International Journal of Computing and Network Technology
@ 2014: Scientific Publishing Center, University of Bahrain

Resource and Query Aware, Multi-Attribute Resource Discovery for P2P Systems

H. M. N. Dilum Bandara[1,*] and Anura P. Jayasumana[2,*]

[1]Department of Computer Science and Engineering, University of Moratuwa, Katubedda 10400, Sri Lanka
[2]Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO 80523, USA

Received 15 Dec. 2013, Revised 11 Feb. 2014, Accepted 16 Feb. 2014, Published 1 May 2014

Abstract: Distributed, multi-attribute Resource Discovery (RD) is a fundamental requirement in collaborative Peer-to-Peer (P2P), grid, and cloud computing. We present an efficient and load balanced, P2P-based multi-attribute RD solution that consists of five heuristics, which can be executed independently and distributedly. First heuristic maintains a minimum number of nodes in a ring-like overlay while pruning nodes that do not significantly contribute to the range query resolution. Removing nonproductive nodes reduces the cost (e.g., hops and latency) of advertising resources and resolving queries. Second and third heuristics dynamically balance the key and query load distribution by transferring some of the keys to its predecessor/successor and by adding new predecessors/successors to handle transferred keys when existing nodes are insufficient, respectively. Last two heuristics form cliques of nodes (that are placed orthogonal to the overlay ring) to dynamically balance the highly skewed key and query loads. By applying these heuristics in the presented order, a RD solution that better responds to real-world resource and query characteristics is developed. Its efficacy is demonstrated using a simulation-based analysis under a variety of single and multi-attribute resource and query distributions derived from real workloads.

Keywords: Multi-attribute queries, load balancing, peer-to-peer, range queries, resource discovery

1. Introduction

Collaborative Peer-to-Peer (P2P) systems [1], grids, and cloud computing require the ability to discover and aggregate groups of heterogeneous, distributed, and dynamic resources as and when needed. These systems utilize a variety of *resources* such as processor cycles, storage capacity, network bandwidth, sensors, scientific algorithms, services, and data to not only consume a variety of contents but also to generate, modify, and manage those contents. P2P-based distributed Resource Discovery (RD) is a natural fit for collaborative applications and further enhances their scalability and robustness. P2P-based RD has also been proposed for conventional applications such as grid, desktop grid, and cloud computing, as a timely aggregation of complex resources is becoming increasingly necessary due to the proliferation of parallel applications that utilize multiple and distributed resources.

Many P2P-based solutions have been proposed to discover multi-attribute, dynamic, and distributed resources [1-5]. However, compared to single-attribute P2P systems such as file sharing, formal characterization of real world, multi-attribute resources and queries received attention only recently [6-8]. In the absence of data and understanding of the characteristics, designs and evaluations of existing solutions have relied on many simplifying assumptions such as independent and identically distributed (i.i.d.) attributes, large domains for attribute values (i.e., number of distinct attribute values $D \gg$ number of nodes N), uniform or Zipf's distributions of all the resources/queries [2, 5], and queries with a large number of attributes and small range of attribute values [2-4]. However, in [6, 8-9], we observed that characteristics of real-world systems diverge drastically where attributes of resources are correlated and characterized by different marginal distributions. Moreover, resources and queries are highly skewed, domains of most attributes are much smaller (i.e., $D \ll N$), and queries tend to request a small number of attributes and large range of attribute values. Analysis in [8, 10], using real-world data from [6] to evaluate seven fundamental design choices for P2P-based RD shows that existing solutions have a high resource advertise and query cost (approximating $O(N)$) as attribute values change frequently and queries are less specific. Moreover, they are prone to significant load balancing issues because $D \ll N$ and resources and queries are highly skewed. While many solutions are

E-mail: Dilum.Bandara@uom.lk, Anura.Jayasumana@ColoState.edu

proposed to balance the index size and query load in P2P systems [4-5, 11-13], they also rely on the aforementioned assumptions (see [8] for details). Such assumptions affect the designs, performance analysis, and applicability of solutions under real workloads. Therefore, more efficient and load balanced, RD solutions are needed to support real workloads.

We present an efficient and load balanced, resource and query aware multi-attribute RD solution. The solution is based on five heuristics that can be executed independently and distributedly on a ring-like overlay. Ring-like overlay is selected as it turns out to be a relatively more efficient and scalable design choice compared to the other solutions [8, 10]. First heuristic maintains only a subset of the nodes in the overlay as $D \ll N$. It prunes nodes that do not significantly contribute to range query resolution while reducing the cost (e.g., hops and latency) of resolving queries. Heuristics two and three dynamically balance the query load and index/key distribution of nodes by transferring part of the keys to its neighbors and by adding new neighbors to handle transferred keys when existing nodes are insufficient, respectively. Last two heuristic, namely fragmentation and replication, form cliques of nodes to dynamically balance the skewed key and query loads associated with few popular resources. In contrast to the common practice of replicating along the overlay ring, cliques of fragments and replicas are placed orthogonal to the ring thereby maintaining lower query cost and better load distribution. Our key contributions are the development of a novel heuristic to prune nodes with lower contribution to the range query resolution, placing replicas and fragments orthogonal to the ring, and extending other four heuristics to support real workloads while overcoming their deficiencies. Moreover, by applying the five heuristics in the presented order, our solution can better respond to the complex characteristics of real-world resources and queries. Simulation-based analysis is used to evaluate the efficacy of the proposed solution under a variety of single and multi-attribute resource and query distributions derived from real workloads.

Section 2 presents the problem formulation. Five heuristics and their application to single-attribute resources are presented in Section 3, and it is then extended to multi-attribute resources in Section 4. Sections 5 and 6 present the simulation setup and performance analysis, respectively. Concluding remarks are presented in Section 7. This is an extended version of the paper in [14]. The major extensions include the problem formulation, formal presentation of heuristics, and an extended performance analysis.

2. Problem Formulation

We first present the resource and query model. Summary of resource and query characteristics observed from real-world systems and P2P load balancing solutions are presented next. Finally, the research problem is formulated.

2.1 Resource and Query Model

Let **R** be the set of resources in the system and **A** be the set of attributes used to characterize those resources. Bold face symbols refer to a set and the corresponding italic symbol refers to its cardinality, e.g., $R = |\mathbf{R}|$. List of symbols is given in Table I. A multi-attribute resource $r \in \mathbf{R}$ is defined as follows:

$$r = (a_1 = v_1, a_2 = v_2, \dots, a_i = v_i) \quad (1)$$

Each attribute $a_i \in \mathbf{A}$ has a corresponding value $v_i \in \mathbf{D_i}$ that belongs to a given domain $\mathbf{D_i}$. $\mathbf{D_i}$'s are typically bounded and may be continuous or discrete, or a set of categories/names. For example, r = (*CPUSpeed* = 2.1 GHz, *CPUCores* = 2, *CPUFree* = 76%, *OS* = "*Linux_2.6*"). Attribute values are further classified as *static* (e.g., *CPUSpeed*) and *dynamic* (e.g., *CPUFree*). Dynamic attributes need to be re-advertised whenever their states change. Thus, the cost of advertising dynamic attributes is proportional to their rate of change. A fixed or a dynamic threshold may be used to reduce the number of advertisements. r is referred to as a *single-attribute resource* when $A = 1$.

Let **Q** be the set of queries issued by the system. A multi-attribute, range query $q \in \mathbf{Q}$ is defined as:

$$q = (m, a_1 \in [l_1, u_1], a_2 \in [l_2, u_2], \dots, a_i \in [l_i, u_i]) \quad (2)$$

where, $m \in \mathbf{Z}^+$ specifies the required number of resources and $a_i \in [l_i, u_i]$ specifies the desired range of attribute values (l_i and u_i are lower and upper bounds, respectively). In practice, attributes specified in a query may contain a mixture of points (i.e., $l_i = u_i$) and ranges (i.e., $l_i < u_i$). The set of attributes specified in a query ($\mathbf{q_A}$) may contain only a subset of the attributes that are used to describe a resource (i.e., $\mathbf{q_A} \subseteq \mathbf{A}$). For example, q = (5, *CPUFree* ∈ [50%, 100%], *OS* ∈ "*Linux_2.5*"). Unspecified attributes are considered as "don't care"s. q is referred to as a *single-attribute query* if $q_A = 1$.

Analysis in [8, 10] shows that a Distributed Hash Table (DHT) built on top of a ring-like overlay (e.g., Chord [12] and Mercury [5]) is relatively efficient and scalable than other design choices for P2P-based multi-attribute RD. Therefore, we consider a ring-like overlay for indexing resources as shown in Fig. 1. Each

v_i of a resource is indexed separately using a Locality Preserving Hash (LPH) function(s) [3]. LPH functions map close by attribute values to neighboring nodes in the ring enabling range query resolution through a series of successors (see Fig. 1). Different attributes may be advertised to different rings [5], different segments of the same ring [2, 4], or to a set of virtual rings mapped to a physical ring [3]. Query q is first split into a set of *sub-queries* where each sub-query searches for one of the attributes specified in $\mathbf{q_A}$. Sub-queries are then simultaneously forwarded to relevant rings/segments and are resolved by forwarding to all the nodes responsible for indexing the specified range $[l_i, u_i]$. Cost of resolving q can be given as:

$$Cost_q = \sum_{i \in q_A} \left(h_i + \left\lceil \frac{(u_i - l_i)}{D_i} \times N \right\rceil - 1 \right) \tag{3}$$

where, h_i is the number of overlay hops to reach the node that indexes l_i, which is typically $O(\log N)$ [12]. After reaching the first node, q will be forwarded to the node responsible for indexing u_i through a series of successors. As in [3-4], we assume the range of attribute values $[l_i, u_i]$ to be uniformly distributed within the overlay. As the domain of attribute i (D_i) is mapped to N nodes, each node is responsible for indexing D_i/N fraction of the attribute space. Therefore, a range query that searches for $[l_i, u_i]$ has to go through $(u_i - l_i)n/D_i$ nodes (i.e., range to search divided by range per node). This approach has a lower resource advertising cost but has a higher query cost due to multiple sub-queries. Alternatively, if all the attribute values are advertised to all the rings/segments, q needs to be forwarded to only one of the rings/segments. Moreover, q can be terminated as soon as m resources are found. Query cost can be further reduced by selecting the ring/segment corresponding to the *most selective attribute* (i.e., attribute with the smallest $(u_i - l_i)/D_i$) [3, 5]. However, cost of advertising resources is high as each v_i needs to be advertised to all the rings or segments.

2.2 Related Work

In [6, 8-9], we presented the characteristics of both the static and dynamic resources as well as the queries from four different real-world systems (namely PlanetLab, SETI@home, EGI grid, and a distributed campus computing facility). These and other recent analysis in [7] invalidate some of the key assumptions used by existing performance studies on RD solutions. For example, [6-9] show that the attributes of both real-world resources and queries are highly skewed and correlated. Several attributes have only a few attribute values, e.g., CPU architecture, *CPUCores*, and *OS*.

TABLE I. LIST OF SYMBOLS.

Symbol	Description
$\mathbf{A}$	Set of attributes used to characterize resources
$\boldsymbol{D_i}$	Domain of attribute i
h_i	No of hops to reach the node indexing lower bound l_i
$\mathbf{I}^i$, I^i_{Cap}	Resource index and capacity of a node
k_i, k_l, k_u	Key of node i, lower bound l_i, upper bound u_i
$\mathbf{K^i_{In}}$, $\mathbf{K^i_{Out}}$	Set of keys corresponding to *In* and *Out* queries
l_i	Lower bound specified in a range query
m	Required no of resources specified in a query
N	No of nodes in the overlay
$\mathbf{q_A}$	Set of attributes specified in query q
$\boldsymbol{Q}$	Set/no of queries issues to the RD system
Q^i_{Cap}	Query capacity of a node/resource
Q^i_{Thr}	Out query threshold
$\mathbf{R}$	Set of resources in the system
u_i	Upper bound specified in a range query
v_i	Value of i-th attribute

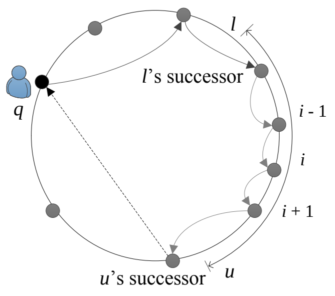

Figure 1. Range query resolution on a ring-like overlay.

Thus, size of their domain $D_i \ll N$. Moreover, distribution of those attributes was extremely skewed, e.g., 98% of the CPUs in SETI@home were ×86. Attribute values have different marginal distributions and change at different rates. For example, 32% of the PlanetLab nodes significantly changed their dynamic attribute values 100 times or more within a 24-hour period (out of 288 samples taken at five-minute intervals). 3% of the nodes changed their attribute values over 200 times within the same period. Furthermore, most queries are less specific where they specify a few attributes (small q_A), large number of resources (large m), and large range of attribute values (large $(u_i - l_i)$). For example, 80% of the PlanetLab queries specified only one or two attributes and 89% of the queries specified *CPUFree* $\in$ [40%, 100%].

These findings invalidate the commonly used assumptions such as i.i.d. attributes, uniform/Zipf's distribution of all the attributes, $D_i \gg N$, and queries requesting a large q_A, small m, and small $(u_i - l_i)$ [2-5]. From (3) it can be seen that the cost of resolving real-world queries approximate $O(N)$ as m, $(u_i - l_i)$, and $(u_i - l_i)/D_i$ are relatively large. This is a problem in collaborative P2P, grid, and cloud computing where N

range from several thousands to millions of resources. Furthermore, as $D_i \ll N$ resources are mapped only to a small subset of the DHT nodes. Consequently, a few DHT nodes are involved in indexing resources and answering majority of the queries while a larger fraction of nodes do not either index resources or answer queries [8, 10]. Therefore, existing solutions are both inefficient and are prone to load balancing issues under real workloads.

Load on a DHT node can be defined in terms of index size, advertise and query messages received, and overlay messages forwarded. Index size is measured using the number of resources or memory required to store those resources. Advertise, query, and forward loads are measured using the number of messages or bandwidth consumed. Existing solutions assume all the nodes should be added to the overlay as it helps to balance the load by each node indexing approximately R/N resources or answering Q/N queries. For example, Chord proposed to balance the index size distribution by having $N \log N$ virtual nodes in the overlay [13] under the assumption that resources are uniformly distributed and $D_i \gg N$. Godfrey et al. [13] extended the concept of virtual nodes to balance the query load by moving virtual nodes from highly loaded physical nodes to lightly loaded ones. These solutions are not suitable for real-world RD as the query cost is $O(N \log N)$, resources and queries are skewed, and most of the nodes will not index resources or answer queries as $D_i \ll N$. SWORD [4] proposed to expand D_i by appending a few random bits to the hash values of v_i such that identical resources will be mapped to different DHT nodes. While this helps to distribute the index size, it does not balance the query load, as queries have to always start from the node corresponding to l_i. One may argue that attributes such as bandwidth and disk space have a much larger domain; hence, this problem is unrealistic. However, it is not useful to advertise resources at a very high resolution as it significantly increases the advertising cost and users are not interested in fine-grained queries [6, 8]. Moreover, as the real-world systems tend to oscillate between idle and busy periods [6, 17] their attribute values are not uniformly distributed through D_i. Therefore, it is not useful to explicitly expand D_i. Key transfer is another approach where an overloaded node transfers part of its index to its neighbor(s) [14-15]. This approach is somewhat effective and has a lower overhead. However, when range queries are less specific, series of nodes tends to be overloaded. Therefore, a node may not be able to transfer its load to its neighbors. When a range of nodes is overloaded, it is proposed to migrate nodes in unloaded regions of the overlay to overloaded ones and then transfer keys [14-15]. This is possible only if the key space is further divisible and distributed statistics are collected to keep track of which nodes are loaded and unloaded. In practice, it is possible to have a very large number of identical resources (e.g., 98% of ×86 nodes in SETI). Similarly, a node may receive a very large number of queries due to skewed distributions (e.g., 89% of the queries specified $CPUFree \in [40\%, 100\%]$). Such large indexes and query loads need to be split across multiple nodes using replication and/or fragmentation. In [11], it proposed to arrange resource attributes on a logical Range Search Tree (RST) that is mapped to a DHT. Each node in the RST is represented as a load-balancing matrix that is expanded or contracted with the load. When the index is too large, the number of columns in the matrix (fragments) is expanded. The number of rows (replicas) is expanded when the query load is too high. All fragments and replicas are mapped to the DHT. A dynamic hash function is proposed to determine which fragment or replica to query. However, as the size of load balancing-matrix changes hash function also changes therefore needed to be informed to all the nodes. Hence, this solution is more suitable for relatively stable networks with immutable resources. Moreover, locality of attribute values is lost when the RST is mapped to the DHT consequently increasing query cost to $O(N \log N)$. Therefore, existing load balancing solutions do not work efficiently under real workloads.

2.3 Problem Statement

Future RD solutions will apply a fixed or dynamic threshold while advertising resources, as it reduces the cost of advertising and users do not specify very specific queries. Applying such a threshold will lead an even unbalanced distribution of index size as $D_i \ll N$. Conversely, this can be used to reduce the query cost, as the number of nodes along the ring does not need to exceed the largest D_i. Then by adding fragments and replicas orthogonal to the ring (contrary to the common practice of adding along the ring), we can balance both the index size and query load without increasing the query cost.

Consider a ring-like overlay with a set of $\mathbf{N}$ nodes indexing $\mathbf{R}$ resources. Each $r \in \mathbf{R}$ is willing to contribute some index capacity I^r_{Cap} and query capacity Q^r_{Cap}. These capacities are typically determined using several factors such as the computing power, memory, bandwidth, or energy of a resource, or amount of resources that a user is willing to contribute to the P2P system. We assume I^r_{Cap} and Q^r_{Cap} are measured in terms of number of resources and messages, respectively. Our goal is to find a solution that minimizes the query resolution latency on a ring-like overlay while satisfying the node capacity constraints. If we assume that the time required to resolve a query within a DHT node is small compared to the network latency and each overlay link has approximately the same latency, then the problem can be restated as minimizing the number of hops required to resolve a query which is given in (3). Given that users determine ($u_i - l_i$) and $\mathbf{q_A}$, and $\mathbf{D_i}$ is fixed, problem further reduces to minimizing N as it is the only system-level parameter (h_i depends on N). Therefore, our goal is to minimize N while satisfying the node capacity constraints.

3. Handling Single-Attribute Resources

Fig. 2 illustrates three consecutive nodes ($i - 1$, i, $i + 1$) on the overlay ring shown in Fig. 1. Let k_i be the key of i-th node. Histograms indicate the number of resources mapped to a given key. Let $\mathbf{I}^i$ be the set of resources indexed at i. Five range queries q_1 to q_5 are indicated as scatted lines. For example, q_1 starts at node i and ends at $i + 1$. While q_4 starts at $i - 1$ and suppose to end at $i + 1$, it terminates at i as the desired number of resources are found. While a query q moves from one node to another it appends matching resources to the query. Let k_l represents the key generated by applying a LPH function to the lower bound l_i of a range query. Similarly, let k_u be the hash value of u_i. Query resolution starts at the successor node of k_l (Fig. 1). For example, q_1 and q_2 can be considered as coming directly into node i from the overlay network. Set of such queries is defined as *In queries* ($\boldsymbol{Q^i_{In}}$) where $q \in \boldsymbol{Q^i_{In}}$ when $k_l \in (k_{i-1}, k_i]$. A query that is answered or reaches k_u goes out of the node, e.g., q_2 to q_4 goes out from i. Set of such queries is defined as *Out queries* ($\boldsymbol{Q^i_{Out}}$), i.e., $q \in \boldsymbol{Q^i_{Out}}$ when $k_u \in (k_{i-1}, k_i]$ or q is resolved by i. Some queries are forwarded from the predecessor, e.g., q_3 to q_5 are forwarded from node $i - 1$ to i. Set of such queries is referred to as *Forward queries* ($\boldsymbol{Q^{i-1}_{Fwd}}$), i.e., $q \in \boldsymbol{Q^{i-1}_{Fwd}}$ when $[k_l, k_u] \cap (k_{i-1}, k_i] \neq \phi$ and $k_l \leq k_{i-1}$. Therefore, the query load on a node is the sum of *In* and *Forward* queries received within a given time interval t. Our objective is to maintain the index size and query load on a node i within its bounds, i.e., $I^i \leq I^i_{Cap}$ and $Q^i_{In} + Q^{i-1}_{Fwd} \leq Q^i_{Cap}$. We first discuss the heuristics in the content of single-attribute range queries and then in the following section they are extended to support multi-attribute range queries.

3.1 Heuristic 1 – Prune

Consider the query distribution illustrated in Fig. 3(a), which is derived from range queries for *CPUSpeed*, *CPUFree*, and *DiskFree* in PlanetLab. Such a distribution arises when users specify a large ($u_i - l_i$) and m. Suppose this range is covered by four nodes a, b, c, and d and queries start either at a or b (mostly at b) and terminate at d. c is not answering any queries ($Q^i_{Out} = 0$) and merely forwards them to its successor d ($Q^b_{Fwd} = Q^c_{Fwd}$). This occurs when a node does not index any resources or indexed resources are insufficient to answer a given query. It is desirable to remove c from the ring as it helps to reduce the number of hops a query has to travel. If c indexes any resources, they have to be moved to b or d before leaving. A node i may pick its successor ($i + 1$) to move keys when $I^i + I^{i+1} \leq I^{i+1}_{Cap}$ or may pick the predecessor ($i - 1$) when $I^i + I^{i-1} \leq I^{i-1}_{Cap}$. However, removing c does not increase the query load on either b or d. Moving index to the successor is preferred as it reduces the changes to the overlay, e.g., d's key will not change when c moves its index while b's key need to be changed. Moreover, the query bandwidth requirement of d also reduces (see [8] for proof).

Let us now consider the query distribution in Fig 3(b). Such a distribution arises when intermediate nodes are able to answer queries completely (e.g., q_2 and q_4 in Fig. 2) or k_u is reached (e.g., q_3). It is still useful to remove nodes that do not answer a large number of queries to further reduce the query cost. For example, nodes a, b, and d are good candidates as they do not answer many queries. Hence, we remove a node from the ring when the number of *Out queries* is below a given threshold Q^i_{Thr} (i.e., when $Q^i_{Out} < Q^i_{Thr}$). However, we now need to be aware of both the index size and query load transferred to a node's successor/predecessor to prevent them from being overloaded. For example, if node i is removed, its successor $i + 1$ will receive three additional queries q_2, q_3, and q_4 which are in $\boldsymbol{Q^i_{Out}}$.Thus, $i + 1$ can handle i-th node's load only when $I^i + I^{i+1} \leq I^{i+1}_{Cap}$ and $Q^{i+1}_{In} + Q^i_{Fwd} + Q^i_{Out} \leq Q^{i+1}_{Cap}$. If the index is moved to the predecessor, $i - 1$ will receive q_1 and q_2 as its key will change to k_i. Therefore, keys can be moved to $i - 1$ only when $I^{i-1} + I^i \leq I^{i-1}_{Cap}$ and $Q^{i-1}_{In} + Q^{i-2}_{Fwd} + Q^i_{In} \leq Q^{i-1}_{Cap}$. By keeping track of Q^i_{Out} a node can decide by itself whether $Q^i_{Out} < Q^i_{Thr}$ or not. However, if it indexes any resources, it needs to check with the predecessor/successor before leaving. A node will continue to remain in the ring, if its neighbors are not willing to accept its index and query load. When a node is removed from the ring, it will connect to one of the nodes in the ring and use it as a proxy to issue queries and advertise resources (similar to superpeer-based P2P systems).

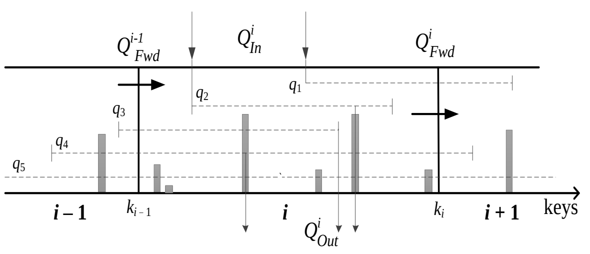

Figure 2. Series of nodes on a ring-like overlay.

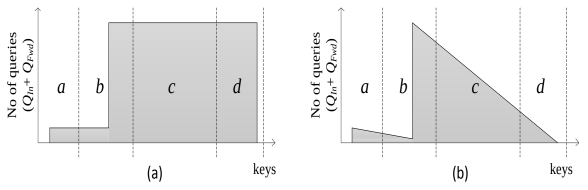

Figure 3. Two example range query distributions. Scattered lines indicate the range of keys handled by nodes.

3.2 Heuristic 2 – Key Transfer

Suppose node *c* in Fig. 3(b) is overloaded, i.e., $I^c \geq I^c_{Cap}$ and/or $Q^c_{In} + Q^b_{Fwd} > Q^c_{Cap}$. *c* can reduce the load by moving some of its keys to *b* or *d* (*d* is preferred as it requires minimum changes to the overlay). For example, one of the queries at *i* can be removed if key k_i (Fig. 2) is moved before the start of query q_1. Two queries can be removed, if it is moved even further towards k_{i-1}. Similarly, by reducing k_i index size can be also reduced. Let excess query load be $Q^i_{Excess} = Q^{i-1}_{Fwd} + Q^i_{In} - Q^i_{Cap}$ and index size be $I^i_{Excess} = I^i - I^i_{Cap}$. As Q^{i-1}_{Fwd} is fixed, only Q^i_{In} can be reduced. Therefore, it will be useful for a node to transfer its query load to its success only if $Q^i_{In} \geq Q^i_{Excess}$. To decide which keys to transfer to *i* + 1, we need to track k_l specified in each query $q \in \mathbf{Q}^{\mathbf{i}}_{\mathbf{In}}$. Let $\mathbf{K}^{\mathbf{i}}_{\mathbf{In}}$ be the set of k_ls collected from each $q \in \boldsymbol{Q}^{\boldsymbol{i}}_{\boldsymbol{In}}$ (same key may appear multiple times). Then we can find the largest key $k \in \mathbf{K}^{\mathbf{i}}_{\mathbf{In}}$ s.t. the reduced query load $Q^i_{Reduce} = countIF(\mathbf{K}^{\mathbf{i}}_{\mathbf{In}}, \text{“}\geq k\text{”}) \geq Q^i_{Excess}$. *countIF*() function counts the number of keys in $\mathbf{K}^{\mathbf{i}}_{\mathbf{In}}$ that satisfies the given condition, e.g., “≥*k*”. Similarly, index size reduced at *i* should satisfy $I^i_{Reduce} = countIF(\mathbf{I}^{\mathbf{i}}, \text{“}\geq k\text{”}) \geq I^i_{Excess}$. When k_i is moved towards node *i* – 1 some of the queries that are being currently answered by *i* will be forwarded to *i* + 1. For example, when k_i is moved up to k_l of q_1, q_2 and q_3 will be forwarded to *i* + 1. Thus, the successor's query load will be increased by $Q^{i+1}_{Transfer} = countIF(\mathbf{K}^{\mathbf{i}}_{\mathbf{Out}}, \text{“}\geq k\text{”})$. Where $\mathbf{K}^{\mathbf{i}}_{\mathbf{Out}}$ is the set of keys that queries terminate at node *i*. The index size of *i* + 1 will also increase by I^i_{Reduce}. Therefore, a subset of the keys can be transferred to the successor only when $Q^{i+1}_{In} + Q^{i+1}_{Fwd} + Q^{i+1}_{Transfer} \leq Q^{i+1}_{Cap}$ and $I^{i+1} + I^{i+1}_{Transfer} \leq I^{i+1}_{Cap}$. If successful, after transferring keys *i*-th node's key is set to $k_i = k - 1$. If the successor is unable to accept the load, the predecessor can be tried. However, the process is reversed where load reduced on *i* is determined by $\boldsymbol{Q}^{\boldsymbol{i}}_{\boldsymbol{Out}}$ and load transferred to *i* – 1 is determined by $\boldsymbol{Q}^{\boldsymbol{i}}_{\boldsymbol{In}}$.

3.3 Heuristic 3 – Key Transfer to New Nodes

In range query systems, it is possible for a range of nodes to be overloaded. Therefore, transferring keys to either the predecessor or successor may not be possible. Given that a node is aware of *In*, *Out*, and *Forward* loads, it can determine whether it would be useful to add a new node between its current successor or predecessor. Adding a successor is possible and useful if $Q^i_{In} \geq Q^i_{Excess}$ and there is enough key space between *i* and *i* + 1 (i.e., $k - k_{i-1} > 1$, where *k* is determined from heuristic two). Query load transferred to the new successor is $Q^{new}_{Transfer} = countIF(\boldsymbol{K}^{\boldsymbol{i}}_{\boldsymbol{Out}}, \text{“}\geq \text{k”}) + Q^i_{Fwd}$ and its index size is $I^{new}_{Transfer} = countIF(\mathbf{I}^{\mathbf{i}}, \text{“}\geq k\text{”})$. New successor's key $k_{new} = k_i$ and new $k_i = k - 1$. Similarly, a predecessor can be added when $Q^i_{Out} \geq Q^i_{Excess}$ and $k < k_i$. If the transferred load is too much for one node (i.e., $Q^{new}_{Transfer} > Q^{new}_{Cap}$ or $I^{new}_{Transfer} > I^{new}_{Cap}$), series of successors/predecessors can be added. As heuristic one removes unnecessary nodes, many nodes are not part of the ring. One of these nodes can be added when necessary by querying a special node that may keep track of those nodes, randomly picked from the nodes that are connected to a node on the ring, or found using the RD system by issuing a query. Therefore, in contrast to [15-16] our approach does not require an explicit mechanism to track and locate loaded and unloaded nodes in the overlay.

3.4 Heuristic 4 – Replicate Index on New Nodes

While second and third heuristics are effective in distributing some of load with minor overhead and modifications to the ring, they rely on the assumption that key space is divisible. However, due to skewed resource and query distributions key space is not perfectly divisible, and number of identical copies of a resource or queries for a given range can easily surpass the capacity of even the most resourceful node. Such cases can be detected using $\mathbf{I}^{\mathbf{i}}$, $\mathbf{K}^{\mathbf{i}}_{\mathbf{In}}$, and $\mathbf{K}^{\mathbf{i}}_{\mathbf{Out}}$. Query load can be spread across multiple nodes by replicating resources as shown in Fig. 4. Such a collection of nodes is called a *clique*. Then a range query needs to visit only one of the replicas along the path consequently splitting the load. To spread the load across multiple nodes, predecessor(s) need to be informed about the existence of multiple nodes. While forwarding queries, predecessor(s) may pick a successor using round robin or random load balancing policies. When a resource is advertised, it needs to be informed to all the replicas. In practice, a node can handle relatively large number of queries as most query messages will fit into a single packet and require a sequential search on the resource index. Therefore, a few replicas will be sufficient to handle the most popular key. A clique may be fully connected to reduce the cost of replication (Fig. 4).

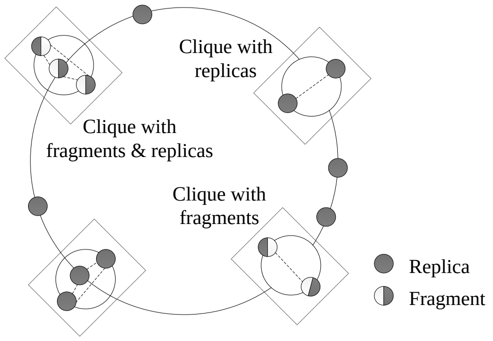

Figure 4. Fragments and replicas placed authogonal to the overlay ring.

3.5 Heuristic 5 –Fragment Index on New Nodes

Heuristic four can be applied only if the number of identical resources is within a node's index capacity. If the number of identical resources is very large, resource index needs to be fragmented across multiple nodes, where each node keeps track of a subset of the resources. Fragments are also placed orthogonal to the ring (Fig. 4). However, if resources in one fragment are insufficient to resolve a query other fragments need to be searched before going to the successor. Most queries are unlikely to be forwarded to other fragments, as resource indexes tend to be relatively large in practice, e.g., only 1 MB is required to index over 10,000 resources each with 20 attributes (assuming 32-bits per v_i).

Heuristics are triggered based on the local statistics collected by a node hence have a lower overhead and can be executed independently and distributedly. It is desirable to deploy all the heuristics within a node as each heuristic addresses a specific concern. Moreover, by applying them in the presented order, an efficient and scalable RD solution can be developed. For example, a node that is not pruned by the first heuristic may have a large query load and/or index size. Hence, it is desirable to evaluate rest of the heuristics. Second heuristic tries to balance the load by moving the keys with minimum disruption to the ring. Third heuristic is useful when nodes on the ring are not sufficient to handle the load. However, there is some cost in adding a new node to the overlay. Fragmentation and replication handle cases of extreme loads but introduce even more changes to the overlay. Thus, by applying the heuristics in the presented order query performance can be improved while reducing the cost of overlay maintenance and key movement. Histograms can be used to keep track of $\boldsymbol{I^i}$, $\boldsymbol{K^i_{In}}$, and $\boldsymbol{K^i_{Out}}$ and they will consume only a small amount of memory as the expected number of distinct attribute values is relatively small. Q^i_{In} and Q^i_{Out} may be calculated from the histograms or separate counters may be used. Another two counters are required to keep track of Q^{i-1}_{Fwd} and Q^i_{Fwd}. Therefore, heuristics are triggered based on the local statistics and communicate with the neighbors only when a node is overloaded. Heuristics may be evaluated and executed periodically or when counters reach the capacities of a node. A clique may include both the fragments and replicas (see Fig. 4). If the existence of fragments and replicas are informed to predecessors, Q^i_{In} can be equally distributed across nodes on a clique. Therefore, notification messages can be sent to potential predecessors similar to that in Chord [12]. However, in practice only the close by predecessors need to be informed as they forward most of the overlay messages. We do not anticipate a large increase in overlay routing entries as cliques are small. While heuristics two is presented in [15] and three is presented in [15-16], we utilize them more efficiently in our solution by being aware of the capacities of nodes and eliminating the need to collect distributed statistics. Placing replicas and fragments orthogonal to the ring has a lower overhead than [11], as our approach does not require the hash function to be changed with the load and informing it to all nodes in the system.

4. Handing Multi-Attribute Resources

Five heuristics are directly applicable for multiple rings [5] or partitioned ring [2, 4] based solutions as they maintain separate resource indexes for each attribute type. When multiple virtual rings corresponding to different attributes are mapped to the same address space as in [3], a node may have to index the same resource multiple times under different attributes. In such cases, the index may be compressed by removing duplicates entries of the same resource. Therefore, moving a key may not really move an indexed resource as others keys used to index the resource may be still within the range of the node. This problem can be overcome by modifying the *countIF()* function to take into account the multiple keys used to index the same resource.

TABLE II. WORKLOADS USED IN SIMULATIONS.

Workload	Resources	Queries
File sharing	100,000 copies of 10,000 distinct files, ~Zipf's(0.7) [18]	Case 1 – ~Zipf's(0.5), Case 2 – ~Zipf's(1.0). Query arrival ~exponential(2 min).
CPU speed	CPU speed of 100,000 randomly sampled nodes from SETI@home. Can be approximated by ~N(2.36, 0.28).	Pulse-like queries derived from PlanetLab. Use empirical CDF to generate ranges of attribute values. Query arrival ~exponential(2 min).
CPU free	Case 1 – A synthetic dataset of 100,000 CPU free values derived using linearly-interpolated empirical CDF from PlanetLab. Case 2 – Case 1 dataset inverted as $x(t) = 100\% - x(t_0)$ at 600 sec	Pulse-like queries derived from PlanetLab. Use empirical CDF to generate range of attribute values. Query arrival ~exponential(2 min).
PlanetLab	527 PlanetLab node trace with 12 static & 12 dynamic attributes. Also consider 250, 750, 1000 node traces generated using [9].	PlanetLab – Synthetic trace generated using empirical CDF derived from q_A, popularity of attributes, $[l_i, u_i]$, and m [9]. ~exponential(10 sec).

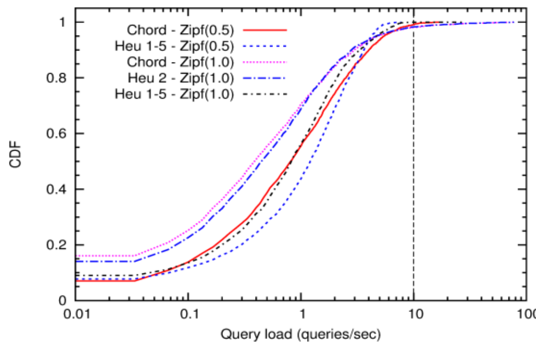

Figure 5. Query load distribution of file sharing workloads after steady state.

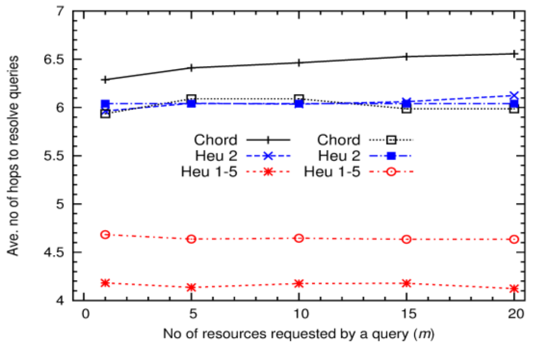

Figure 6. Average hop count required to resolve queries after steady state.

5. Simulation Setup

A discrete event simulator is developed to demonstrate the effectiveness of the proposed heuristics. See [8] for the source code. Chord is used as the underlying overlay as it supports keeping multiple fingers to successors. For multi-attribute resources, we assume multiple virtual rings are mapped to the same address space and queries are issued only to the most selective attribute [3]. Four single and multi-attribute workloads are derived using real data from P2P file sharing, PlanetLab, and SETI@home and described in Table II. It is known that both the number of queries for a file and copies of a file follow a Zipf's-like distribution [18]. Hence, with the first workload we demonstrate the applicability of heuristic under skewed resources and point queries. I^i_{Cap} = 500 entries, Q^i_{Cap} = 10 queries/second, and $Q^i_{Thr} = 0.1Q^i_{Cap}$. Rest of the workloads assume Q^i_{Cap} = 50 queries/second as range queries tend to visit many nodes consequently increasing the query load on a node. *CPUSpeed* dataset can be approximated by a Gaussian distribution. *CPUFree* dataset is skewed and most nodes were idle. A node trace from PlanetLab is used as the multi-attribute dataset. As the dataset is relatively small for this workload I^i_{Cap} = 100. We start the network with R/I^i_{Cap} nodes as the network needs to have at least this many nodes. Predecessors select fragments or replicas using round robin scheduling. Heuristics are evaluated every 30 seconds. To prevent the heuristics from responding to minor variations in index size and query load, Exponentially Weight Moving Average (EWMA) of counter values are used to trigger a heuristic. Results are based on ten samples with different random seeds.

6. Performance Analysis

We first analyze the single-attribute workloads in detailed and then present the results for multi-attribute workload. As our solution will be better than any solution that adds all the nodes to the overlay (as N is reduced), we compare our results with a Chord overlay with the same number of nodes. We also compare results with second heuristic (*Heu 2*) as it can be readily implemented on top of Chord. Heuristics three to five are not directly comparable, as they need specific mechanisms such as special nodes to track loads and dynamic hash functions. For the file sharing workload, during the steady state all three solutions had approximately the same average hop count of 5.7. Fig. 5 shows the distribution of query load. It can be seen that when all five heuristics (*Heu 1-5*) are combined the P2P system was able to stay within the allocated query capacity (indicated by a vertical scattered line). While *Heu 2* did marginally better than having only the Chord ring, one of the nodes still had to handle the query load for the most popular file. Similarly, it was also observed that all five heuristics were able to balance the index size. When Zipf's parameter $\alpha = 1.0$, 301 nodes were added to the network to handle the highly skewed query load and moderately skewed replicas of files. 265 nodes were added when $\alpha = 0.5$ (moderately skewed). Among the 301 nodes, 257 of them were placed along the ring and the rest were placed orthogonal to the ring. However, reduction in query cost was not noticeable as cost of point queries is log *N*. Hence, our solution is able to achieve comparable performance for point queries while balancing both the index and query load.

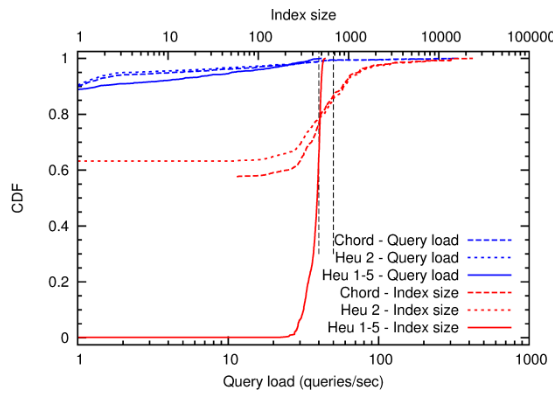

Figure 7. Index size and query load distribution of free CPU workload after steady state. $m = 20$.

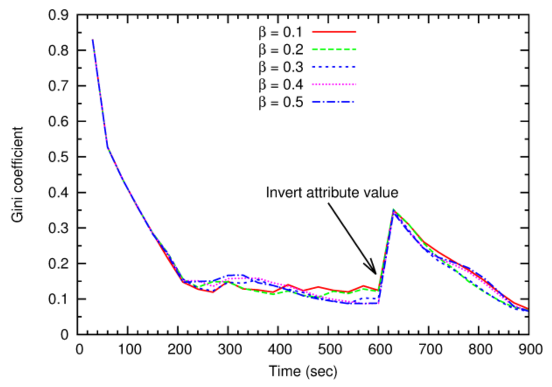

Figure 8. Variation in Gini coefficient of index size distribution of CPU free workload with time. $m = 20$.

Fig. 6 compares the query cost under *CPUSpeed* and *CPUFree* workloads with increasing *m*. *Heu 1-5* reduced the query cost of *CPUSpeed* dataset by 37% and *CPUFree* dataset by 23% compared to a Chord. A linear increase in the query cost of *CPUSpeed* dataset is observed as the v_i's are spread around following a Gaussian distribution. However, the rate of increase in *Heu 1-5* is lower compared to other two solutions. As the *CPUFree* dataset is highly skewed, a large number of free resources can be found by visiting few nodes. Hence, query cost does not increase noticeably with increasing *m*. Fig. 7 shows the load distribution for *CPUFree* dataset. It can be seen that 99% and 91% of the nodes were able to stay within the allocated query and index capacity using *Heu* 1-5, respectively. The largest index in *Heu 1-5* had 556 entries while the other two solutions had 23,733 entries each. Similar trends were also observed for *CPUSpeed* workload. Heuristics are triggered when EWMA of a counter exceeds a given threshold. However, the weighting factor β used to calculate EWMA determines how fast the system gets stabilized and its cost. Fig. 8 illustrates the inequality among index sizes of nodes measured using the Gini coefficient [15] (lower the better). When $\beta = 0.1$ system is biased towards long-term trends hence retain the system in a stabilized state. Whereas large β values quickly respond to short-term trends. However, large β values are not suitable as they have a higher cost as keys are constantly being moved and overlay is updated, e.g., 1.6% more messages related to load balancing were generated when $\beta = 0.5$ compared when $\beta = 0.1$. It is known that production systems experience sudden changes in availability of resources [8, 17]. Therefore, we invert the *CPUFree* values of resources at 600 s (as explained in Table II) to measure the responsiveness of *Heu 1-5* for such rapid changes. Query distribution was not changed, as it is not known whether user queries change in response to such rapid changes in resources. Fig. 8 shows that the system goes back to the original state within ~240 s when $\beta = 0.3$. Thus, *Heu 1-5* are also adaptable to rapid changes in attribute values. $\beta = 0.3$ generated 7% less messages related to load balancing compared to when $\beta = 0.1$ (by 900 s). Therefore, we use $\beta = 0.3$ for rest of the performance analysis as it has a balanced load distribution, lower response time, and lower cost.

Fig. 9 illustrates the query cost with increasing number of PlanetLab nodes. As the number of nodes in the system increases, both *R* and *Q* increase (because the query rate of each node is fixed). Moreover, as *A* increases, the same resource is mapped to many overlay keys hence resources are spread over a large address space. Hence, more hops are required to resolve queries. Furthermore, queries for different attributes are issued to different ranges in the address space. Therefore, to balance the load more nodes are added to different address ranges in the ring. Consequently, query cost increases, as is it proportional to the number of nodes along the ring. Alternatively, though Chord and *Heu 2* have the same number of nodes in the ring, they are uniformly spread around the ring consequently reducing the number of nodes that an average query needs to go through. This is the reason that the cost of resolving multi-attribute resources using *Heu 1-5* is higher than Chord and *Heu 2* with a similar number of nodes. However, worst-case path length of Chord and *Heu 2* is higher than *Heu 1-5* (see

Fig. 9(b)), as all the nodes are placed along the ring compared to the fourth and fifth heuristics which place some of the nodes orthogonal to the ring. Fig. 10 illustrates that *Heu 1-5* can effectively balance the load under multiple attributes as well. Therefore, proposed heuristics enable discovery of real-world resources with lower overhead while balancing the index size and query load. Moreover, they rely on local statistics, communication among clique members and predecessors/successors, and do not require dynamic hash functions.

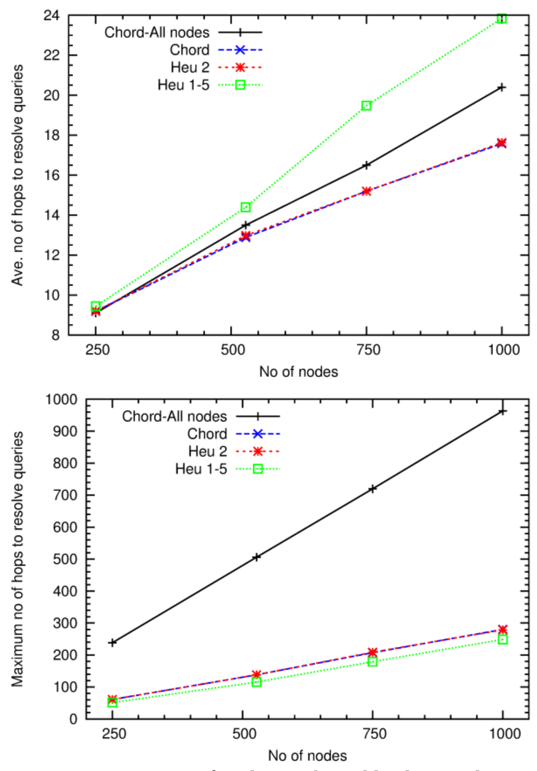

Figure 9. Query cost for PlanetLab workload at steady state: (a) Average query cost; (b) Maximum query cost.

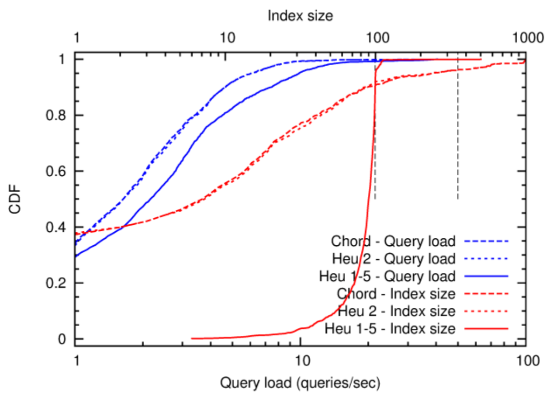

Figure 10. Index size and query load distribution of PlanetLab dataset. R = 1,000.

7. Summary and Future Work

Five heuristics for efficient P2P-based multi-attribute resource discovery is presented. Heuristics rely on local statistics to capture the complex characteristics of real-world resources and queries and try to retain only the nodes that answer queries in the overlay. Resource index is transferred among existing and new nodes to maintain the index size and query load of a node within its capacity. Simulation-based analysis demonstrates their ability to reduce the query cost, balance the load, and adapt to rapid changes in attribute values. We are currently extending the solution to also balance the load due to frequent advertising of dynamic resources and messages forwarded by overlay nodes.

Acknowledgment

This research is supported in part by the Engineering Research Center program of the National Science Foun-dation under award number 0313747.

References

[1] H. M. N. D. Bandara and A. P. Jayasumana, "Collaborative applications over peer-to-peer systems – Challenges and solutions," Peer-to-Peer Networking and Applications, vol. 6, no. 3, pp. 257–276, 2012.

[2] H. Shen, A. Apon, and C. Xu, "LORM: Supporting low-overhead P2P-based range-query and multi-attribute resource management in grids," Proc. 13th Int'l Conf. on Parallel and Distributed Systems, pp. 1–8, 2007.

[3] M. Cai et al., "MAAN: A multi-attribute addressable network for grid information services," J. Grid Comput., pp. 3-14 2004.

[4] J. Albrecht et al., "Design and implementation tradeoffs for wide-area resource discovery," ACM Trans. Internet Technology, vol. 8, no. 4, pp. 113-124, 2008.

[5] A. R. Bharambe, M. Agrawal, and S. Seshan, "Mercury: Supporting scalable multi-attribute range queries," Proc. ACM SIGCOMM, pp. 353-366, 2004.

[6] H. M. N. D. Bandara and A. P. Jayasumana, "Characteristics of multi-attribute resources/queries and implications on P2P resource discovery," Proc. 9th ACS/IEEE Int'l Conf. on Computer Systems and Applications, pp. 173-180, 2011.

[7] E. M. Heien, D. Kondo, and D. P. Anderson, "A correlated resource models of Internet end hosts," IEEE Trans. Parallel and Distrib. Syst., vol. 23, no. 6, pp. 977–984, 2012.

[8] H. M. N. D. Bandara, "Enhancing collaborative peer-to-peer systems using resource aggregation and caching: A multi-attribute resource and query aware approach," PhD Dissertation, Colorado State Univ., Fall 2012. http://hdl.handle.net/10217/78733

[9] H. M. N. D. Bandara and A. P. Jayasumana, "On characteristics and modeling of P2P resources with correlated static and dynamic attributes," Proc. IEEE GLOBECOM, pp. 1-6, 2011.

[10] H. M. N. D. Bandara and A. P. Jayasumana, "Evaluation of P2P resource discovery architectures using real-life multi-attribute resource and query characteristics," Proc. IEEE CCNC, pp. 634-639, 2012.

[11] J. Gao and P. Steenkiste, "An adaptive protocol for efficient support of range queries in DHT-based systems," Proc. 12th IEEE Int'l Conf. on Network Protocols, pp. 239 –250, 2004.

[12] I. Stoica et al., "Chord: a scalable peer-to-peer lookup service for Internet applications," Proc. ACM SIGCOMM, pp. 149-160, 2001.

[13] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load balancing in dynamic structured P2P systems," Proc. IEEE INFOCOM, pp. 2253–2262, 2004.

[14] H. M. N. D. Bandara and A. P. Jayasumana, "Resource and query aware, peer-to-peer-based multi-attribute resource discovery," Proc. 37th IEEE LCN '12, pp. 276-279, 2012.

[15] I. Konstantinou, D. Tsoumakos, and N. Koziris, "Fast and cost-effective online load-balancing in distributed range-queriable systems,"IEEE Trans. Parallel and Distrib. Syst., vol. 22, no. 8, pp. 1350 – 1364, 2011.

[16] Q. H. Vu, B. C. Ooi, M. Rinard, and K.-L. Tan, "Histogram-based global load balancing in structured peer-to-peer systems," IEEE Trans. Knowledge and Data Engineering, vol. 21, no. 4, pp. 595–608, 2009.

[17] A. Iosup and D. Epema, "Grid computing workloads: Bags of tasks, workflows, pilots, and others," IEEE Internet Comput., vol. 99, pp. 19-26, 2010.

[18] S. B. Handurukande et al., "Peer sharing behaviour in the eDonkey network, and implications for the design of server-less file sharing systems," Proc. ACM European Conf. on Computer Systems, 2006.