Probabilistic Best-fit Multi-dimensional Range Query in Self-Organizing Cloud

Sheng Di, Cho-Li Wang, Weida Zhang, Luwei Cheng
Department of Computer Science
The University of Hong Kong
Pokfulam Road, Hong Kong
{sdi, clwang, wdzhang, lwcheng}@cs.hku.hk

***Abstract*—With virtual machine (VM) technology being increasingly mature, computing resources in modern Cloud systems can be partitioned in fine granularity and allocated on demand with "pay-as-you-go" model. In this work, we study the resource query and allocation problems in a Self-Organizing Cloud (SOC), where host machines are connected by a peer-to-peer (P2P) overlay network on the Internet. To run a user task in SOC, the requester needs to perform a multi-dimensional range search over the P2P network for locating host machines that satisfy its minimal demand on each type of resources. The multi-dimensional range search problem is known to be challenging as contentions along multiple dimensions could happen in the presence of the uncoordinated analogous queries. Moreover, low resource matching rate may happen while restricting query delay and network traffic. We design a novel resource discovery protocol, namely Proactive Index Diffusion CAN (PID-CAN), which can proactively diffuse resource indexes over the nodes and randomly route query messages among them. Such a protocol is especially suitable for the range query that needs to maximize its best-fit resource shares under possible competition along multiple resource dimensions. Via simulation, we show that PID-CAN could keep stable and optimized searching performance with low query delay and traffic overhead, for various test cases under different distributions of query ranges and competition degrees. It also performs satisfactorily in dynamic node-churning situation.**

I. Introduction

Cloud computing [1], [2] has emerged as a compelling distributed paradigm with elastic VM's resource isolation technology [3], [4], [5]. Resources could be elastically partitioned and reassembled to meet users' actual needs [2], [6], [7]. Such a dividable resource allocation scheme is gaining more attention in recent years. As an example, the *proportional share model* (PSM) [8] allows resource shares be allocated proportional to users' assigned bids, and it has been leveraged in several Cloud systems [9], [10], [11].

In this work, we aim to design an efficient resource discovery protocol in a Self-Organizing Cloud (SOC) such that each individual host could autonomously find a qualified volunteer computer on the Internet for its task's execution via multi-dimensional range query. Every joined host, either a public server or a desktop computer, serves as an individual node on a structured P2P overlay network. To perform a multi-dimensional range query, the task's resource demand is expressed as a vector specifying its minimal requirements along each resource type (e.g., CPU, memory, network, storage) such that the task can be finished on time. The discovery process is conducted by propagating the query messages hop by hop towards the peer nodes that keep the qualified resource records on different attribute dimensions. Once a qualified resource node is found and determined, its split resource shares will be allocated to the task. However, as two users may simultaneously request for nodes with similar resource types and capacities, the same candidate nodes could be returned as their query results without proper coordination. This may cause the task schedulers to dispatch and run their tasks on the same node, resulting in resource contention problem and making all of them cannot meet expected execution times. Such an issue is very challenging due to the fact that the overall performance of any virtual execution environment is closely related to its allocated resource shares along many dimensions, so that we cannot use existing single-dimensional contention-free models [12].

We consider Distributed Hash Table (DHT) (such as [13], [14]) the most suitable P2P network structure due to its predictable logarithmic hops on message delivery for routing each query to its destination. However, supporting efficient range queries in DHT remains a difficult problem as the ordinary hash function of DHT protocols makes it hard to preserve the original order relationship among the stored data records. Thus, many existing solutions [15], [16], [17], [18], [19], [20], [21], [22] have tried to leverage tree- or ring-based order-preserving hash functions to search range-matched data records. Yet, these approaches either suffered longer query delay time or the cost for maintaining these extra hash functions is rather high. On the other hand, they spread multiple messages for each query request and try to find matched results as many as possible (i.e. all qualified records). This may easily cause heavy network traffic and lead to low scalability. For instance, if a query demands CPU$\geq$4GFlops and all the CPU records are distributed within [0, 8GFlops] in a DHT space, about half of nodes in the network need to respond the request.

We endeavor to bound the network traffic overhead with the growth of query scale. Unlike the parallel query solutions used by existing works [16], [22], we strictly limit each query request to just issue single query message to be

routed on the network and return the first k matched results. However, under such a single-message query constraint, the chance of finding the candidate nodes with qualified multi-dimensional resources for each request could be much lower than that of the aggressive parallel query solutions. This problem is especially serious in DHT space because the resource states' records may not be uniformly distributed, but intensively stored in only a few small-zone nodes. Thus, how to design a routing mechanism to make each single-message query able to effectively search qualified resources should be carefully studied, otherwise the widely-dispersed resources cannot be fully utilized.

Consequently, we propose a new multi-dimensional range query protocol, namely Proactive Index Diffusion CAN (PID-CAN), on the basis of Content Addressable Network (CAN) overlay [14]. The reason why we choose CAN [14] as the basis of our design is due to its intrinsic multi-dimensional routing support, which can be easily extended by many applications. In PID-CAN, upon receiving zone-overlapped state messages, any node will spread its identifier (a.k.a. index) backward along multiple dimensions over CAN to notify a few other randomly selected nodes (a.k.a. index nodes), whose distances are 2^k hops. We also study the index-diffusion efficiency under different randomized index-node selection policies. Based on our optimized indexing strategy, we devise a randomized query routing mechanism, which could effectively restrict query contentions. Moreover, each query from anywhere of the network can find its best-fit resources such that the qualified-resource matching rate is significantly improved and the contention on multi-dimensional resources is immensely restricted.

The rest of the paper is organized as follows. In Section II, we formulate the best-fit resource query problem, by aiming to optimize task execution in SOC. In Section III, we formally describe the novel protocol, namely Proactive Index-Diffusion CAN (PID-CAN). In Section IV, we evaluate our design via simulation, with respect to throughput ratio, message delivery cost, scalability, failed task ratio, fairness, etc. The related works are discussed in Section V. We conclude and present future work in Section VI.

II. Problem Formulation

In the Self-Organizing Cloud, each user contributes his/her computer to execute tasks submitted by local users or migrated from other nodes. Each node has a task scheduler to determine if submitted tasks should be executed locally or remotely for better resource utilization.

Assume the Self-Organizing Cloud is constructed by connecting n host machines on the Internet, denoted as p_i (i=1, 2, $\cdots$, n). Let c_i denote the *resource capacity vector* of p_i, where c_i=(c_{i1}, c_{i2}, $\cdots$, c_{id})^T, and d refers to the number of physical resource types owned by p_i.

Let m_i denote the total number of tasks submitted to the node p_i, and a task submitted to the node p_i can be expressed as t_{ij}, where j=1,2,$\cdots$,m_i. For each task, its user needs to specify an *expectation vector*, denoted $e(t_{ij})$, which indicates the minimum resource demand on each resource type for completing the task within expected time.

To improve resource utilization, the available resources could be time shared by multiple running tasks. We further denote node p_i's *availability vector* as a_i=c_i−l_i, where l_i is an *aggregated load vector* indicating the minimal load consumed by the current tasks running at p_i using each type of resources, i.e., l_i=$\sum_{j=1}^{s_i} e(t_{ij})$, where s_i means the number of tasks scheduled onto p_i. To make best use of underlying resources, we adopt the proportional share model (PSM) [8] for resource allocation. That is, the actual resource amount vector (denoted $r(t_{ij})$) allocated to task t_{ij} on node p_i will be determined by Equation (1).

$$r(t_{ij}) = \frac{e(t_{ij})}{l_i} \cdot c_i \tag{1}$$

For example, on a node p_r, if we assume that there were three running tasks with expected CPU speed and memory size being {2 GFlops, 100 M}, {3 GFlops, 200 M}, and {4 GFlops, 300 M} respectively, and p_r's capacity vector c_r is {13.5 GFlops, 1200 M}. According to PSM, the three tasks could actually get {3 GFlops, 200 M}, {4.5 GFlops, 400 M}, and {6 GFlops, 600 M} until a new task is scheduled on this node or any of the running tasks completes its work.

Based on the proportional resource sharing policy, if the number of running tasks on a node is not carefully controlled, it is possible that each task's resource share may become smaller than its minimum demand. To guarantee the expected completion time for all running tasks, when a task τ is submitted to node p_i, the node selected for executing τ (denoted as p_r) found by the task scheduler at node p_i must satisfy Inequality (2).

$$a_r \succeq e(\tau) \tag{2}$$

In order to make any node able to quickly locate any other qualified resource nodes with multi-dimensional attributes within predictable delay, all the nodes are organized in a CAN [14] overlay. In CAN, each node is connected to a few other nodes as its neighbors and cooperatively maintain the global information by periodically exchanging resource usage states with its neighbors. To search a qualified node for task execution, a multi-dimensional range query is performed by forwarding the *expectation vector* over CAN to locate a candidate node with available resource capacities that satisfy Inequality (2). In order to control the query traffic overhead (which is determined by the number of messages constructed per query), we strictly limit every query can just issue one message and the number of its routing hops is also expected to be minimized.

The effect of multi-dimensional range query will be evaluated by the *failed task ratio* (denoted as F-Ratio(t) which refers to the ratio of the number of tasks that cannot find any qualified nodes to the total number of generated

tasks ($\sum_{i=1}^{n} m_i$) until a specific time point t) and *system throughput ratio* (denoted by T-Ratio(t), calculated as the ratio of the number of finished tasks to the total number of generated tasks until time point t). Smaller F-Ratio implies higher effectiveness of querying resource nodes, which may lead to fewer tasks that cannot be started (or scheduled). That is, F-Ratio directly reflects the resource matching rate of the query protocol. T-Ratio could implicitly reflect the resource contention degree delivered by the designed discovery protocol, in that bigger throughput means more tasks successfully finished, which is probably due to relatively lower degree of tasks' resource contention on selected execution nodes.

In sum, our designed query protocol should minimize the failed task ratio and maximize the system throughput ratio.

III. Proactive Index-Diffusion CAN (PID-CAN)

In this section, we present the new discovery protocol, Proactive Index-Diffusion CAN (PID-CAN), which supports efficient multi-dimensional range query in the fully decentralized self-organizing Cloud, as compared to the traditional CAN [14] that could perform exact-match query but cannot find qualified records based on a specified range. First, we discuss an improved CAN [14], called Index-Node Supported CAN (INSCAN), which will be adopted by PID-CAN. We also present the strategy for performing delay-bounded range query on INSCAN. We then show how to diffuse indexes over INSCAN to improve the resource matching rate. Lastly, we introduce possible strategies of lowering the resource contention probability.

A. INSCAN-based Range Query (INSCAN-RQ)

In traditional CAN, the search space is dynamically partitioned by all peers into multi-dimensional zones and each node is responsible for storing a set of resource information records (i.e. a_i) which match its corresponding zone. Hence, for any node along every dimension, there are a lower-bound and an upper-bound for its zone. If there is only one non-overlapped range dimension between two nodes (such as p_i and p_j) and they are adjacent at this dimension, we call them *adjacent neighbors*. If the non-overlapped range of p_i is no less than p_j's, p_i is called p_j's *positive neighbor* and p_j is called p_i's *negative neighbor*. If the ranges in all dimensions of one node are overlapped or no more than those of another node, the former is called *negative-direction node* of the latter. For example, in Fig. 1, Node 22 is Node 12's negative neighbor and Node 13's negative-direction node.

In INSCAN, every node not only includes the adjacent neighbors like traditional CAN overlay, but also a few sampled 2^k-hop-distance nodes (a.k.a. *index nodes*). The set of index nodes on each node could be updated periodically by flooding the querying messages to its neighbors along the d dimensions until reaching the edge of the CAN space. This structure enables each peer node to locate any other ones within $O(\log_2 n)$ hops in the multi-dimensional CAN space, instead of $O(n^{\frac{1}{d}})$ in the original CAN. Each node periodically detects its own availability (i.e. a_i) and routes it over INSCAN until it is completely enclosed in a multi-dimensional zone. For example, as shown in Fig. 1, If Node 6's up-to-date availability vector is $\{0.95, 0.7\}$, then the vector should be stored in node 17, whose zone fully overlaps the vector. Based on the routing rule, the state-update message delivery distance is $O(\log_2 n)$ hops.

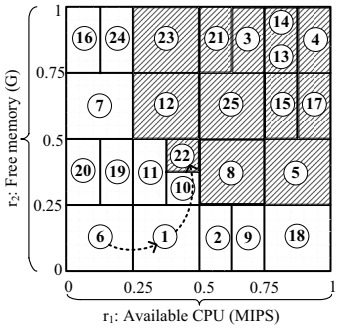


Figure 1. Routing on INSCAN-RQ: node 1 & node 18 are node 6's index nodes because of 2^k-hop distances. Assuming Node 6 renders a range-query overlapping node 22's zone, then all shaded zones need to be checked.

Based on INSCAN, it is easy to find the nodes whose zones overlap the boundary lines of the query range (Node 22, 12, 23, 8, 5 in Fig. 1) as well as all the other responsible nodes within the range (shadow area in Fig. 1), yet the heavy network traffic overhead is inevitable for getting complete range-matched results in this range. We call it INSCAN-based Range Query (INSCAN-RQ) and it is easy to prove that its query delay upperbound is $2\log_2 n$ but the network traffic per query is $\log_2 n$+N-1, where N is the total number of all responsible nodes (shadow area in Fig. 1). In order to bound query message traffic overhead, a straightforward solution is using a random-walk query routing method after locating the boundary-corner node (e.g., Node 22 in Fig. 1). However, in the situation with scarce available resources, random-walk query routing may hardly find qualified resources, significantly degrading resource matching rate.

B. Proactive Index-Diffusion Strategy

Our index-diffusion design aims to make users discover best-fit nodes with available capacities for each required resource type, with restricted query message traffic overhead.

Like the traditional CAN, each node in INSCAN is also in charge of a specific zone as a state keeper to collect all the updated state messages matching the zone. Differently, each node periodically checks the status of its cache (denoted as γ) whether it contains a set of received state messages or not. Once a node detects its cache is non-empty, it will diffuse its own identifier (such as host IP) to a few other index nodes, to make itself be discovered by other nodes around the global system. As mentioned previously, the number of hops for message delivery between a node and its index-nodes is restricted to 2^k in order to control the maintenance cost, where k=0,1,2,$\cdots$,$\left\lfloor \log_2 n^{\frac{1}{d}} \right\rfloor$. We call a node's index-

node located at its positive (negative) direction along some dimension *positive-index node* (*negative-index node*).

1) Index-Diffusion Analysis: Since the identifier can be continually propagated from index-node to index-node, any other requester node at the negative location of the index-node could quickly locate it, in turn for finding more resource records on demand. Below, we prove that each node could diffuse its index with only a few hops of recursive relay from index-node to index-node, to any of its negative-direction nodes with limited message delivery overhead.

Theorem 1: The delay complexity of relay hops for notifying any node's index to any of its negative-direction nodes is $O(\log_2 n)$, where n refers to the total number of nodes.

Main Idea: Note that $\log_2 n$=d·$\log_2 n^{\frac{1}{d}}$, so our objective is to prove the delay complexity is bounded under $d \cdot \log_2 n^{\frac{1}{d}}$. The example shown in Fig. 2 illustrates the basic idea of our proof. In this example, suppose there are r=$n^{\frac{1}{d}}$=19 nodes along each dimension, it is obvious that the top-most node (Node 1) will take the longest time, but less than $O(\log(19))$=4, to diffuse its own index. Specifically, over the first hop, Node 2, 3, 5, 9, and 17 could receive the index (Node 1's identifier). Via the second hop, Node 4, 6, 7, 10, 11, and 13 could receive the relayed index. For instance, Node 7 could receive Node 1's index forwarded from Node 5 or Node 3. With just 3 hops, most of the negative-direction nodes of Node 1 could receive its index notification.

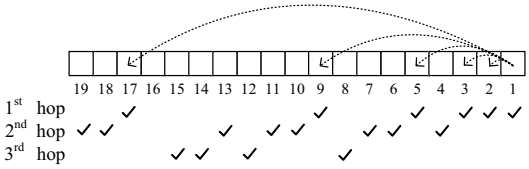
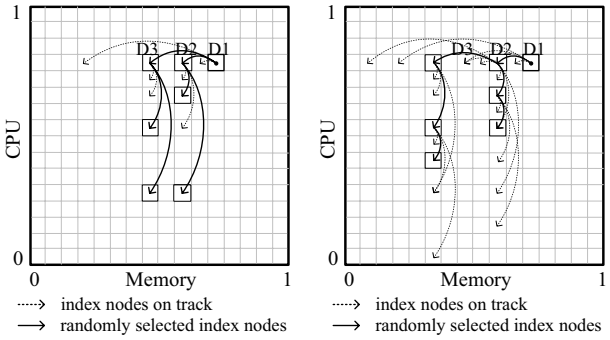


Figure 2. Quick Backward Index Diffusion

Proof: Since there are d dimensions and n nodes in total, the number of nodes along any dimension is about r=$n^{\frac{1}{d}}$. Then, as long as we prove the time cost of the top-most node diffusing its index to all of its negative-direction nodes along each dimension is no more than $[\log_2(r)]$, we could easily induce the final conclusion, i.e. $O(d \cdot \log_2 n^{\frac{1}{d}})$.

Inspired by the example shown in Fig. 2, we need to prove $\exists\ h \leq [\log_2(r)]$, such that the distance λ (i.e. the number of hops) between any two nodes along one dimension could be expressed as 2^{a_1}+2^{a_2}+$\cdots$+2^{a_h}, where $a_i \in N$. Since $\lambda < r$, if we denote λ in binary format, it is easy to observe that the number of its digits just indicates h's minimum value. For example, $(13)_{10}$=$(1101)_2$ means that 13=$2^3 + 2^2 + 2^0$ and h=3. Hence, $h \leq [\log_2(\lambda)] + 1 \leq [\log_2(r)]$. ■

Obviously, it is infeasible for peer nodes to broadcast their indexes (either their own identifiers or those of other nodes to forward) due to the considerable message delivery overhead. Suppose L negative-index nodes are selected along each dimension as the notification targets, the total number of the messages (denoted as ω) to deliver for any index is equal to L+L^2+$\cdots$+L^d=$\frac{L\cdot(L^d-1)}{L-1}$. Hence, the message



(a) Spreading Method (b) Hopping Method

Figure 3. Two Index-Diffusion Methods

overhead could be controlled by setting L to a small value. For example, if $L = 2$ and $d = 3$, the total number of messages is only 14. In other words, L has to be small constant (we always set it to 2). Then, the key issue is how to select the limited number of negative-index nodes at each index-relay hop, such that the index-diffusion could achieve the maximum efficiency. We discuss this problem in the following text.

2) Index-Diffusion Algorithms: In order to notify the indexes as broadly and efficiently as possible, our strategy adopts probabilistic theory. That is, the negative-index nodes to which an index needs to be sent are randomly selected rather than based on some fixed rules. There are two candidate solutions: (1) *spreading methods* and (2) *hopping methods*, as illustrated in Fig. 3 ($L = 2$). For the former, the L negative-index nodes along each dimension will be determined completely by the initial index-senders (Fig. 3 (a)); for the latter, the index will be forwarded from index-node to index-node along each dimension (Fig. 3 (b)). Obviously, the former suffers fewer message delivery hops, but its indexes cannot be diffused as widely as the latter's. In fact, the index delivery delay complexity of hopping method is $O(\log_2 n)$ as proved in Theorem 1. That is, the hopping method's index delivery delay is also acceptable, thus it could be considered better than the spreading method, which will be validated in our simulation.

The index-diffusion process could be realized by our *index-sender* and *index-relay* algorithms. Their pseudo-codes are shown in Algorithm 1 and Algorithm 2. We just show the pseudo-code of the hopping method, since the spreading method's can be easily converted from it.

The index-sender algorithm on each node is performed periodically, and the index message (containing the identifier) of the node will be sent out if and only if its cache is non-empty. The format of index message is {ID, *dim_NO*, *dim_TTL*}, where *dim_NO* indicates which dimension the message should be propagated to and *dim_TTL* refers to the maximum number of hops to forward along the *dim_NO*th dimension. In Algorithm 1, the initial dimension's sequence number and *dim_TTL* are set to 1 and L respectively (line 3). L is set to 2 in our experiment to limit the message

delivery overhead. *NINode* refers to a negative-index node, whose distance can just be 2^k, k=1,2,$\cdots$, $\left\lfloor \log_2 n^{\frac{1}{d}} \right\rfloor$ from the current node p_i.

Algorithm 1 INDEX-SENDER ALGORITHM

This program is periodically invoked as the current node p_i detects that it owns records.

```
1: while (TRUE) do
2:     if (cache γ is non-empty) then
3:         Construct an index message, i.e. {p_i's ID, 1, L};
4:         Randomly select an NINode along the dimension NO. 1;
5:         Send {p_i's ID, 1, L} to NINode;
6:     end if
7:     Sleep for a tiny cycle;
8: end while
```

The index-relay algorithm will be asynchronously triggered by individual nodes whenever they receive forwarded indexes from outside. Line 1∼4 is used to forward the received index message to a random negative-index node within the residual dimension TTL (i.e. q), in order to diffuse indexes along the *same* dimension. Line 5∼9 increments the relay dimension by forwarding the received index to a randomly selected negative-index node along the next dimension. In our simulation, we will show that a small L could already lead to a quite satisfactory efficacy in the resource discovery, especially due to our probabilistic design (Line 4 in Algorithm 1 and Line 2 & 7 in Algorithm 2). Upon receiving an index message, the node will store it into a list, denoted as *PIList*, which means Positive Index List.

Algorithm 2 INDEX-RELAY ALGORITHM

This program is invoked upon receiving an index $\{p_k$'s ID, j, $q\}$.

```
1: if (q − 1 > 0) then
2:     Randomly select an NINode along the dimension NO. j;
3:     Send index message {p_k's ID, j, q − 1} to NINode;
4: end if
5: if (j < d) then
6:     Construct a new index message: {p_k's ID, j + 1, L};
7:     Randomly select an NINode along the dimension NO. j+1;
8:     Send {p_i's ID, j + 1, L} to NINode;
9: end if
```

C. Contention-minimized Multi-dimensional Query

For each resource query, there are three phases in finding its qualified resources: (1) locating duty-node, (2) randomly determining index agents, and (3) randomly checking index-nodes. On requester node, a query message (a.k.a. duty-query message) is initially generated and routed to the node D_1 whose zone overlaps the user-defined expectation vector $e(t_{ij})$, and this node (D_1) is called *duty-node* (or boundary-corner node). On D_1, an index-agent list (denoted as ι) will be constructed by randomly selecting d positive neighbors (one neighbor per dimension), which are considered the reservoir of the positive-index nodes. Thereafter, node D_1 will send an index-agent message containing $e(t_{ij})$ and $\{\iota - A_1\}$ (i.e. the index agents excluding the selected one) to one index agent (Node A_1) randomly selected from ι. The index agent A_1 will assemble and propagate an index-jump message that contains a number of positive-index nodes selected from its *PIList*. Each index node in the index-jump message will be checked until enough number of qualified resource records are found. If such an index-jump message hopping cannot find enough demanded resource nodes, the message will be sent back to node A_1 and another index agent (A_2) randomly selected from ι by A_1 will be set as the next index agent. As soon as the agent node A_2 receives the new index-agent message, it will also perform the index-jump message hopping to keep searching resources.

In order to realize the resource query mentioned above, we need three individual algorithms to respectively handle the three different kinds of messages, duty-query message, index-agent message, and index-jump message. The pseudo-codes are presented in Algorithm 3, Algorithm 4, and Algorithm 5, which are driven by the corresponding arrival messages. In addition, as a set of state records about the qualified resource nodes are found at the index-nodes, the records will be enclosed in an index-jump notification message (i.e. *FoundList*, denoted as φ) and sent to the requester.

In Algorithm 3, after the duty node is located (Line 4), index agent determination will be performed (Line 5∼7).

Algorithm 3 DUTY-QUERY MESSAGE HANDLER

Suppose the program is running on current node p_i.

```
1: if (the request is not delivered but submitted to the node) then
2:     v = e(t_ij); /*assign expectation vector*/
3: end if
4: if (v is right enclosed in p_i's multi-dimensional zone) then
5:     Construct the index-agent list ι using d positive neighbors;
6:     Randomly select an index agent α from ι;
7:     Send the index-agent message {v, {ι − α}} to node α;
8: else
9:     Forward duty-query message {v} based on CAN's routing rule;
10: end if
```

When any node receives an index-agent message, Algorithm 4 will be triggered immediately. An index-jump list (denoted as $\jmath$) is built using the positive-index list (*PIList*), which was constructed by the proactive index-diffusion. Then, the index nodes will be searched hop by hop for qualified resource records stored on them (Algorithm 5).

Algorithm 4 INDEX-AGENT MESSAGE HANDLER

Suppose the program is running on current node p_i.

```
1: Randomly select a few indexes from p_i's PIList and put them in ȷ;
2: if (ȷ is not empty) then
3:     Randomly choose an index node β from the list ȷ;
4:     Send the index-jump message {v, δ, {ȷ − β}} to β;
5: else
6:     Randomly select an index agent α from ι;
7:     Send index-agent message {v, {ι − α}} to node α;
8: end if
```

On any index node, Algorithm 5 may notify the searched resources' identifiers to the requester node (Line 2∼5). If the expected number of qualified resource nodes are found, the query would be terminated (Line 15), or else, either index-jump message or index-agent message will be propagated similar to the index-agent message handler.

Algorithm 5 INDEX-JUMP MESSAGE HANDLER

Suppose the program is running on current node p_i.

1: Search the cache (i.e. γ) on p_i and put qualified records in a list φ;
2: **if** (φ is not empty) **then**
3: Send φ to the requester node;
4: $\delta = \delta$ - $|\varphi|$; /*δ refers to the expected number of qualified results.*/
5: **end if**
6: **if** ($\delta > 0$) **then**
7: **if** ($\jmath$ is not empty) **then**
8: Randomly choose next index node β from list $\jmath$;
9: Send index-jump message $\{v, \delta, \{\jmath - \beta\}\}$ to β;
10: **else**
11: Randomly select an index agent α from ι;
12: Send index-agent message $\{v, \{\iota - \alpha\}\}$ to node α;
13: **end if**
14: **end if**

We also explore another strategy, *Slack-on-Submission* (SoS), in order to further avoid the query contention among different requesters with the similar expectation vectors. As a user triggers a resource query for a task t_{ij}, its original expectation vector $e(t_{ij})$ will immediately be skewed/slacked to be a new random value $e'(t_{ij})$ subject to Formula (3), where $\preceq$ denotes componentwise inequality between two vectors and $c_{\max}$ implies the upper-bound capacity vector in the whole DHT space, which can be statistically aggregated using cached information [23]. Then, the query with $e'(t_{ij})$ will follow the basic query procedure conducted by Algorithm 3∼5. If the number of query results cannot fulfill the user's expectation, the expectation vector could be restored from $e'(t_{ij})$ to the original $e(t_{ij})$ and the search will be conducted again until finding enough expected resources.

$$e(t_{ij}) \preceq e'(t_{ij}) \preceq c_{\max} \tag{3}$$

IV. PERFORMANCE EVALUATION

A. Experimental Setting

We first built an emulated credit-scheduler (or proportional-share scheduler) in accordance with the design of XEN [24]. Then, we constructed the CAN protocol [14] using the Peersim simulation tool [25]. There are thousands of participating nodes, each with random settings (Table I) and various user tasks (Table II). Each task needs a least-qualified five-dimensional vector {computation load, I/O load, network load, disk size and memory size} to launch, and its execution time is only related to the first three resource types. Tasks' workloads are randomly generated such that their overall average execution time is 3000 seconds. We simulate the Internet communication by grouping all nodes into different LANs and two nodes across LANs have to communicate via WAN network bandwidth. By leveraging the event-driven mode under Peersim tool [25], each experiment simulates 86400 seconds (i.e. one day) using totally 4320 event cycles and the user requests (or tasks) will be periodically generated on each node based on Poisson process with 3000 seconds as its mean. Hence, the total number of tasks to process in one day on a system with 2000 nodes is about $2000 \times \frac{86400}{3000} \approx 57600$. The TTL (or age) of each state-update message is 600 seconds and the message updating cycle is 400 seconds. According to the existing experimental report [5], we set the cost (or percentage loss of total resource capacity) in maintaining one VM instance as follows: processor rate=5%, IO speed=10%, network bandwidth=5%, memory cost=5M.

Table I
SYSTEM SETTING

Parameter	Value
# of nodes	2000 $\sim$ 12000
# of processors per node	1,2,4,8
computation rate per processor	1,2,2.4,3.2 Hz (or 10MI)
I/O speed per node	20,40,60,80 MbPS
memory size per node	512, 1024, 2048, 4096 M
disk size per node	20, 60, 120, 240 Gb
LAN network bandwidth	5 $\sim$ 10 Mbps
WAN network bandwidth	0.2 $\sim$ 2 Mbps

Table II
USER TASK'S DEMAND

Parameter	Value	Parameter	Value
demand ratio λ	1, 0.5, 0.25	cpu rate	$\lambda \sim 25.6\lambda$
I/O speed	$20\lambda \sim 80\lambda$	memory size	$512\lambda \sim 4096\lambda$
disk size	$20\lambda \sim 240\lambda$	bandwidth	$0.1\lambda \sim 10\lambda$

We first analyze the pros and cons of SID-CAN (Spreading Index Diffusion over CAN) by comparing it with two other related works, Newscast gossip protocol [26] and K-Hop DHTNEIGHBOR based range-query strategy (KHDN-CAN). Newscast gossip protocol is a typical unstructured P2P solution, under which neighbors of each node are randomly changed based on the Newscast model [26] over time to enhance message diffusion range and the fan-out degree (i.e., the number of neighbors) is limited to $\log_2(n)$ to avoid excessive network traffic. In KHDN-CAN, once a state message is routed to its duty node, it will be further spread to negative CAN neighbors with K hops, such that each query can easily locate the K-hop sampled positive neighbors around the minimal-demand zone nodes, for searching the qualified resources closest to expectation vectors. KHDN-CAN can be considered RT-CAN [22] tailor-made for SOC environment, where real-time-states are stored in vectors rather than local R-Trees. KHDN-CAN can also be considered converted from INSCAN-RQ. For fairness, we make such three protocols' network traffic close to each other in experiment, by tuning the neighbor degree in Newscast gossip protocol and hop number K in KHDN-CAN.

Thereafter, we will show different results by combining various index-diffusion methods (either spreading or hopping) and various resource query methods (either non-SoS or with SoS). There are six different protocols to compare, including *SID-CAN*, *HID-CAN*, *SID-CAN+SoS*, *HID-CAN+SoS*, *SID-CAN+VD*, and *Newscast* protocol. SID-CAN and HID-CAN are short for Spreading Index Diffusion over CAN and Hopping Index Diffusion over CAN respectively. These two resource query methods focus on the original expectation vector (i.e. $e(t_{ij})$). Comparatively, SID-CAN+SoS and HID-CAN+SoS will use Slack-on-Submission (SoS), that is, the primary duty-query message

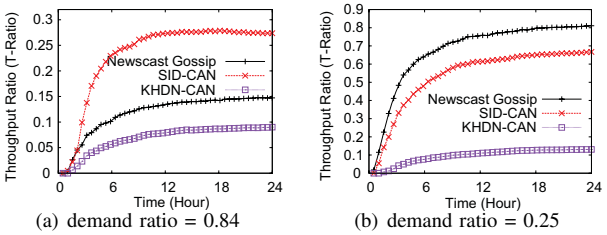
will contain the slacked expectation vector ($e'(t_{ij})$) instead of the original one. SID-CAN+VD adopts an extra virtual dimension [27] to resolve the resource competition problem.

We focus on four performance metrics, *throughput ratio (T-Ratio)*, *failed task ratio (F-Ratio)*, *fairness index*, and *scalability*. The throughput ratio is defined as the ratio of the number of finished tasks and the total number of generated tasks in the system over time. The failed task ratio refers to the value that the number of the tasks which cannot find any qualified resources divided by the number of submitted tasks. Jain's fairness index [28] (denoted φ) is commonly used to evaluate the scheduling fairness for finished tasks, and it is defined as Equation (4) (its higher value means fairer treats in executing tasks). In this formula, e_{ij} (i.e. t_{ij}'s execution efficiency) is defined as t_{ij}'s expected execution time divided by its real completion time, where the expected execution time is estimated using its load amount and the system-wide average node capacity and network bandwidth.

$$\varphi = \frac{(\sum_{i=1}^{n}\sum_{j=1}^{m_i} e_{ij})^2}{(\sum_{i=1}^{n} m_i)\cdot(\sum_{i=1}^{n}\sum_{j=1}^{m_i} e_{ij}^2)} \quad (4)$$

B. Experimental Result

With 2000 simulated nodes, the PID-CAN based on the index-spreading method (i.e. SID-CAN) outperforms other competitors (including Newscast gossip protocol and KHDN-CAN), as most of the queries request widely different resource amounts (see Fig. 4 (a)). However, it suffers sub-optimal performance as long as the requested resource amounts are not distributed widely, that is, it cannot adapt to the cases with relatively intensive range queries. Fig. 4 (b) shows that SID-CAN performs even worse than the Newscast gossip protocol if all queries are randomly distributed within a small range [0, 0.25×$c_{\max}$]).



(a) demand ratio = 0.84 (b) demand ratio = 0.25

Figure 4. Contrary Results under Different Query Ranges

The main reason why SID-CAN works sub-optimally is due to the fact that it cannot effectively process or distribute the requests evenly on the widespread resource nodes. In other words, if all resource amounts demanded by tasks are not uniformly distributed in the whole DHT space, the requests in SID-CAN are likely to compete for the same resource nodes over CAN, causing undesired hotspots.

We compare the efficiency of six different protocols with respect to various demand ratio (λ) in Fig. 5 through Fig. 7. When λ=1 (i.e. all tasks randomly demand the multi-dimensional resource amounts within the range [0, 1·$c_{\max}$]), we could observe that SID-CAN and HID-CAN as well as their SoS versions prominently outperform the other two algorithms. Newscast gossip protocol performs worst due to its completely random nature over partial-view cache. In other words, the ability of locating least satisfactory resource around the whole system acts as the major factor to impact the performance in this situation, so SoS will become redundant here. We also observe that HID-CAN performs as well as SID-CAN, which delivers the optimal result here.

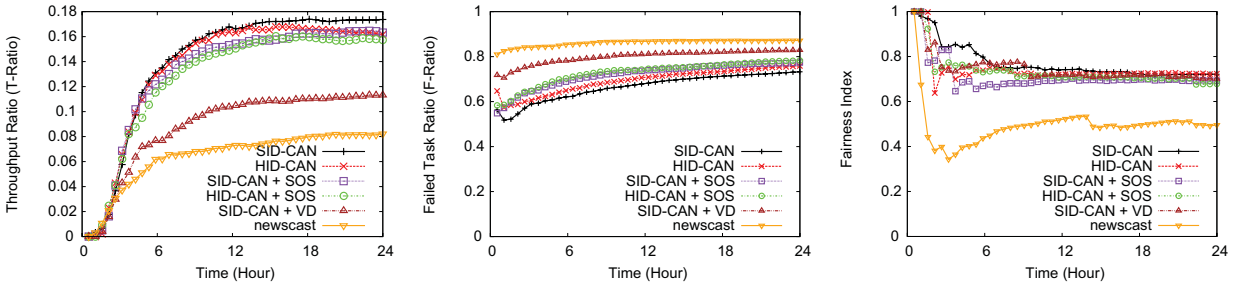
Through these three figures, we observe that all the performance metrics are improved as we decrease the demand ratio (λ). This is reasonable because smaller demand ratio (i.e. smaller resource amount demanded per task) will definitely induce easier resource matching. An interesting observation is that the Newscast protocol performs even much better than SID-CAN when the demand ratio is small. For instance, when λ=0.25 (i.e. all the tasks demand small amount of resources), the Newscast protocol performs well on throughput ratio (up to 0.74), while the result of HID-CAN is pretty close to that of Newscast on this metric. Whereas, it is pity that the Newscast protocol suffers distinctly more failed tasks and poorer fairness index than our designed HID-CAN or SID-CAN protocol under various demand ratios. It is worth notice that our HID-CAN suffers only 2 failed tasks out of the totally 14362 submitted tasks when the demand ratio is relatively small (such as λ=0.25) in the whole one-day test, compared to 1793 failed tasks using Newscast protocol (see Fig. 7 (b)).

Another interesting result is that SoS does take positive effect in some cases. For instance, SID-CAN + SoS performs a little worse than without SoS support in Fig. 5 (a), while it performs much better in the large demand ratio situation (Fig. 7 (a)). Although SID-CAN + SoS could perform stably in different situations, such a solution suffers twice resource query overhead than those without SoS.

Overall, we conclude that HID-CAN is a stable protocol, which always performs efficiently in any situation on almost all metrics, such as throughput ratio, failed task ratio, and fairness index. Consequently, HID-CAN should be considered the best choice for the SOC platform.

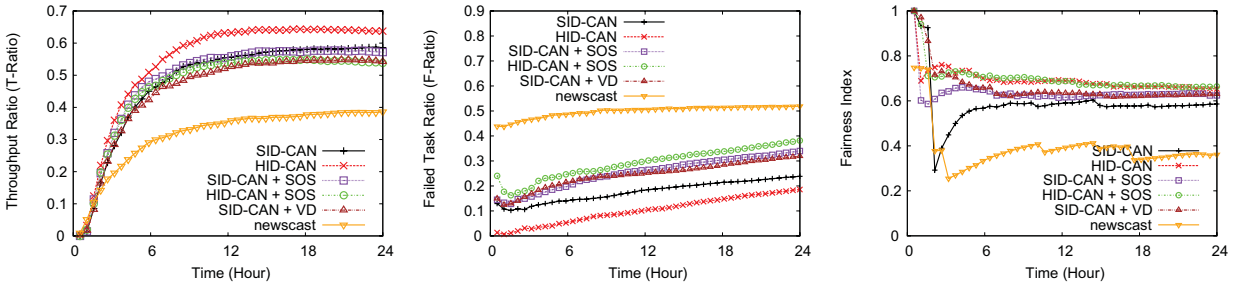
We also evaluate the scalability of our recommended algorithm, HID-CAN, during one-day test as shown in Table III. We could clearly observe that the four primary performance metrics do not notably change with the increasing system scale. We define message delivery cost as the summed number of various messages (including state-update message, duty-query message, index-jump message, index-agent message, etc.) sent/forwarded per node. Table III shows that the message delivery cost increases very slowly, probably under logarithmic speed.

Finally, we evaluate the HID-CAN under different levels of dynamic environment with a certain ratio of churning nodes, as shown in Fig. 8 (λ=0.5). Since index-diffusion delay or the departure maintenance cost on each node is



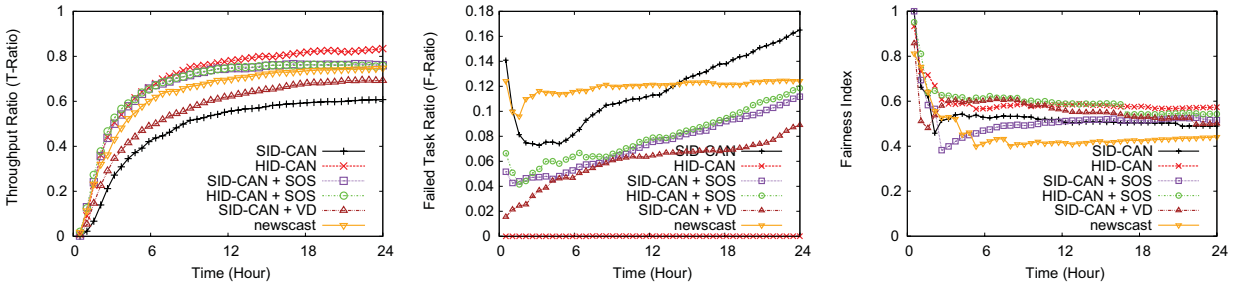
(a) throughput ratio (b) failed task ratio (c) fairness index

Figure 5. The efficacy of resource discovery protocols (λ=1)



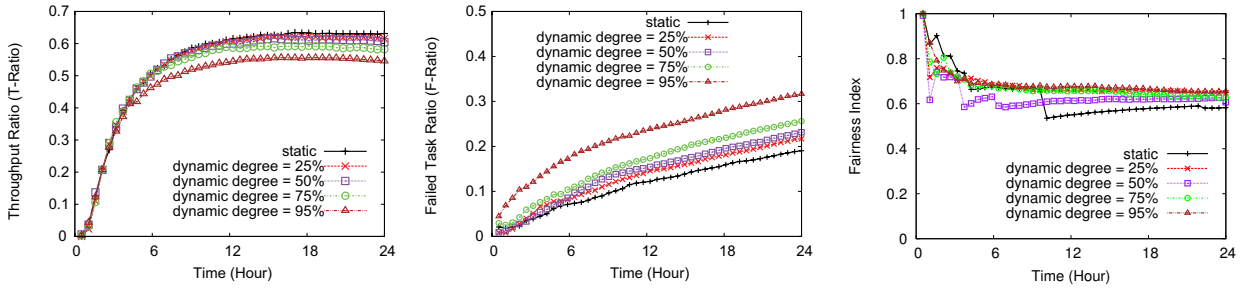
(a) throughput ratio (b) failed task ratio (c) fairness index

Figure 6. The efficacy of resource discovery protocols (λ=0.5)



(a) throughput ratio (b) failed task ratio (c) fairness index

Figure 7. The efficacy of resource discovery protocols (λ=0.25)



(a) throughput ratio (b) failed task ratio (c) fairness index

Figure 8. HID-CAN under Different Node Churning Rates

Table III
SYSTEM SCALABILITY OF HID-CAN

metric \ scale	2000	4000	6000	8000	10000	12000
throughput ratio	0.637	0.618	0.612	0.606	0.592	0.597
failed task ratio	18.6%	19.8%	19.7%	20.1%	21.4%	20.7%
fairness index	0.653	0.623	0.638	0.644	0.651	0.641
msg delivery cost	3403	4311	5019	5728	6078	6427

only about several network delays each of which takes about only 200 milliseconds on the WAN, these costs are usually tolerable compared to application data transmission time. Hence, for the dynamic situation, we mainly focus on the question: whether or not the frequently changing CAN structures would impact the resource discovery effect. We use *dynamic degree* to denote the ratio of the churning nodes and the total number of nodes within one task's lifetime on average (i.e. 3000 seconds). The node-churning events are uniformly distributed to every moment in each whole experimental duration. For example, dynamic degree = 0.25 means that there are about 25% nodes arbitrarily disconnected from the network every 3000 seconds and also there are the same number of new nodes joining meanwhile. We implement the node departure maintenance on each departure node's neighbors to refresh their neighborhoods and a *binary partition tree* based *background zone reassignment* algorithm [14] to ensure each node always corresponds to a globally unique zone.

From Fig. 8, we observe that the resource allocation result is degraded a little bit with increasing degree of dynamic environment. When the churning node ratio is up to 50%, the throughput ratio and failed task ratio will not be remarkably influenced compared to the static environment without churning-nodes. This validates that our HID-CAN protocol performs quite satisfactorily in dynamic situation.

V. RELATED WORK

During past few years, there already exist a lot of range-based query methods over DHT [15], [16], [17], [18], [19], [21], [22]. They have two short-comings compared to our solution. They always rely on some additional order-preserving (or locality-preserving) hash function to reorganize the DHT nodes, significantly complicating the system implementation. For example, Mercury [15] maps d attribute-hubs to DHT (such as Chord [13]), and each range query is split to multiple sub-queries based on different attributes and conducted in the multiple hubs respectively. Armada [16] maps all the objects to DHT nodes through a conceptual partition tree, while Murk [19] indexes multi-dimensional data partition using kd-tree. Other tree structures (such as skiptree [17] and trie [21]) were also leveraged to improve the range query over DHT. In comparison, our solution never borrows additional hash functions but still achieves expected query effect by simply proactively diffusing index-nodes over the INSCAN overlay. To our knowledge, there are some researches [29], [22] which also adopt the structure similar to INSCAN. C^2 [29], for example, combines CAN and Chord, making the messages be routed exactly according to the Chord rule along each dimension. Whereas, without carefully designed proactive index-diffusion strategy, simple combination of Chord and CAN cannot deliver satisfactory resource matching rate for range query demand.

On the other hand, all the existing solutions mainly aim to get as complete range-query result as possible with limited query delay. There are usually two phases for each range query: locating the boundary (or centric) responsible nodes within the specified range and then checking all of them and their neighbors one by one until finding all the data. Apparently, this may easily incur unbounded query delay or intolerably heavy network traffic. Armada [16] proposes a delay-bounded range-query method and the INSCAN based Range Query could also be proven as a query message delay bounded solution. However, such a short-response feature is achieved at the cost of heavy network traffic because of the flooded query messages from the partition tree's root node or boundary-line duty nodes to all of its range-overlapped leaf nodes. RT-CAN [22] partitions the query range to several concentric circles and checks the responsible nodes from inside out, and this method is proven well-adaptive to the load imbalanced situation. Through experiments on Amazon's EC2, however, RT-CAN's query throughput/performance also shows notable degradation with even slightly expanding query range, in that larger range causes more objects to be retrieved and more nodes to be involved in the query processing. In addition, note that none of the existing works take the mutual resource contention issue into account for maximizing queries' actually gained resource shares. CAN-based protocol in [27] makes use of an additional virtual dimension to disperse the potential competition, but such a method performs unstably due to its inevitabe over-dispersed qualified resource records.

In comparison, without traversing all responsible nodes within the query range, high resource matching rate could also be achieved by our elaborative random diffusing non-empty-cache nodes' identifiers (i.e. index nodes) in a proactive manner. In addition, by randomly selecting index nodes in the query phase, our solution could effectively mitigate the mutual contention among requesters, maximizing each requester's real allocated multi-dimensional shares along every resource dimension. In particular, the HID-CAN protocol (a specific version of PID-CAN) has been proven very effective for keeping the stable resource discovery effect with low message delivery cost under various demand ratios.

VI. CONCLUSION AND FUTURE WORK

This is the first work to study the resource discovery protocol especially suitable for multi-dimensional virtualized resource allocation on Self-Organizing Cloud (SOC). Each resource discovery job should be a multi-dimensional range query with a minimal demand due to the sharable resources in SOC. By randomly propagating nodes' identifiers (or

indexes) from index-node to index-node over CAN, our design (PID-CAN) can effectively increase the success rates in searching qualified resources, especially in accordance with the characteristics of proportional-share model (PSM). Compared to spreading index diffusion (SID) method, the hopping index diffusion (HID) method shows much better and more stable performance without the necessity of extra competition-aware assistance (such as Slack-on-Submission or additional virtual dimension). We also validate that HID-CAN could perform stably in dynamic node-churning environment. For the future work, we plan to study the PSM based execution fault-tolerance issues using check-pointing technologies on top of the HID-CAN protocol.

ACKNOWLEDGMENTS

This research is supported by a Hong Kong RGC grant HKU 7179/09E and a HKU Basic Research grant (Grant No. 10401460), and also in part by a Hong Kong UGC Special Equipment Grant (SEG HKU09).

REFERENCES

[1] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 50–55, 2009.

[2] Amazon elastic compute cloud: http://aws.amazon.com/ec2/.

[3] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat, "Enforcing performance isolation across virtual machines in xen," in *Middleware*, 2006, pp. 342–362.

[4] L. Cherkasova, D. Gupta, and A. Vahdat, "Comparison of the three cpu schedulers in xen," *SIGMETRICS Perform. Eval. Rev.*, vol. 35, no. 2, pp. 42–51, 2007.

[5] J. P. Walters, V. Chaudhary, M. Cha, S. G. Jr., and S. Gallo, "A comparison of virtualization technologies for hpc," *In AINA'08: 25th International Conference on Advanced Information Networking and Applications*, pp. 861–868, 2008.

[6] Cloud desktop: http://www.gladinet.com/.

[7] icloud project: http://www.icloud.com/en.

[8] M. Feldman, K. Lai, and L. Zhang, "The proportional-share allocation market for computational resources," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, pp. 1075–1088, 2009.

[9] L. E. Grit and J. S. Chase, "Weighted fair sharing for dynamic virtual clusters," *SIGMETRICS Perform. Eval. Rev.*, vol. 36, pp. 461–462, June 2008.

[10] B. Raghavan, K. Vishwanath, S. Ramabhadran, K. Yocum, and A. C. Snoeren, "Cloud control with distributed rate limiting," *SIGCOMM Comput. Commun. Rev.*, vol. 37, pp. 337–348, August 2007.

[11] S. K. Barker and P. Shenoy, "Empirical evaluation of latency-sensitive application performance in the cloud," in *Proceedings of the first annual ACM SIGMM conference on Multimedia systems*, ser. MMSys '10. New York, NY, USA: ACM, 2010, pp. 35–46.

[12] S. Di and C.-L. Wang, "Conflict-minimizing dynamic load balancing for p2p desktop grid," in *Grid'10: The 11th IEEE/ACM International Conference on Grid Computing*, 2010, pp. 137–144.

[13] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *SIGCOMM '01: Proceedings of the 2001 conference on App., tech., arch., and prot. for comp. comm.*, vol. 31, no. 4. New York, NY, USA: ACM, October 2001, pp. 149–160.

[14] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *SIGCOMM '01: Proceedings of the 2001 conference on App., tech., arch., and prot. for comp. comm.* New York, NY, USA: ACM, 2001, pp. 161–172.

[15] A. R. Bharambe, M. Agrawal, and S. Seshan, "Mercury: supporting scalable multi-attribute range queries," in *SIGCOMM '04: Proceedings of the 2004 conference on App., tech., arch., and prot. for comp. comm.* New York, NY, USA: ACM, 2004, pp. 353–366.

[16] D. Li, J. Cao, X. Lu, and K. C. C. Chen, "Efficient range query processing in peer-to-peer systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 1, pp. 78–91, January 2009.

[17] A. Gonzalezbeltran, P. Milligan, and P. Sage, "Range queries over skip tree graphs," *Computer Communications*, vol. 31, no. 2, pp. 358–374, February 2008.

[18] S. Wang, Q. H. Vu, B. C. Ooi, A. K. Tung, and L. Xu, "Skyframe: a framework for skyline query processing in peer-to-peer systems," *The VLDB Journal*, vol. 18, pp. 345–362, January 2009.

[19] P. Ganesan, B. Yang, and H. Garcia-molina, "One torus to rule them all: Multi-dimensional queries in p2p systems," in *In WebDB'04: Proceedings of the 7th International Workshop on the Web and Databases*. ACM Press, 2004.

[20] H. Shen and C.-Z. Xu, "Performance analysis of dht algorithms for range-query and multi-attribute resource discovery in grids," in *ICPP'09: 38th International Conference on Parallel Processing*, 2009, pp. 246–253.

[21] A. Datta, M. Hauswirth, R. John, R. Schmidt, and K. Aberer, "Range queries in trie-structured overlays," *IEEE International Conference on Peer-to-Peer Computing*, pp. 57–66, 2005.

[22] J. Wang, S. Wu, H. Gao, J. Li, and B. C. Ooi, "Indexing multi-dimensional data in a cloud system," in *SIGMOD Conference*, 2010, pp. 591–602.

[23] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregation in large dynamic networks," *ACM Transactions on Computer Systems*, vol. 23, no. 3, pp. 219–252, 2005.

[24] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*. New York, NY, USA: ACM, 2003, pp. 164–177.

[25] Peersim simulator: http://peersim.sourceforge.net.

[26] W. K. Mark Jelasity and M. van Steen, "Newscast computing," Vrije Universiteit Amsterdam, Tech. Rep., 2006.

[27] J. S. Kim and et al., "Using content-addressable networks for load balancing in desktop grids," in *HPDC'07: 16th International Symposium on High Performance Distributed Computing*, New York, USA, 2007, pp. 189–198.

[28] R. K. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modelling*. John Wiley & Sons, April 1991.

[29] W. Cai, S. Zhou, W. Qian, L. Xu, K. Tan, and A. Zhou, "C2: a new overlay network based on can and chord," *Int. J. High Perform. Comput. Netw.*, vol. 3, no. 4, pp. 248–261, 2005.