c and the corresponding data is not yet removed from the system. Hence, the queries that are issued during this step continue to be resolved using I_c.

A data object in the OID system is indexed using o number of indices, i.e., $|I_c| = o$. This means that there are o copies of the same data object in the system. Therefore, it has to be made sure that each copy of the data object is not re-indexed using each index in I_r. For example, if I_c and I_r are as shown in Fig. 3, a data object indexed using I_c would be located at four locations in the system. Now if the same data object is re-indexed at each location using every index in I_r, it would be sent four times to each new location in the network. To avoid this, the data is re-indexed as follows.

Data re-indexing at a peer starts with the comparison of the installed set of indices I_c with the new set of indices I_r (see Fig. 3). First, the common elements in both sets are ignored. Next, a mapping is defined from each element in I_c to each corresponding element in I_r. The data objects that had been previously indexed using an element of I_c are now re-indexed only using the corresponding element in I_r. For example, in Fig. 3, the data objects that had been

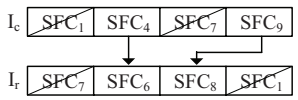

Figure 3. Data Re-Indexing

indexed using SFC_4 in I_c are only re-indexed using SFC_6 in I_r. The re-indexing process involves a look-up for the new location of a data object and then transfer of the data object to this location.

Once the re-indexing of the data is finished at a peer, it sends an acknowledgement to the parent node in the DHT broadcast/aggregation tree. During the second step of the index installation process, the acknowledgements from all peers in the network are aggregated until the adaptation peer receives the aggregated acknowledgement. Queries still continue to be resolved using the old set of indices I_c.

Upon receiving an aggregated acknowledgement from the child nodes in the DHT broadcast/aggregation tree, the adaptation peer starts the third step of index installation by broadcasting a `use index` message. When this message is received at a peer, the peer removes I_c, discards the corresponding data, empties the monitored query log, and starts using I_r for query resolution. Note that the data common between I_c and I_r is not discarded. During this step of index installation, if a query is issued from a peer that has not received the `use index` message yet, then there are two possibilities. First, the query will be resolved using I_c, if all peers involved in query resolution have not discarded the data corresponding to I_c. Second, even if a single peer involved in query resolution has discarded I_c, then the peer that issued the query will be asked to re-issue it using I_r.

If the adaptation peer fails before the first step of the index installation phase, then the process of index adaptation is repeated by the new adaptation peer. However, if the adaptation peer fails after the first step of index installation, the new adaptation peer is already aware of the state of index installation due to the broadcast of new indices in the network. Therefore, the new adaptation peer executes the next steps of the index installation phase.

5. System Evaluation

In this section, we present the results from the performance evaluation of the adaptive OID system. We simulated our system using the PeerSim [1] simulator. The simulations were performed on an AMD Opteron machine with 4 GB of RAM.

Considering resource discovery in grid computing as an example scenario, we represent the data objects in our simulations as resource specifications. Each resource specification consists of attributes shown in Table 1. The value for each attribute in a resource specification is randomly generated from the value domain of the attribute.

Attribute	Value Domain	Definition
CPU Speed	1.0 – 4.0	CPU clock speed in gigahertz
Busy CPU	0 – 100	Percentage of CPU(s) in use
Mem Size	1.0 – 8.0	Total Memory size in gigabytes
Mem Used	0 – 100	Percentage of Memory in use
HDD Size	100.0 – 3000.0	Total HDD size in gigabytes
DL Bandwidth	0.5 – 100	Bandwidth of down link in mbits/sec

Table 1. Attribute List

Unlike the database management systems where benchmark workloads are made available by the TPC [2], no such workload of MAR queries is readily available for P2P systems. Hence, we generate the workloads using the attributes in Table 1 for simulating different scenarios of our system.

For each point on the graphs displayed in this section, the corresponding experiment is repeated 10 times with different workloads, and an average value is plotted.

5.1. Varying Number of Attributes

In this section, we present the results from the performance evaluation of our system using a workload of queries with varying number of attributes. We show that an adaptive OID system is essential for continuous optimization of overall system performance for MAR queries. Table 2 shows the parameter values used for this simulation.

Parameter	Value	Definition
N	1000	Total number of peers in the DHT
n	1600	Total number of queries in the workload
o	3	Maximum number of indices
λ	5000	Total number of data objects
β	33	First level sampling parameter
γ	2	Second level sampling parameter

Table 2. Simulation Parameters

The workload is generated in a manner that the start of the workload contains queries with 4 attributes followed by queries with 3, 2, and 4 attributes again. To simulate a slow change in the workload over time, the attributes in queries are varied slowly, i.e., the change from queries with 4 attributes to queries with 3 attributes and so on, is not sudden.

Each attribute in a query is randomly selected from the list shown in Table 1. Similarly, the range for an attribute in a query is randomly selected from the domain of the attribute. The values for parameters β and γ are set so that almost 10% of peers in the network are sampled.

We simulate the adaptive OID system, the non-adaptive system, and a system with only a single adaptation (partially adaptive system), by executing the generated workload from random peers in the DHT over a period of time. The non-adaptive system is a system with only a single data index over all 6 attributes shown in Table 1. For the partially

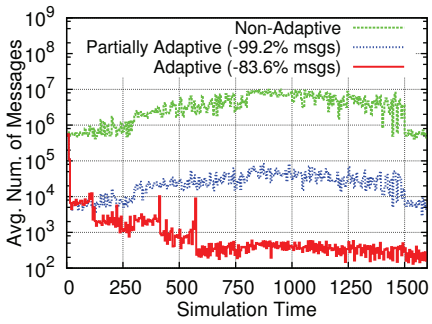

Figure 4. Varying Number of Attributes

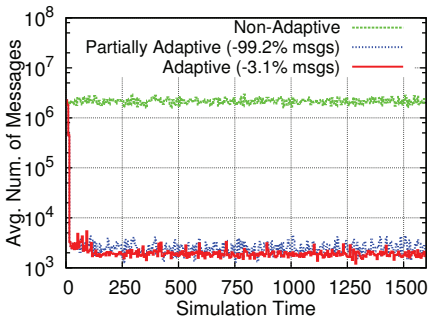

Figure 5. Fixed Number of Attributes

adaptive system, the adaptation takes place after 10 simulation time units. Moreover, for the adaptive OID system, the index adaptation process is scheduled to run after every 10 simulation time units. A single simulation time unit is long enough to allow execution of a single query.

For every 5 simulation time units, we plot the average number of messages in all three systems during that 5-time-unit-window (see Fig. 4). The number of messages represents all the messages in the system including messages for the index adaptation process. For the adaptive OID system, the peaks in the number of messages (see Fig. 4) mark the points where index installation takes place. The higher the peak, the larger the number of indices that are exchanged.

Similar to the non-adaptive system, the adaptive OID system and the partially adaptive system start with one index over all attributes. However, the first adaptation happens very soon in both the systems and 2 additional indices are installed (see Fig. 4). This improves the performance of MAR queries in both systems because the queries are able to find less expensive indices for resolution. Since the first adaptation is based on a very small workload, the second adaptation follows soon in the adaptive OID system. The system continues to adapt itself over time according to the workload of queries. After each adaptation, the performance of MAR queries improves as the average number of messages in the system are reduced.

Figure 4 shows that the partially adaptive system produces 99.2% less messages than the non-adaptive system. Moreover, the adaptive OID system produces 83.6% less messages than the partially adaptive system. Therefore, the adaptive OID system is several orders of magnitude better than the non-adaptive system. Figure 4 also shows that, in order to optimize the overall system performance for MAR queries, a system with continuous adaptations is essential.

The performance of the non-adaptive system worsens with decreasing number of attributes in queries (see Fig. 4). This happens because with decreasing number of query attributes, more attributes have to be considered as wild-cards on a single large index. The performance of the system gets better towards the end of the simulation because the number of attributes in queries increases from 2 to 4 attributes.

In order to further analyze the impact of the number of attributes in queries, we perform another simulation where the number of attributes in the workload is kept constant to 3 attributes. Other simulation parameters have the same values as in Table 2. Figure 5 shows the performance of all three systems with respect to the average number of message in a 5-time-unit-window.

Figure 5 shows that the adaptive OID system quickly adapts its indices to the changing workload of queries. Major adaptations come close to the start of the simulation. After that, even though some small adaptations happen in the system, the performance of the system remains roughly constant. This happens because the indices adapted during the start of the simulation remain beneficial for the complete simulation. The performance of the non-adaptive system remains almost constant, and several orders of magnitude worse than the adaptive system, throughout the simulation.

With a constant number of attributes in queries, the performance of the partially adaptive system comes close to the performance of the adaptive system (see Fig. 5). However, the adaptive system still produces 3.1% less messages compared to the partially adaptive system. This difference in the number of messages grows larger over time. Therefore, in a long running system, the adaptive system would perform significantly better than a partially adaptive system.

5.2. Varying Number of Indices

In this section, we present the performance evaluation of the adaptive OID system by showing the impact of varying number of indices on the system. We perform 3 different simulations using the same workload as in the first simulation discussed in Sec. 5.1. The maximum number of indices o is varied from 3 to 5 across these simulations. Other simulation parameters have the same values as in Table 2. For each simulation we plot the average number of messages in a 10-time-unit-window.

Generally, the larger the set of indices, the better the performance of the system after an adaptation (see Fig. 6), be-

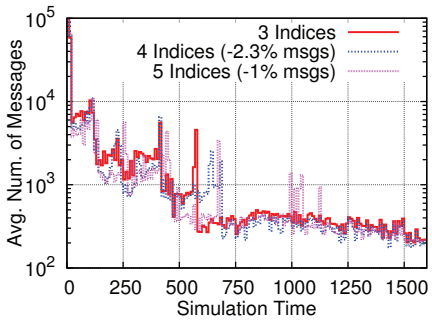

Figure 6. Varying Number of Indices

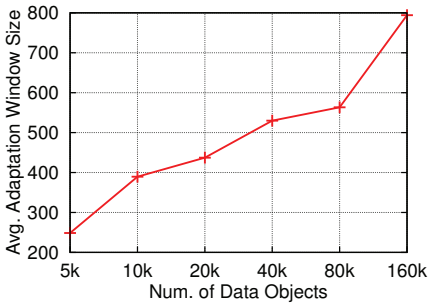

Figure 7. Varying Number of Data Objects

cause with increasing number of indices, more queries find an optimal index for resolution. Since more queries are optimized, the overall system performance also improves slightly with increasing number of indices, e.g., the system with 4 indices produces 2.3% less messages than the system with 3 indices. Similarly, the system with 5 indices produces 1% less messages than the system with 4 indices.

5.3. Varying Number of Data Objects

In this section, we present the performance evaluation of the adaptive OID system by showing the impact of varying number of data objects on the system. We perform 6 different simulations using the same workload as in the first simulation discussed in Sec. 5.1. The total number of data objects in the system λ is doubled across the simulations, starting from 5000 and going up to 160,000. Other simulation parameters have the same values as in Table 2. For each simulation we plot the average adaptation window size defined as: average number of simulation time units needed for an adaptation to happen in the system.

Figure 7 shows the performance of the adaptive OID system with respect to the average adaptation window size. The larger the number of data objects in the system, the longer it takes for an adaptation to happen. The reason is that with increasing number of data objects, the index installation cost also increases. Hence, a larger and more diverse workload is needed for the adaptation to be beneficial.

5.4. Distributed Workload Collection

In this section, we discuss the results from the performance evaluation of the distributed workload collection (see 4.1) phase of the index adaptation process. We perform 16 simulations using the same workload as in the first simulation discussed in Sec. 5.1. For a fixed DHT network size, we vary the values of β and γ across 4 simulations, such that the total number of peers sampled in the network vary between 6% and 12% (in steps of 2%) of the total network size. This simulation scenario is repeated for varying DHT network sizes of $N = (10^2, 10^3, 10^4, 10^5)$. Other simulation parameters have the same values as in Table 2.

During each simulation, after a distributed workload collection phase ends, we measure the cost deviation metric defined as:

$$\left(\frac{|cost(W, I_r^{SW}) - cost(W, I_r^{W})|}{cost(W, I_r^{W})} \right) * 100$$

where W is the complete set of MAR queries from all peers, I_r^{SW} is the recommended set of indices obtained using the sampled workload SW, and I_r^{W} is the recommended set of indices obtained using the complete set of queries W.

The cost deviation indicates how good the recommended set of indices is (in percentage), if it is obtained using the sampled workload, compared to the recommended set of indices obtained using the global workload. The lower the cost deviation, the better the performance of the system because the indices are more optimized for future queries.

For each simulation, the average cost deviation is plotted in Fig. 8. For the network size of 10^2, the calculation for the number of peers to sample using β and γ, was rounded-off to the same value (7% of the network size) in case of 6% and 8% sampled peers.

Figure 8 shows that for a fixed network size, the larger the number of sampled peers, the smaller is the cost deviation. This happens because with increasing number of sampled peers, a better approximation of the complete set of queries is acquired. Hence, the recommended set of indices obtained using the sampled workload is more similar to the recommended set of indices obtained using the complete set of queries. Figure 8 also portrays that with increasing network size, sampling a smaller percentage of peers in the network is sufficient for having a low cost deviation.

6. Conclusion and Future Work

In this paper, we presented the design and evaluation of the adaptive OID system. The adaptive OID system optimizes the overall system performance for MAR queries by dynamically adapting the set of indices in a DHT. The set of

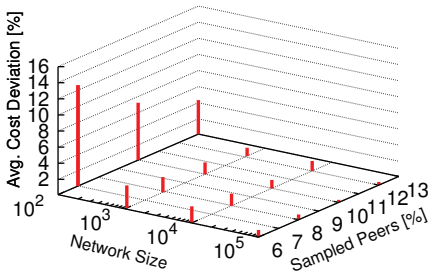

Figure 8. Distributed Workload Collection

indices is adapted using a four-phase index adaptation process. During the first phase, a workload of MAR queries is collected from the DHT network using uniform random sampling of peers. This workload is then used in the second phase for obtaining a new set of indices using the index recommendation tool [15]. During the third phase the cost and the benefit of installing a new set of indices is estimated. If it is beneficial to install the new set of indices, the installation is carried out during the fourth phase of index adaptation process.

Our evaluations show that the adaptive OID system continuously adapts the set of indices in the system according to the dynamic workload of MAR queries. The adaptations are most useful when there is a variety of different queries in the system. Nonetheless, the adaptive OID system shows several orders of magnitude improved performance compared to a non-adaptive system.

Currently, the complete log of MAR queries is retrieved from a peer during the distributed workload collection phase. In future, we plan to change this phase so that it is possible to retrieve the query log until a specified point in time in the past. This would limit the amount of network information flow during the sampling process, making the distributed workload collection phase more scalable.

References

[1] *PeerSim: A P2P Simulator*. http://peersim.sourceforge.net/.

[2] *Transaction Processing Performance Council* . http://www.tpc.org/.

[3] W. Acosta and S. Chandra. Exploiting the Properties of Query Workload and File Name Distributions to Improve P2P Synopsis-based Searches. In *Proc. of Intl. Conf. on Computer Communications*. IEEE, 2008.

[4] A. Andrzejak and Z. Xu. Scalable, Efficient Range Queries for Grid Information Services. In *Proc. of Intl. Conf. on P2P Computing*. IEEE, 2002.

[5] M. Cai, M. Frank, J. Chen, and P. Szekely. MAAN: A Multi-Attribute Addressable Network for Grid Information Services. In *Proc. of Intl. Workshop on Grid Computing*. IEEE, 2003.

[6] Y. Chawathe, S. Ramabhadran, S. Ratnasamy, A. LaMarca, S. Shenker, and J. Hellerstein. A Case Study in Building Layered DHT Applications. In *Proc. of Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications*. ACM, 2005.

[7] Z. Deng, D. Feng, K. Zhou, Z. Shi, and C. Luo. Range Query Using Learning-Aware RPS in DHT-Based Peer-to-Peer Networks. In *Proc. of Intl. Symp. on Cluster Computing and the Grid*. IEEE, 2009.

[8] K. Doka, D. Tsoumakos, and N. Koziris. HiPPIS: An Online P2P System for Efficient Lookups on d-dimensional Hierarchies. In *Proc. of Workshop on Web Information and Data Management*. ACM, 2008.

[9] S. El-Ansary, L. O. Alima, P. Brand, and S. Haridi. Efficient Broadcast in Structured P2P Networks. In *Peer-to-Peer Systems II*. Springer, 2003.

[10] P. Ganesan, B. Yang, and H. Garcia-Molina. One Torus to Rule Them All: Multi-dimensional Queries in P2P Systems. In *Proc. of Intl. Workshop on the Web and Databases*. ACM, 2004.

[11] D. Hilbert. Über die stetige Abbildung einer Linie auf ein Flächenstück. In *Mathematische Annalen*, 1891.

[12] K. Huang and D. Zhang. DHT-based Lightweight Broadcast Algorithms in Large-scale Computing Infrastructures. *Future Gener. Comput. Syst.*, 2010.

[13] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti. A Local Search Mechanism for Peer-to-Peer Networks. In *Proc. of Conf. on Information and Knowledge Management*. ACM, 2002.

[14] G. Koloniari, Y. Petrakis, E. Pitoura, and T. Tsotsos. Query Workload-aware Overlay Construction using Histograms. In *Proc. of Intl. Conf. on Information and Knowledge Management*. ACM, 2005.

[15] F. Memon, F. Dürr, and K. Rothermel. Index Recommendation Tool for Optimized Information Discovery Over Distributed Hash Tables. In *Proc. of Intl. Conf. on Local Computer Networks*. IEEE, 2010.

[16] F. Memon, D. Tiebler, F. Dürr, K. Rothermel, M. Tomsu, and P. Domschitz. OID: Optimized Information Discovery using Space Filling Curves in P2P Overlay Networks. In *Proc. of Intl. Conference on Parallel and Distributed Systems*. IEEE, 2008.

[17] L. T. Nguyen, W. G. Yee, and O. Frieder. Query Workload Driven Summarization for P2P Query Routing. In *Proc. of Intl. Conf. on Peer-to-Peer Computing*. IEEE, 2008.

[18] C. Schmidt and M. Parashar. Flexible Information Discovery in Decentralized Distributed Systems. In *Proc. of Intl. Symp. on High Performance Distributed Computing*. IEEE, 2003.

[19] Y. Shu, B. C. Ooi, K.-L. Tan, and A. Zhou. Supporting Multi-dimensional Range Queries in Peer-to-Peer Systems. In *Proc. of Intl. Conf. on P2P Computing*. IEEE, 2005.

[20] G. Skobeltsyn and K. Aberer. Distributed Cache Table: Efficient Query-driven Processing of Multi-term Queries in P2P Networks. In *Proc. of Intl. Workshop on Information Retrieval in P2P Networks*. ACM, 2006.

[21] P. Triantafillou and T. Pitoura. Towards a Unifying Framework for Complex Query Processing over Structured Peer-to-Peer Data Networks. In *Proc. of Intl. Workshop on Databases, Information Systems and P2P Computing*. Springer, 2003.