Available online at www.sciencedirect.com
ScienceDirect
Computer Networks 51 (2007) 1072–1094
Computer Networks
www.elsevier.com/locate/comnet

CISS: An efficient object clustering framework for DHT-based peer-to-peer applications ☆

Jinwon Lee *, Hyonik Lee, Seungwoo Kang, Su Myeon Kim, Junehwa Song

Korea Advanced Institute of Science and Technology (KAIST), Division of Computer Science, Department of EECS, Network Computing Laboratory, 373-1 Guseong-dong Yuseong-gu, Daejeon, Republic of Korea

Received 18 May 2005; received in revised form 27 June 2006; accepted 18 July 2006
Available online 14 August 2006

Responsible Editor: R. Boutaba

Abstract

In most DHT-based peer-to-peer systems, objects are totally declustered since such systems use a hash function to distribute objects evenly. However, such an object de-clustering can result in significant inefficiencies in *advanced access operations* such as multi-dimensional range queries, continuous updates, etc, which are common in many emerging peer-to-peer applications. In this paper, we propose CISS (Cooperative Information Sharing System), a framework that supports efficient object clustering for DHT-based peer-to-peer applications. CISS uses a Locality Preserving Function (LPF) instead of a hash function, thereby achieving a high level of clustering without requiring any changes to existing DHT implementations. To maximize the benefit of object clustering, CISS provides efficient routing protocols for multi-dimensional range queries and continuous updates. Furthermore, our cluster-preserving load balancing schemes distribute loads without hot-spots while preserving the object clustering property. We demonstrate the performance benefits of CISS through extensive simulation.

Keywords: Distributed hash table (DHT); Object clustering; Peer-to-peer application; Multi-dimensional range query; Load balancing

1. Introduction

Distributed Hash Tables (DHTs) [31,33,36,38] have been receiving a lot of attention as a scalable and efficient infrastructure for Internet-scale data management [24,27]. DHT organizes highly distributed and loosely coupled peer nodes into a peer-to-peer network. Such a P2P network makes it possible for users to store and query over a massive number of objects. DHT has already been adopted in many P2P systems including wide-area file systems [10,26]

☆ A preliminary version was presented in International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P'04), collocated with VLDB'04, Toronto, Canada, August 2004.

* Corresponding author. Tel.: +82 42 869 3586; fax: +82 42 869 3510.

E-mail addresses: jcircle@nclab.kaist.ac.kr (J. Lee), hilee@nclab.kaist.ac.kr (H. Lee), swkang@nclab.kaist.ac.kr (S. Kang), smkim@nclab.kaist.ac.kr (S.M. Kim), junesong@nclab.kaist.ac.kr (J. Song).

1389-1286/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2006.07.005

and Internet-scale query processors [15,17]. These DHT-based systems efficiently support basic access operations such as `Put (key, object)` and `Get (key)`, and thus simple P2P applications can be developed easily with such basic operations.

However, many emerging P2P applications require *advanced access operations* such as *multi-dimensional range queries, continuous updates, similarity searches, aggregation*, etc, which access a set of semantically related objects in parallel or in sequence, rather than performing separate accesses for individual objects. As an example, consider massively multiplayer online games (MMOGs). Each player continuously queries the game status of nearby players in a virtual world, which can be treated as multi-dimensional range queries. These range queries access a set of closely located players in parallel. Meanwhile, streams of updates such as players' locations and status [5,19] are intensively generated. Due to players' continuous movements in the virtual world, a sequence of successive updates is semantically close [23]. Similar situations occur in P2P auctions. They also intensively generate a high number of range queries and updates which access highly related objects over multiple attributes, e.g., interest area queries [29].

As opposed to basic operations, advanced access operations cannot be efficiently supported by existing DHT-based systems. We observe that this is mainly due to the de-clustering nature of the DHTs. In DHT-based systems, objects are totally declustered since such systems use a hash function to distribute objects evenly across different peer nodes. Even highly co-related objects are spread over different peer nodes, which makes it difficult to access related objects in parallel or in sequence. Consider a multi-dimensional range query as an example. With DHTs, resolving the query requires a number of lookup operations. Although it searches for semantically related objects, each key value in the query range should be enumerated and individually searched for via a separate DHT lookup. A similar situation occurs with continuous updates. DHT lookups have to be performed for every object update to locate the corresponding peer node.

In this paper, we propose CISS (Cooperative Information Sharing System), a novel framework that supports efficient object clustering for DHT-based P2P applications. As the de-clustering nature of the DHT is the source of its limitation, CISS provides the clustering property to DHT to match the need of emerging P2P applications and significantly improves the efficiency of advanced access operations. For example, in multi-dimensional range queries, a group of semantically related objects can be accessed via a single DHT lookup. Hence, the number of DHT lookups can be greatly reduced. In addition, only a small number of nodes are involved in query processing. CISS also can take advantage of semantic closeness in a sequence of object updates. Since semantically related objects are clustered, continuous updates can be routed to the same peer node without performing additional DHT lookups. Thus, the number of lookups and the latency of update routing can be considerably reduced. Such a performance improvement is often critical for many online P2P applications, such as those mentioned above. Harnessing CISS, the P2P applications will provide a faster response to users than using the original DHTs alone.

To provide an efficient framework for object clustering over DHTs, CISS addresses several technical challenges. *First, CISS provides a Locality Preserving Function (LPF) as its object distribution function*. Using the LPF instead of a hashing function, it achieves a high degree of object clustering without requiring any changes to existing DHT implementations. Furthermore, it performs *multi-dimensional clustering* as objects are generally composed of multiple attributes and also accessed by using multi-dimensional keys. For the LPF, we provide a key encoding scheme which constructs a key from multiple attributes of an object while preserving locality. The encoding scheme considers different data types of different attributes and further applies the Hilbert Space Filling Curve (SFC).

Second, to maximize its benefit, CISS provides efficient routing protocols. In this paper, we concentrate on two important access operations, i.e., multi-dimensional range queries and continuous updates, as discussed in the examples above. To route multi-dimensional range queries efficiently, we propose a *forwarding-based query routing protocol*. By forwarding a query to succeeding peer nodes, the protocol prevents the possibility of query congestion in a peer node while reducing the number of costly DHT lookups. To route continuous updates efficiently, we also propose a *caching-based update routing protocol*. The routing protocol does not perform additional lookups if streams of updates belong to the key range of the most-recently-searched peer node, significantly reducing update routing overhead. We consider these operations as representative of advanced operations in the

sense that the former accesses a group of objects in parallel and the latter in sequence. We expect many other operations can be constructed as combinations and/or variations of the two.

Third, CISS addresses the issue of load imbalance which naturally results from clustering and provides a cluster preserving load balancing policy. While the idea of clustering forms the basis for supporting advanced operations, it conflicts with an important concept of the original DHTs, i.e., achieving scalability by de-clustering objects across peer nodes. Thus, with skewed distribution of queries and objects, the system may easily result in serious load imbalance and hence hotspots. To prevent hotspots, load balancing must be performed. More importantly, the object clustering property must be preserved even after load balancing. CISS addresses this challenge with two load balancing schemes, i.e., local- and global-handover.

The rest of the paper is organized as follows. Section 2 reviews related work in the area of DHT-based P2P systems. In Section 3, we describe the architecture of CISS. In Section 4, we explain technical issues faced in realizing CISS, including LPF, query and update routing protocols, and cluster-preserving load balancing schemes. Section 5 presents results from simulation studies of CISS. Finally, Section 6 concludes our work.

2. Related work

In this section, we compare CISS with other DHT-based P2P systems. Other than DHT-based P2P systems, we could consider unstructured P2P systems such as Gnutella [44] and Freenet [45]. They basically use a flooding-based approach for object lookups. Thus, they incur heavy network and system overhead [31,36]. Moreover, it is quite difficult for these systems to provide guaranteed lookup performance in any senses [4]. Consequently, we do not consider an unstructured P2P system as a suitable base system for P2P applications which require efficient support for advanced access operations.

To the best of our knowledge, CISS is the first attempt to provide an efficient clustering framework for advanced access operations over DHT. Some studies on range queries [1,6,14,20,25,32,34] exist and can be considered as a limited attempt for one-dimensional clustering. CISS is different from those research works in that it supports multi-dimensional clustering and addresses related issues more thoroughly. Thus, CISS is more efficient in providing advanced access operations, e.g., multi-dimensional range queries and continuous updates. In addition, CISS addresses the load imbalance problem that arises from object clustering. As mentioned before, the concept of object clustering conflicts with the main idea of the original DHTs, and may easily result in serious load imbalance. We think that an effective clustering framework should importantly consider the issues related to this conflict between clustering and the DHT. While there are some studies [7,10,30,36] related to load balancing in DHT-based P2P systems, their contexts are only on basic lookup operations under the original DHT that uses consistent hashing.

In [1,34], the authors extend CAN [31] for range queries assuming one-dimensional clustering by using query flooding techniques. In [14,20], the authors proposed a newly designed range addressable P2P network instead of utilizing existing DHT implementations. CLASH [25] and PHT [32] apply an extensible hashing technique over DHT. They efficiently achieve adaptive object clustering as well as support range queries. However, basic access operations require multiple DHT lookups, $O(\log(D))$ times where D is the maximum key depth. All these research works handled range queries and clustering in one-dimensional space.

Several research works [6,12,35] tried to provide multi-dimensional range queries over DHT. Mercury [6] supports multi-dimensional range queries based on one-dimensional object clustering. It constructs a separate DHT for each attribute. In order to reduce the number of DHT lookups, a query is sent only to the DHT of the attribute with the lowest query selectivity. However, since it still based on one-dimensional clustering, many objects which are irrelevant to the given multi-dimensional range query may be accessed, resulting in degradation of the overall performance [28]. Furthermore, Mercury can be very inefficient when updates are frequent since updates need to be sent to all DHTs for correct query processing.

Squid [35] supports multi-dimensional range queries over DHT by using the Hilbert Space Filling Curve (SFC). It reduces the number of DHT lookups for query processing by recursively refining queries. However, it may easily incur severe query congestion over some peer nodes matching high order bits of the queries and thus limit the overall scalability of the whole system. To balance loads among different peer nodes, they apply the virtual

sever-based scheme [7,10,30,36] which was developed for original DHTs. However, such direct application of the virtual-server based approach may much disturb object clustering. This is mainly because physical peer nodes can manage non-contiguous key ranges, i.e., multiple virtual servers.

Ganesan et al. [12] proposed multi-dimensional range query routing schemes based on a Space Filling Curve (SFC) and a k-d tree. The SFC-based scheme is similar to our prior work [22]. However, it only assumes a skip graph [3,16] as its underlying DHT without considering other DHT structures. The k-d tree-based scheme yields better object clustering than the SFC-based one in high dimensional spaces. However, it is only applicable to CAN [31]. In addition, it is difficult to achieve load balancing in such a tree-based approach as the authors point out.

Recently, Ganesan et al. [13] proposed an online scheme to balance storage space taken by peer nodes in the context of P2P databases. That is, the scheme tries to evenly distribute data tuples to peer nodes, thereby guaranteeing the storage imbalance ratio to be under a constant value, e.g., 4.24. However, it cannot be adopted to balance the overall loads of a system as it only handles the imbalance of storage space. Note that the overloaded nodes may fail or provide poor service, limiting the overall scalability of the whole system. In CISS, overloaded nodes are quickly relieved by dynamically detecting load state and performing cluster-preserving load balancing. In addition, we show experimentally that the proposed load balancing scheme hardly affects the lookup performance of the underlying DHT.

3. System architecture

CISS is a three-tier system as shown in Fig. 1. Such an architecture is similar to existing DHT-based P2P systems [10,17,26]. The CISS mediates between a P2P application and a DHT, supporting an efficient object clustering.

To use CISS, a P2P application developer first has to describe the object model, of the application. The object model, such as key attributes, attribute names, data types, and auxiliary meta-data information, is described in a *schema*. The schema is used for the key encoding (see Section 4.1 for several schema examples). Then, a P2P application can issue queries as well as updates to CISS by using a simple conjunctive normal form interface (see Table 1).

CISS consists of client and server modules. The client module of CISS takes the updates or queries

Table 1
Interfaces for DHT and CISS

DHT	CISS
Lookup(key) $\rightarrow$ IP address	Update: $(A1 = \text{value}) \wedge (A2 = \text{value}) \wedge \cdots$
Join (node identifier)	Query: $\text{Predicate}_{A1} \wedge \text{Predicate}_{A2} \wedge \cdots$
Leave( )	Predicate = Attribute **Operator** Value Operator = {>, <, $\geqslant$, $\leqslant$, =}

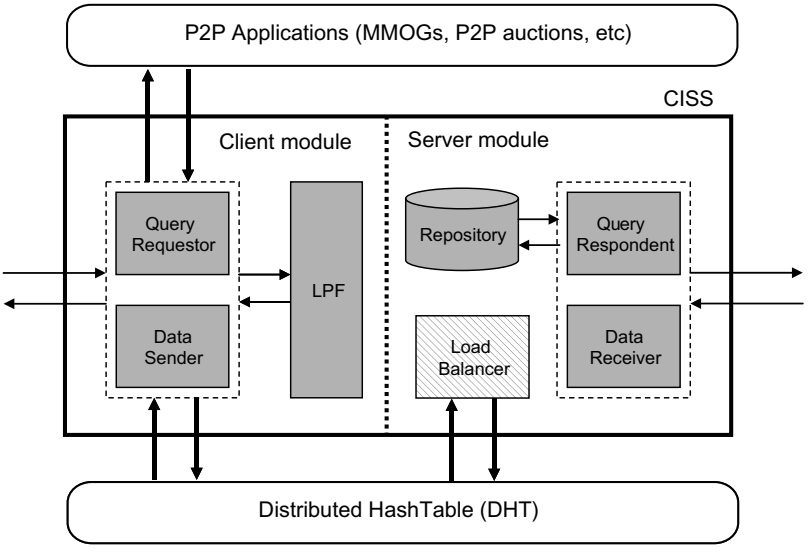


Fig. 1. CISS architecture.

which are generated from P2P applications. It then routes them to rendezvous peer nodes for processing. The client module utilizes an LPF to encode multiple attributes of an object to a key. This key is used to perform a DHT lookup to search for a rendezvous peer node in the P2P network. The server module of CISS stores objects to its repository and processes incoming queries. It returns matched results to requesting peer nodes. The load balancer in the server module takes charge of cluster-preserving load balancing. CISS use a *leave* followed by a *join* style load balancing scheme which is supported by most underlying DHTs. Thus, load balancer interacts with DHT for *leaves* and *joins* as well as DHT lookups.

While the DHT is a scalable and efficient solution for many Internet-scale applications, it is seriously limited in supporting advanced access operations. Our observation is that clustering is an important property to overcome such limitation of the DHT. However, as mentioned, clustering conflicts with the original idea of DHT. Hence, a naïve attempt to provide clustering on DHTs deprives a system of the advantage of the DHTs, e.g., resilience to skewed distribution of objects and queries, and easily results in hotspots and limited scalability. CISS in cooperation with its underlying DHT layer gives scalability and efficiency upon the advanced operations. CISS framework first provides a LPF for object clustering, efficient protocols for query and update routing, and further load balancing schemes which can efficiently support above mentioned operations. Technical details of CISS are described in Section 4.

CISS utilizes DHT as a basic lookup layer. DHT [31,33,36,38] organizes highly distributed and loosely coupled peer nodes into a P2P network for storing and querying a massive number of objects. In a DHT environment, not only the placement of objects on nodes, but also the join and leave of nodes in the P2P network can be done efficiently without any global knowledge. As an underlying lookup layer, CISS uses a one-dimensional DHT such as Chord [36], Pastry [33] and Tapestry [38]. It is mainly for performance reason[1]; while there are DHTs supporting multi-dimensional lookups such as CAN [31], the one-dimensional DHTs show much better performance for basic lookup operations. Fig. 2 shows Chord as an example of the DHT. In the figure, five peer nodes cooperatively manage a 4-bit key space. Each node has a unique node identifier and is responsible for the key range between itself and its predecessor.

From the standpoint of object lookup in the P2P network, DHT has two attractive characteristics. First, DHT enables content-based search over a large number of distributed objects. An object is encoded as an N-bit key based on its contents, e.g., attribute values. Upon an update or a query for the same object, the same N-bit key is generated. Hence, the update and query are routed to the same rendezvous peer node. As such, the content-based search is efficiently achieved through this rendezvous point approach with a small number of messages. Second, DHT enables efficient object lookup. Given an object, lookups proceed in a multi-hop fashion; each node maintains information (IP addresses) about a small number of other nodes (neighbors) and forwards the lookup message recursively to the node whose ID is closest to the key of the given object. DHT theoretically ensures that the rendezvous peer node for the object is located in $O(\log S)$ hops, where S is the number of peer nodes in the P2P network.

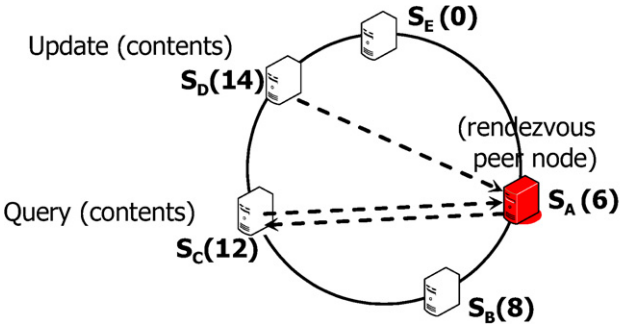
4. Technical issues

4.1. Locality preserving function (LPF)

To support object clustering in CISS, we newly develop a LPF which generates the key of an object used for lookup and routing with an underlying DHT. As previously mentioned, a key is generated considering the application schema specified by the application developer. The schema contains numerical parameters used for key encoding, e.g., D (the number of key attributes), *minimum* and *maximum* values of Numerical attributes and d (hierarchy depth) of String attributes.

Let us consider two sample schemas shown in Figs. 3(b) and 4(b). Fig. 3(b) shows the schema of a MMOG. From the MMOG, the location of an avatar is determined by (x, y) coordinates in 2-D space. The minimum and maximum values of x and y are set to 0 and 100, respectively, just as an example for the ease of explanation. In practice,

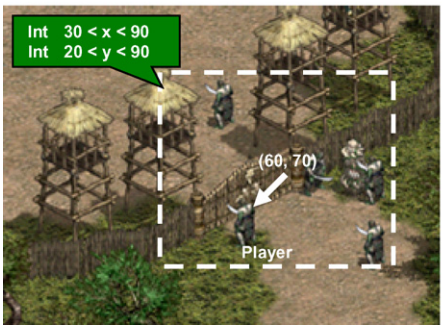
[1] In general, the lookup performance of DHTs which support one-dimensional lookup is $O(\log S)$ where S is the number of nodes. However, that of CAN is $O(dS^{1/d})$ when d-dimension is used. Thus, the lookup performance of CAN is much worse unless d is very high, e.g., 20 when S is 10^6. This is demonstrated through extensive simulations in [12].



Peer node	Key range
S_A	(0, 6]
S_B	(6, 8]
S_C	(8, 12]
S_D	(12, 14]
S_E	(14, 0]

Fig. 2. DHT example (Chord).

a

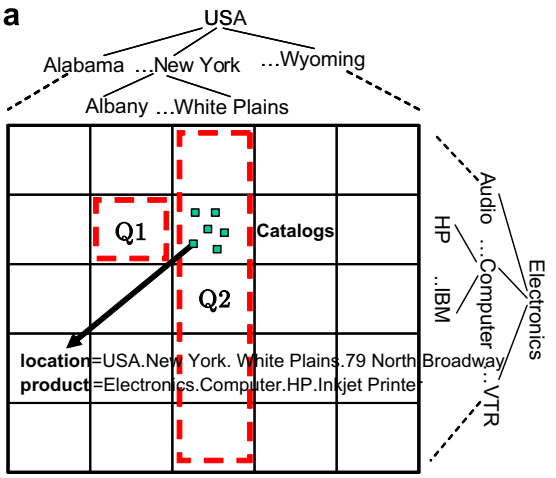


b

Name	*Type*		*Key*
X location	`Int`	minimum=0 maximum=100	**Yes**
Y location	`Int`	minimum=0 maximum=100	**Yes**
Player ID	`String`	hierarchy depth=1	No
Score	`Int`	minimum=0 maximum=10	No

Fig. 3. MMOGs. (a) Queries in a MMOG and (b) schema for a MMOG.

a



b

Name	*Type*		*Key*
Location	`String`	hierarchy depth=4	**Yes**
Product	`String`	hierarchy depth=4	**Yes**
Phone number	`String`	hierarchy depth=4	No
Date	`DATE`	minimum=00/01/01 maximum=99/12/31	No

Fig. 4. P2P auctions. (a) Queries in a P2P auction and (b) schema for a P2P auction.

the range is highly variable depending on games and will be much larger. Fig. 4(b) describes the schema of a P2P auction. From the P2P auction, it is very common to search for item sets with range queries using location and product category [29]. Based on this scenario, D is set to 2. The hierarchical depth, d values of Product and Location are set to 4, just as a simple example. The d values of Product and Location also vary depending on the item catalog and the address system of a country.

Note that P2P application developers must consider the following guideline in selecting *key attributes*. This is important because only key attributes are used for key encoding among all attributes of an object. If an attribute is not selected for key encoding, queries on this attribute must be sent to all peer nodes. On the other hand, if all attributes are selected, the benefit of object clustering decreases; for example, a query for a single attribute is minutely divided and routed to multiple nodes. Thus, the selection of key attributes should consider dominantly issued query types for the greatest clustering benefit. For example, the developers of MMOGs may choose two attributes (X and Y locations) because two-dimensional range queries are dominantly issued (see Fig. 3(a)). Similarly, Location and Product can be selected for P2P auctions (see Fig. 4(a)).

The LPF constructs the key of an object through two steps. First, it encodes each attribute value to a subordinate key. It then maps the generated multiple subordinate keys to a key by using the Hilbert SFC. Both steps preserve the locality of objects. If D attributes are determined as key attributes, each key attribute is encoded to N/D ($=M$) bits, where N is the number of bits used in the underlying DHT implementations, e.g., 160 in Chord and 128 in Pastry. (We can give different weights on the number of bits allocated for different key attributes in the encoding, if the relative densities of the ranges of different key attributes are known apriori.)

Step 1: $\{(A1 = \text{value}) \wedge (A2 = \text{value}) \wedge \cdots\} \rightarrow \{\text{bits}_{A1} \wedge \text{bits}_{A2} \wedge \cdots\}$ **(Subordinate keys)**
Step 2: $\{\text{bits}_{A1} \wedge \text{bits}_{A2} \wedge \cdots\} \rightarrow$ ***N*-bit key of object**($A1$: Attribute 1, $A2$: Attribute 2, ...)

Step 1: Subordinate key encoding – the LPF classifies data types of attributes into Numeric and String types, and applies different encoding schemes for each. We explain each encoding scheme with examples.

The encoding scheme for the `Numeric` type handles `int`, `long`, `float`, `double` and `DATE` datatypes. In our example, the MMOG uses attributes of Numeric type, i.e., X and Y locations. For subordinate key encoding of Numeric type, each attribute value is simply rescaled by multiplying a coefficient, $(2^M - \text{minimum})/(\text{maximum} - \text{minimum})$, where minimum and maximum are defined in the schema. For example, the two attribute and value pairs, i.e., $\{x = 60 \wedge y = 70\}$ are encoded to $\{x = 1010 \wedge y = 1011\}$ if $M = 4$. In the same way, the two-dimensional range query $\{(30 < x < 60) \wedge (20 < y < 90)\}$ in Fig. 3(a) is encoded to $\{(0101 < x < 1110) \wedge (0011 < y < 1110)\}$. Attribute clustering is well preserved as the numeric type is a total ordered set.

For the `String` type, we propose a *hash-concatenation encoding scheme*. The P2P auction in Fig. 4(b) uses two attributes of string type, Location and Product, as their key attributes. The values of the attributes are represented using a *hierarchical naming structure* as follows.

A1: Location = USA.New York.White Plains. 79 North Broadway
("USA" is the value of the topmost level in the hierarchy, "New York" is the second highest and so on)
A2: Product = Electronics.Computer.HP.Inkjet Printer
("Electronics" is the value of the topmost level in the hierarchy, "Computer" is the second highest and so on)

In P2P auctions, range queries like Q1 and Q2 in Fig. 4(a) are dominantly issued.[2] As in Q1 and Q2, queries with wild cards in one or more levels in the naming hierarchy are common. We consider that queries with partial string matching in a middle level of the hierarchy, e.g., USA.N*.White Plains, are unusual. Hence, clustering according to the hierarchy is sufficient, and that between similar string values in the same level is not necessary.

The hash-concatenation scheme hashes the value of each level in the hierarchy into M/d bits, where d is the hierarchy depth, which is defined in the schema. It then concatenates the hashed values one after another. To hash a variable-length string value of each level to a fixed-length bit representation, a modified SHA-1 [36] hash function is used.[3]

[2] Although String type keyword queries such as "%keyword%" are popular in P2P file sharing, we do not tackle such queries because they do not benefit from object clustering.

[3] SHA-1 [36] hashes a string to the randomized 160-bits. Each level's string has to be hashed to M/d bits which are less than 160. Thus, just the M/d prefix bits of SHA-1 are used as a hashed value.

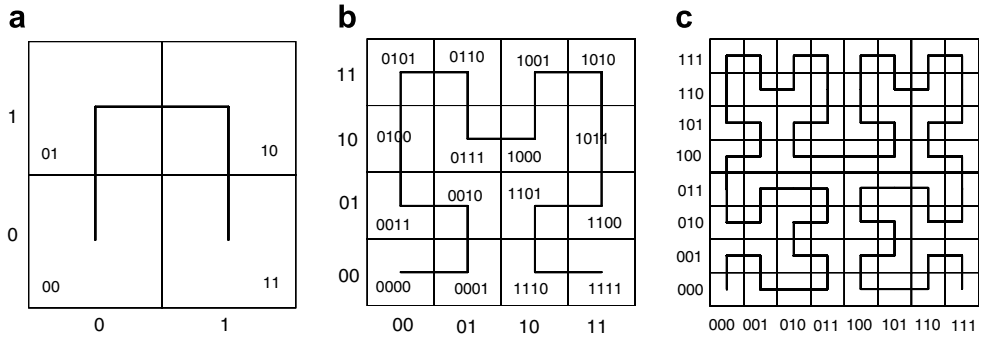


Fig. 5. Hilbert SFC. (a) First order, (b) second order and (c) third order.

Queries are also encoded in the same way. The encoding examples are shown below when M is 80 and d is 4. Each string in the hierarchy is hashed to 20 bits.

A1: Location = USA.New York.White Plains. 79 North Broadway → h_{20}(USA)·h_{20}(New York)·h_{20}(White Plains)·h_{20}(79 North Broadway)
A2: Product = Electronics.Computer.HP.Inkjet Printer → h_{20}(Electronics)·h_{20}(Computer) · h_{20}(HP) · h_{20}(Inkjet Printer)

Q1: Location = h_{20}(USA) · h_{20}(New York) · h_{20}(Albany)*∧
Product = h_{20}(Electronics) · h_{20}(Computer) · h_{20}(HP)*
Q2: Location = h_{20}(USA) · h_{20}(New York) · h_{20}(White Plains)*∧
Product = * (* means a wild card)

The proposed hash-concatenation scheme realizes cluster preserving encoding as follows. First, a variable-length string is encoded to fixed-length bits by hashing. Second, locality is preserved by concatenation. Bit keys with similar hierarchical strings are closely clustered.

Unfortunately, previous string-to-bit encoding schemes such as serial numbering and prefix encoding [5,35] are not feasible in peer-to-peer computing environments. The serial numbering scheme, which stores all mappings from strings to serial numbers, cannot add new values easily. If new values in the hierarchy are added, all peer nodes have to update their mapping function. Also, prefix encoding cannot categorize variable length strings well. Due to limited bit length, this scheme encodes only the prefix characters of a string. Thus, objects are not clustered well according to their hierarchy.

Step 2: Mapping multiple subordinate keys to an *N*-bit key and preserving multi-dimensional clustering – Many schemes have been studied to map a multi-dimensional key to a one-dimensional key. A Space Filling Curve (SFC) [2] is a well-known scheme for this purpose. There are different SFCs such as the z-ordering, the Gray code, and the Hilbert SFC. In CISS, we use the Hilbert SFC because it has superior clustering properties [18].

The Hilbert SFC is defined recursively. Fig. 5 shows the recursive construction of the Hilbert SFC in a two-dimensional space.[4] It begins with the first order approximation as in Fig. 5(a). For higher order approximations, each cube is divided into four sub-cubes, and the previous order approximation is replicated into each sub-cube. Then, the lower left sub-cube is rotated 90° clockwise and the lower right one, 90° counter-clockwise. The direction of traversal of both lower sub-cubes is reversed. The two upper sub-cubes neither rotate nor change direction of traversal. Given a two-dimensional key which consists of two b-bit keys, e.g., $\{x, y\}$, a one-dimensional key of $2b$-bits can be generated from the bth order approximation of the Hilbert SFC (see [18] for details). For instance, the two-dimensional key which consists of two 4-bit keys, $\{x = 1010 \wedge y = 1011\}$, is mapped to an 8-bit key, 10001011 using the 4th order approximation of the two-dimensional Hilbert SFC.

The Hilbert SFC nicely preserves *locality*. The points which are close to each other in a multi-dimensional space are mapped to the points which

[4] The Hilbert SFC in d-dimensional space ($d > 2$) is constructed in a similar way.

are close along the SFC. Thus, a multi-dimensional range is mapped to a few contiguous segments of the SFC, i.e., *clusters*. For example, in Fig. 6(a), a two-dimensional range (10*,*) is mapped to only two clusters. We utilize this property for efficient multi-dimensional range query routing to reduce the number of DHT lookups.

4.2. Efficient routing protocols for range queries and updates

To take the most benefit from object clustering, CISS supports two efficient routing protocols: a forwarding-based query routing protocol and a caching-based update routing protocol.

Forwarding-based query routing protocol: In CISS, a multi-dimensional range query involves multiple contiguous key ranges, i.e., multiple clusters. In order to reduce the number of DHT lookups, CISS only looks up the nodes corresponding to the first key of each cluster. Then, the query is simply forwarded to succeeding peer nodes until all relevant objects are retrieved.

See Fig. 6 as an example. A node $\mathbf{S_3}$ issues a multi-dimensional range query (10*,*). The query is mapped to the two dotted clusters in the gray area in Fig. 6(a). The first key of each cluster is calculated with the previously described LPF; they are 100000 and 110100, respectively. The query requester $\mathbf{S_3}$ searches for the matching peer nodes

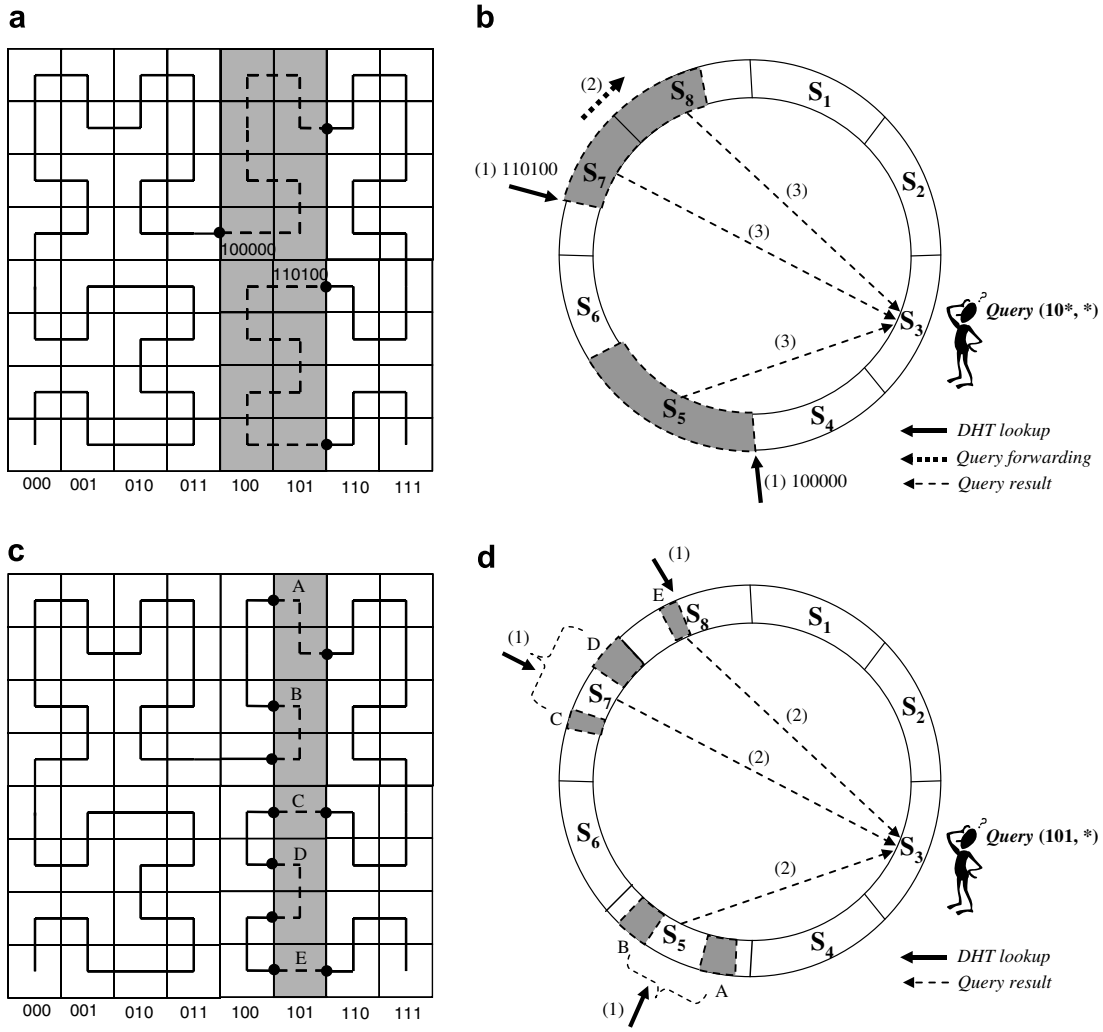


Fig. 6. Forwarding-based query routing protocol. (a) A query (10*,*), (b) a DHT-based P2P network, (c) a query (101,*), (d) a DHT-based P2P network.

corresponding to the two keys via DHT lookups. Then, it sends the query to the located nodes, i.e., $\mathbf{S_5}$ and $\mathbf{S_7}$. In particular, the peer $\mathbf{S_7}$ forwards the delivered query to $\mathbf{S_8}$ since the delivered query range is larger than its covering key range. The query respondents, which receive the query, generate results and send them back to the query requester. In this way, a multi-dimensional query can be efficiently resolved by cluster-based lookup and retrieval, avoiding separate lookups for each object.

CISS further improves the efficiency of query routing via cluster grouping. That is, a group of clusters is treated as a single unit in the routing process as long as the clusters share the DHT routing paths. Consider that a node **S** receives a range query which is mapped to a sequence of clusters, $\langle C_1, \ldots, C_n \rangle$, $1 \leqslant n$. In the sequence, the clusters are sorted in their increasing order of key ranges. That is, for each cluster C_i and C_{i+1}, $1 \leqslant i \leqslant n-1$, the values in the key range of C_{i+1} is larger than those of $C_{i.}$. Then, the node **S** partitions the clusters into multiple sub-sequences, where each subsequence is composed of the clusters which share the same node as their next hop.[5] Then, each subsequence, say, $\langle C_i, \ldots, C_k \rangle$, $1 \leqslant i \leqslant k \leqslant n$, is routed to the same next hop as a group via a single message. The next hop then receives the sub-sequence, $\langle C_i, \ldots, C_k \rangle$. This node may be the target of some clusters in the subsequence, say $C_i, C_{i+1}, \ldots, C_j$, for some $j, i \leqslant j \leqslant k$. If so, this node handles the matching clusters, i.e., $C_i, C_{i+1}, \ldots, C_j$. It may be the case that the last matching cluster, i.e., C_j, may not be fully covered by the target node. If so, query forwarding should be performed as explained. For the rest, it partitions the sequence $\langle C_{j+1}, \ldots, C_k \rangle$ as before and route each sub-sequence to the next hop. This process is repeated until all clusters reach their target nodes.

Fig. 6(c) and (d) shows an example. A node $\mathbf{S_3}$ issues a multi-dimensional range query (101,*). As shown in the gray area of Fig. 6(c), the query is mapped to the five dotted clusters: A, B, C, D and E. Fig. 6(d) shows that clusters A and B are covered by $\mathbf{S_5}$, and C and D are by $\mathbf{S_7}$. Hence, each pair can be resolved by a single DHT lookup, which results in three lookups for all five clusters.

In Squid [35], the authors suggested a mechanism to resolve multi-dimensional keywords and range queries by embedding a tree structure into a P2P network. This mechanism reduces the number of DHT lookups by recursively refining queries through the embedded tree. However, all queries should be initially routed to the peer node corresponding to the root of the tree. Thus, the peer easily becomes a congestion point and a single point of failure. However, the proposed forwarding-based query routing protocol in CISS does not incur such a query congestion problem while supporting efficient query processing with a small number of DHT lookups. In addition, the recursive refinement in Squid is somewhat sequential in that it traverses the embedded tree in a top-down fashion. Different from Squid, CISS can reduce the overall latency to finish the query resolving as it performs the DHT lookups for clusters in parallel.

Caching-based update routing protocol: In many P2P applications, successive updates often highly correlated with each other. That is, a sequence of updates shows high locality [23]. In CISS, we develop a caching-based update routing protocol on top of the object clustering to take advantage of such locality and further improve the system performance. Consider a MMOG as an example. A subsection of the virtual world is managed by a peer node via object clustering. Each player usually spends a significant amount of time in a given subsection, and therefore a number of successive updates generated by the player will belong to the same peer node with a high probability.

As shown in Fig. 7, the CISS client in each peer node implements a *key range cache*. It caches the key range of the most recently searched rendezvous node. Thus, the CISS client does not perform additional DHT lookups if an incoming update belongs to the cached key range (cache hit). In Section 5.1.2, we measure the *hit ratio* of the key range cache to quantify and show the performance benefit of our update routing protocol. The measurement has been performed by varying the degree of data mobility to see its impact to the hit ratio.

The cached key ranges in CISS clients can be stale due to the repartition of key ranges by node leaves or joins. In order to maintain strong consistency of the cached key ranges, a *server invalidation* mechanism is used [8,9,21]. Assume that a client sends an update to the CISS server of a wrong peer node due to a stale key range. The CISS server checks whether each update from a client falls in its current key range or not. Only when the update does not belong to its key range, it sends back an

[5] Given a key, i.e., the first key of a cluster, the next hop is the node whose ID is closest to the key among the entries in the DHT routing table [31,33,36,38].

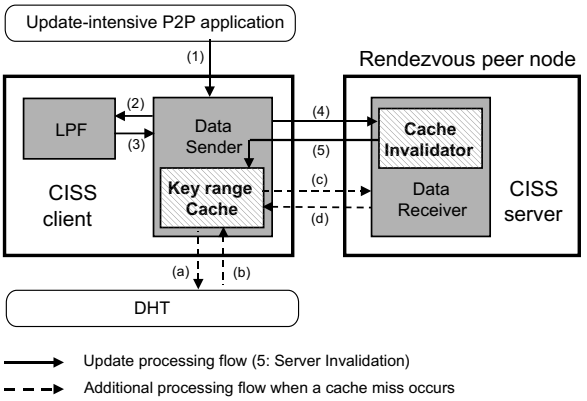


Fig. 7. Caching-based update routing protocol.

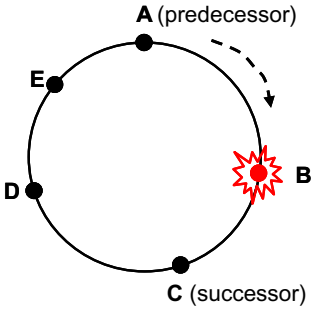


Fig. 8. Local-handover.

invalidation message to the requesting client. After receiving the invalidation message, the client performs a DHT lookup to locate a correct server. Then, it re-sends the lost update from a buffer to the correct server. We expect that the cached key range is fresh in most updates since node leaves or joins do not frequently occur compared to updates. Therefore, we think a small number of invalidation messages are enough to maintain the strong consistency of the key range cache.

4.3. Cluster-preserving load balancing

A basic idea of a DHT-based P2P network is to evenly distribute objects and requests to peer nodes across the network. This is achieved by the consistent hashing adopted in DHTs. Due to the resulting de-clustering, even a skewed distribution of objects or references is well balanced among different peer nodes. Hence, it is not likely that a system suffers from imbalance or hotspot in some parts. However, clustering semantically related objects collides with the idea of such a consistent hashing and even distribution of queries and objects. Hence, a skewed distribution of queries and objects may easily result in significant load imbalance. Hence, an effective object clustering framework should provide an effective load balancing scheme to resolve possible imbalance problem. What is important is that the load balancing scheme should preserve the object clustering property. CISS supports two cluster-preserving load balancing schemes: *local-handover* and *global-handover*.

Local-Handover: The overloaded node hands over a part of its own key range to one of its predecessor or successor. Fig. 8 shows an example of a local-handover. When node ***B*** gets overloaded, it hands over a part of its key range to its predecessor node ***A*** or successor node ***C*** as follows. If ***A*** is selected to take the load of ***B***, ***A*** leaves the P2P network and joins closer to ***B*** so as to take the part of ***B***'s key range.[6] This reduces the key range which ***B*** must manage. Similarly, ***C*** can take ***B***'s load. After the local-handover is performed, each node still manages a contiguous key range. Thus, object clustering is preserved. However, cascading load propagation may occur as a result of the handover. For instance, ***A*** may become overloaded and need to perform local-handover since it has to take over a portion of ***B***'s load as well as its original one. Similarly, predecessors (i.e., $\boldsymbol{E}, \boldsymbol{D}, \ldots$) may become overloaded succeedingly if a local-handover is performed.

Global-Handover: To alleviate the above mentioned shortcoming, we propose the global-handover. In this scheme, an overloaded node hands over a part of its key range to a non-neighbor node called *victim node*. After probing some randomly selected nodes in a P2P network, the most lightly loaded node is selected as a victim node. The victim node leaves from the network and hands over its entire key range to one of its neighbor nodes. Note that the leave of the victim node must not cause its successor node to be overloaded. If the sum of the successor node's current load and the victim node's load is larger than the successor's capacity, the next lightly loaded node is decided as a victim node. Fig. 9 shows an example. If node ***D*** is selected as a victim node, ***D*** leaves and then joins the network

[6] Node ***A*** does not leave the network physically. It just moves closer to ***B*** to take over a part of ***B***'s key range. For this purpose, CISS call a *leave* and a *join* function of DHT simultaneously after transferring objects.

Table 2
Load balancing cost

	Local-handover	Global-handover
DHT routing table update	• O(log S) messages	• O(log S) messages
Victim probing	• None	• k DHT lookups
Load information collection	• 2 neighbors	• k victims
Object transfer	• From the overloaded node to the neighbor node	• From the overloaded node to the victim node + From the victim node to the successor of victim node

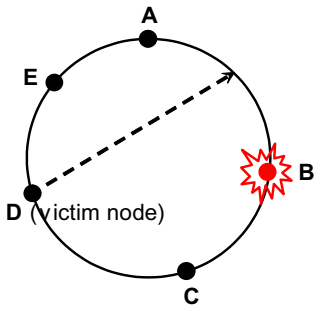


Fig. 9. Global-handover.

as a predecessor of ***B*** and takes over a contiguous sub-range of ***B***. Note that object clustering is still preserved in all nodes including ***E***. Also, there is no possibility of cascading load propagation.

For load balancing operations, each node maintains load information. A node divides its own key range into multiple sub-ranges and maintains load information for each sub-range. A sub-range is the basic unit for handover. Each node periodically measures the number of requests imposed on each sub-range. The load of a node is the sum of each sub-range's load. If the load is larger than its capacity, the node considers itself overloaded and performs load balancing.

The cost of the load balancing schemes is summarized in Table 2. It can be decomposed to four parts, i.e., *the cost for DHT routing table update*, *that for victim probing*, *load information collection* and *lastly the cost for object transfer*. The cost for updating the routing tables is the same for both local- and global-handover. For both cases, one leave and join of a node occur, incurring O(log S) messages. Upon a node leave or join, some nodes which have an entry for the node in their DHT routing tables should update the entry. In a network with S peer nodes, each node is listed in the routing tables of O(log S) peer nodes, and thus the number of nodes that need to be updated is O(log S). Hence, a node leave or join naturally results in O(log S) messages.[7] In global handover, ***k*** DHT lookups for victim probing are required, and then the load information is collected from the ***k*** selected nodes. In contrast, victim probing is not necessary in local-handover since load information can be directly collected from the successor and predecessor which are already known. The cost for object transfer can be measured as the number of objects transferred and their sizes. For local-handover, the transfer occurs from the overloaded node to neighbor nodes. Similarly, in global-handover, transfer occurs between the overloaded node and the selected victim node. However, the global-handover requires additional object transfers; the victim node needs to hand over its objects to its successor node beforehand. To minimize the load balancing cost, CISS prefers the local-handover and performs the global-handover only when cascading load propagation is expected.

Performing the proposed load balancing schemes may cause uneven range partition over peer nodes. Such an uneven range partition might affect the lookup performance of underlying DHT since basic DHTs assume evenly partitioned ranges by using the consistent hashing. Through extensive simulations in Section 5.2.3, we show that our load balancing schemes have negligible side effect on the lookup performance of the underlying DHT.

5. Performance evaluation

In this section, we demonstrate the performance benefits of CISS through extensive simulation studies. For the simulation, we have implemented a C++-based simulation engine which includes the Hilbert SFC-based LPF, the core functions of the

[7] While Pastry [33] and Tapestry [38] take O(log S) messages for a leave or a join, basic Chord [36] needs O($\log^2 S$) messages. However, it also has a sophisticate mechanism to take O(log S) messages.

Table 3
Notation of 2-dimensional range queries according to selectivity

Notation	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9	Q_{10}
Selectivity	2^{-24}	2^{-21}	2^{-18}	2^{-18}	2^{-15}	2^{-15}	2^{-12}	2^{-12}	2^{-9}	2^{-6}
Type	$Q(4,4)$	$Q(4,3)$	$Q(3,3)$	$Q(4,2)$	$Q(3,2)$	$Q(4,1)$	$Q(2,2)$	$Q(3,1)$	$Q(2,1)$	$Q(1,1)$

CISS client and server module, and the Chord-based DHT network. We evaluate the proposed query and update routing protocols and cluster-preserving load balancing schemes of CISS. We take account of diverse approaches for comparison. For the evaluation of query and update routing protocols, we consider two different methods, i.e., original DHT-based P2P systems [10,17,26] and Squid [35]. Also for the load balancing experiments, we make comparison with the virtual server-based method [7,10,30,36]. To see the side effect of the clustering framework to the basic DHT lookup, we consider two representative types of DHTs, namely, skip-list based (Chord [36]) and tree-based DHTs (Pastry [33]).

5.1. Query and update routing protocols

To analyze the performance of query and update routing protocols, we have performed the simulations with different number of nodes in a P2P network. The identifier of each node is randomly generated. We assume that there is neither node leave nor join to exclude the effects of dynamic topology changes.

5.1.1. Multi-dimensional range query performance

We evaluate the performance of multi-dimensional range query routing with 2-dimensional and 3-dimensional range queries. To generate multi-dimensional range queries, we consider a P2P auction as an example scenario. As a performance metric, the number of DHT lookups for query routing is used. For CISS, we test two types of query routing schemes: one with *per-cluster lookup* and the other with *cluster-grouping*. For comparison, we also evaluate DHT-based systems [10,17,26] which use a consistent hash function and Squid [35] which uses the recursive refinement based on Hilbert SFC.

5.1.1.1. Evaluating 2-dimensional range queries. To evaluate 2-dimensional range queries, we regard that a P2P auction uses two key attributes, Location and Product, which are String types consisting of four levels. Since the LPF constructs 24-bit keys in our simulation, each attribute is encoded using 12-bits, and thus the strings in each level are encoded to 3-bits. We generate all possible query types. For example,

- **Q(4, 4)**: Queries with both attributes having specific values in all four levels of the hierarchy, e.g., (location: *USA.New York.White Plains.79 North Broadway*, product: *Electronics. Computer.HP.Inkjet Printer*)
- **Q(4, 3)**: Queries with the first attribute having specific values in all four levels and the other attribute having values only in the top three levels, e.g., (location: *USA.New York.White Plains.79 North Broadway*, product: *Electronics. Computer.HP.**)

The other types of queries, i.e., $Q(4,2)$, $Q(4,1)$, $Q(3,3)$, $Q(3,2)$, $Q(3,1)$, $Q(2,2)$, $Q(2,1)$, and $Q(1,1)$, are similarly generated. In order to generalize the above queries, we calculate the *selectivity*[8] of the queries, which is defined as the ratio of the hypercubic fraction covered by a query over entire data space ($0 \leqslant selectivity \leqslant 1$). A query with a large selectivity means that the query retrieves objects from a large area in the data space. We denote the queries with a subscript in the increasing order of their selectivity as shown in Table 3.

Fig. 10(a) shows the average number of DHT lookups for ten types of queries for three different approaches. Three solid lines in the figure show the performances with CISS. The solid line labelled with a number is the case of using the per-cluster lookup. In this case, the number of DHT lookups does not depend on the selectivity of queries, but on the shape of queries. That is, the shape of queries is a main factor for determining the number of clusters. For example, queries like Q_1, Q_3, Q_7 and Q_{10} are mapped to just one cluster on the Hilbert SFC, and thus a single DHT lookup is sufficient.

[8] To select a set of queries which can generally represent multi-dimensional range queries is very difficult. We use a way which is commonly adopted for performance study in many research works in database community [12,34].

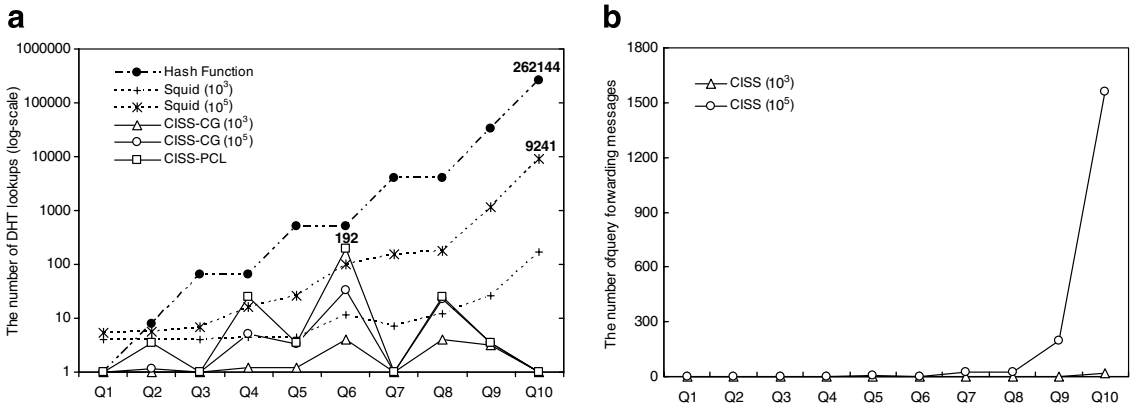


Fig. 10. Evaluation of 2-dimensional range queries: (a) # of DHT lookups and (b) # of forwarding messages.

Meanwhile, the number of nodes in a P2P network does not affect the number of lookups. The other two solid lines show the performances when using the cluster-grouping, one for the case with 10^3 nodes in the network and the other with 10^5 nodes. Regarding the query shapes, CISS with the cluster-grouping shows the similar behaviours as shown with the per-cluster lookup. However, the former achieves much better performance than the latter. This performance improvement gets larger as the number of peer nodes in the network decreases. This is because a node manages a relatively larger portion of the key range in a small-sized network, potentially covering multiple clusters.

In the case of using a hash function, the number of DHT lookups significantly increases with the selectivity of queries. As shown in the figure, CISS outperforms the hash-based case by orders of magnitude. Even in the worst cases such as Q_2, Q_4 and Q_6, CISS is much better than the hash-based approach. Such benefit of CISS stems primarily from object clustering.

The two dotted lines in the figure show the performances of Squid. As mentioned in Section 4.2, Squid reduces the number of DHT lookups by recursively refining queries through the embedded tree. However, a significant problem related to its performance is that all queries should be resolved starting from the same peer node. Thus, some peer nodes in the high level of the tree easily become congestion points or hotspots, seriously limiting the performance of the whole system. Such a congestion problem does not occur in CISS.

In Squid also, the number of DHT lookups increases with the selectivity of queries. In general, a larger selectivity means that more number of branches in the embedded tree involves with the query. It results in more recursive refinements, hence more DHT lookups. More interestingly, the number of DHT lookups increases with the number of peer nodes in the P2P network. When there are a small number of nodes, each node manages relatively large portion of key space. In this situation, each node manages several clusters associated with a query region. Thus, the recursive refinement in Squid can resolve multiple clusters by visiting a target node. However, this benefit diminishes as the number of peer nodes increases. Consider a P2P network with a large number of peer nodes. Only one cluster may reside in a peer node, or even one cluster may span over multiple peer nodes. This requires the recursive refinement to be executed to the deeper level of the embedded tree. Moreover, some intermediary nodes, e.g., the non-leaf nodes of the tree, are visited in the process of recursive refinement, which does not occur in CISS. The number of visits to such intermediary nodes can be often high compared to the number cluster to retrieve. Consequently, CISS with per-cluster lookup is more efficient than Squid when the number of nodes in a P2P network is large. When using the cluster-grouping, CISS is superior to Squid in all query types even with the small number of nodes.

Fig. 10(b) shows the average number of query forwarding messages in CISS. The number of forwarding messages is the same for both the per-cluster lookup and the cluster-grouping. As shown in the figure, query forwarding is not needed in most cases, i.e., Q_1 through Q_9 in the case with 10^3 nodes and Q_1 through Q_6 with 10^5 nodes. The other types of queries, which have a high selectivity, require query forwarding since the query range is larger than the

key range of peer nodes. Also, the number of messages for query forwarding increases with the number of nodes in a P2P network since the key range size of a node becomes smaller. However, query forwarding is much cheaper than a DHT lookup.

5.1.1.2. Evaluating 3-dimensional range queries. To evaluate 3-dimensional range queries, we regard that the P2P auction uses three key attributes, i.e., location, product, and phone number, all String types consisting of four levels. Since the LPF constructs 24-bit keys in our simulation, each attribute is encoded using 8-bits, and thus each level's String is encoded to 2-bits. Similar to the 2-dimensional case, we generate all possible query types, i.e., 20 types. We also calculate the *selectivity* of the queries and denote the queries with a subscript in the increasing order of their selectivity as shown in Table 4.

Fig. 11(a) shows the average number of DHT lookups for the 20 types of queries for three different approaches. Fig. 11(b) shows the average number of query forwarding messages in CISS. Generally, the results show similar patterns to the cases of 2-dimensional range queries. The main difference is that CISS and Squid require more DHT lookups. This is because more clusters are generated in a higher dimensional SFC with the same value of selectivity. For example, when using the per-cluster lookup, the maximum number of DHT lookups in CISS increases from 192 to 1224 while that in Squid increases from 9241 to 13,318. However, CISS still outperforms the hash-based approach by orders of magnitude. Comparing CISS with Squid, its superiority slightly decreases if the per-cluster lookup is used. In the case of 2-dimensional queries with 10^5 nodes, CISS is worse than Squid with two query types, i.e., Q_4 and Q_6, among 10 types, which is 20% of query types. In the case of 3-dimensional range queries, CISS is worse than Squid with 10^5 nodes in six types, i.e., Q_6, Q_7, Q_9, Q_{10}, Q_{13}, and Q_{16}, among 20 types, which is 30%. However, CISS still shows better performance in 13 query types, i.e., 65%. With the cluster-grouping, CISS is superior to Squid regardless of query types and the number of nodes as in the 2-dimensional case.

5.1.2. Update performance

To simulate continuous updates, we use an MMOG scenario. Each node generates the continuous

Table 4
Notation of 3-dimensional range queries according to selectivity

Notation	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9	Q_{10}
Selectivity	2^{-24}	2^{-22}	2^{-20}	2^{-20}	2^{-18}	2^{-18}	2^{-18}	2^{-16}	2^{-16}	2^{-16}
Type	$Q(4,4,4)$	$Q(4,4,3)$	$Q(4,3,3)$	$Q(4,4,2)$	$Q(3,3,3)$	$Q(4,3,2)$	$Q(4,4,1)$	$Q(3,3,2)$	$Q(4,2,2)$	$Q(4,3,1)$
Notation	Q_{11}	Q_{12}	Q_{13}	Q_{14}	Q_{15}	Q_{16}	Q_{17}	Q_{18}	Q_{19}	Q_{20}
Selectivity	2^{-14}	2^{-14}	2^{-14}	2^{-12}	2^{-12}	2^{-12}	2^{-10}	2^{-10}	2^{-8}	2^{-6}
Type	$Q(3,2,2)$	$Q(3,3,1)$	$Q(4,2,1)$	$Q(2,2,2)$	$Q(3,2,1)$	$Q(4,1,1)$	$Q(2,2,1)$	$Q(3,1,1)$	$Q(2,1,1)$	$Q(1,1,1)$

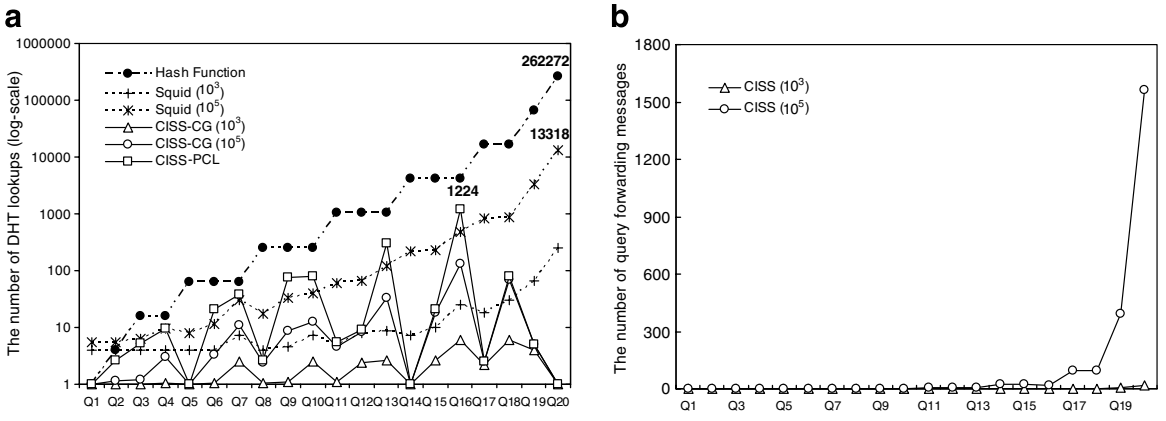


Fig. 11. Evaluation of 3-dimensional range queries: (a) # of DHT lookups and (b) # of forwarding messages.

updates based on a "*mobile avatar*" model. An avatar is the representation of a player's character in a virtual game world. The mobile avatars are designed to wander a $[0, 2^{12}] \times [0, 2^{12}]$ square virtual world based on the ns-2 *random waypoint mobility model* [5,11]. They update their location every 125 ms (for comparison, the first-person shooting game, *Quake II* [43] updates an avatar's location every 50 ms). A location consists of two attributes: x and y coordinates. Before updating its location, each mobile avatar checks whether its current location is in the cached key range. If a cache miss occurs, it looks up the node that is responsible for its current location. The simulation runs for 300 s with three P2P network topologies: 10^3, 10^4 and 10^5 nodes. As a performance metric, we measure the hit ratio of the key range cache.

Fig. 12 shows the average hit ratio of the key range cache as a function of mobility values. A mobility value of 1 means that an avatar can move at most one pixel in the virtual world during an update period. As shown in the figure, our update routing protocol significantly reduces the number of lookups for location updates (by up to 93% with 10^5 nodes). Since the range of mobile avatar movement is much smaller than the range managed by the responsible server, the hit ratio is high with low mobility values. The larger the mobility value, the lower the hit ratio. However, the update routing protocol still achieves a 35% hit ratio with 10^5 nodes even with a high mobility value of 256, i.e., 6.25% of side length of the virtual world. Fig. 12 also shows the hit ratio of three different sizes of network. The key range managed by each node increases as the number of peer nodes decreases. Thus, the hit ratio is the highest in the case with 10^3 nodes.

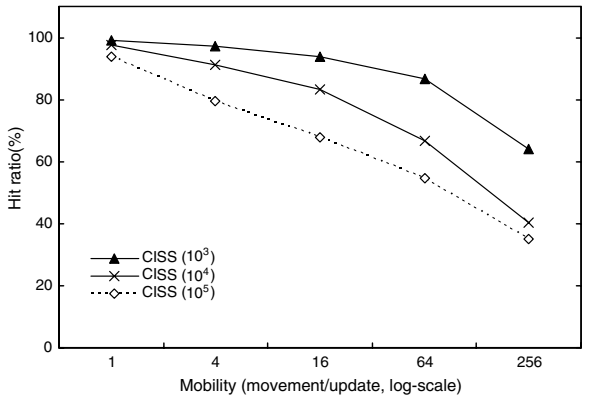


Fig. 12. Hit ratio of the key range cache.

5.2. Load balancing

5.2.1. The effectiveness and cost of load balancing schemes

We consider a MMOG scenario to evaluate our load balancing schemes. In MMOGs, a huge number of avatars, i.e., players' characters wander around from place to place and interact with each other in a virtual world assuming a real or fantasy world. For instance, 300 K players concurrently play in the same virtual world while two million players are registered in popular MMOGs, e.g., Lineage [41] and World of Warcraft [42]. In such a large-scale MMOG environment, multiple servers cooperatively manage the state of a game, which is typically composed of game players' status in a virtual world. Game clients frequently request two types of primitive operations to servers. First, they inquire the game state over nearby regions of the virtual world, which can be treated as two-dimensional range queries. They also continuously generate streams of updates to the game state, e.g., players' locations and interaction messages. Meanwhile, players can be easily crowded in a specific region of the virtual world, which potentially incurs hotspots in the corresponding MMOG servers and thus requires load balancing.

Based on the above real MMOG scenario, we set up a simple simulation environment as follows. The server modules of peer nodes cooperatively act as game servers. Also, each server module manages the zone which is a partial region of the virtual world. The client module of a node corresponds to a game player. We regard that the player' mobile avatar is the object which is stored in the corresponding server module. Each player periodically updates the avatar's attributes, i.e., the location in the $[0, 2^{12}] \times [0, 2^{12}]$ square virtual world.

To determine whether the server module of a node is overloaded or not, each node maintains load information by dividing its own key range into 64 sub-ranges. Each node measures its load by counting the number of location updates during every 240 s period. If the total load is larger than node capacity, i.e., 1000, the node is considered as overloaded, and then it performs load balancing. The number of overloaded nodes is counted every load checking period.

To simulate load imbalance among peer nodes, we generate the situation where many avatars stay in a specific region. In particular, avatars are densely located around the point (0, 0) and sparely

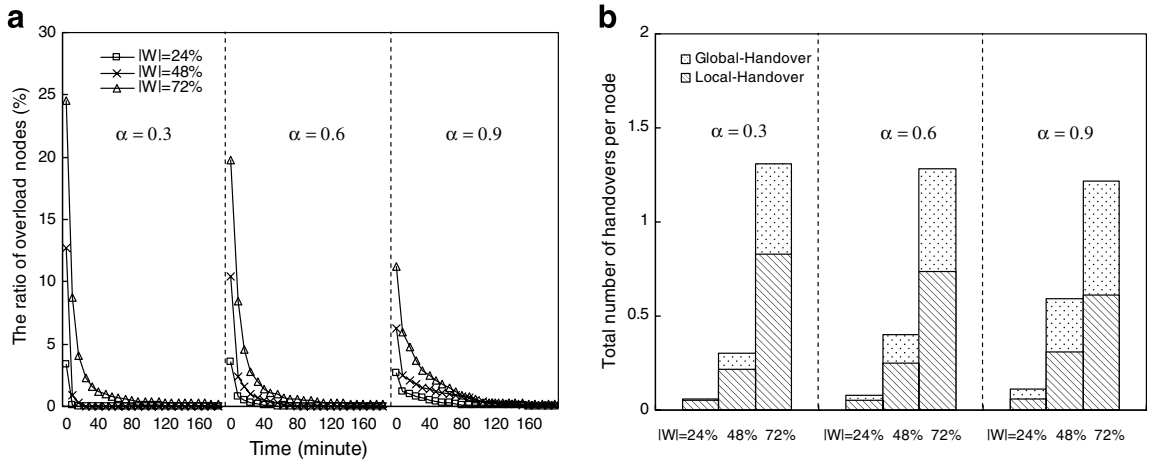


Fig. 13. Effectiveness of load balancing for skewed workloads: (a) Ratio of overloaded nodes and (b) total number of handovers per node.

located around the point $(2^{12}, 2^{12})$ in the virtual world. Such skewed x and y location data are generated by using a *Zipf* distribution,[9] which has probability density function (PDF) $x^{-\alpha}$ where α is a constant less than 1. We apply three different workloads: α value of 0.3, 0.6 and 0.9. The larger the α value, the more skewed the distribution. Also, we vary workload size by changing update rates, i.e., 1, 2 and 3 updates/s. The workload size, $|W|$ corresponds to 24%, 48% and 72% of the capacity of each node, e.g., $1/\text{s} \times 240\ \text{s}/1000 = 24\%$. Note that nodes are uniformly distributed at the beginning of each simulation. Each simulation is performed for 200 min, and the total number of nodes is fixed to 10^4.

We measure two performance metrics of load balancing schemes. First, we measure the effectiveness of load balancing, which consists of *the ratio of overloaded nodes over time* and *the total number of handovers performed until balanced*. We regard that the whole system is well balanced with high probability when the ratio of overloaded nodes is less than 1%. Second, we measure the cost of load balancing, i.e., *message overhead* and *object transfer overhead*.

Fig. 13(a) shows the ratio of overloaded nodes over time. For all skewed workloads, a hotspot region is rapidly developed right after a workload is applied and thus the ratio of overloaded nodes increases very quickly, e.g., 25% for $\alpha = 0.3$ and $|W| = 72\%$. However, it decreases sharply and is stabilized to near 0% by our load balancing schemes. Fig. 13(b) shows the total number of handovers per node until loads are balanced. For all skewed workloads, it is less than two, which is a reasonably small number. The results shown in Fig. 13(a) and (b) demonstrate that the proposed load balancing schemes work effectively even for skewed workloads.

It is worth taking a close look at the graphs in Fig. 13 to better understand the underlying behaviour of the system. First, with the same skewness of workload, the number of initially overloaded nodes increases as the workload size increases (see Fig. 13(a)). Thus, the total number of handovers per node increases as shown in Fig. 13(b).

Second, the total number of handovers increases with workload skewness. However, its amount of change varies with workload size. That is, with small workload sizes, e.g., $|W| = 24\%$ and 48%, the number of handovers increases with the skewness. (See the change of heights of graphs for different α values in Fig. 13(b).) However, with a large workload size, e.g., $|W| = 72\%$, the number does not increase. The detailed reason is as follows. When the workload size is small, the number of initially overloaded nodes is similar even though workload skewness increases. (This can be seen in Fig. 13(a)) Also, we can conjecture that the overloaded nodes are more highly loaded in the highly skewed case. Thus, the overloaded nodes under more skewed workloads perform more handovers to be stabilized, thereby increasing the total number of handovers per node. In contrast, with a large

[9] The zipf is a well-known high-skewed distribution. If $\alpha \geqslant 0.9$, it is very highly skewed. If α is 0, the distribution is uniform distribution.

workload size, the number of initially overloaded nodes decreases with skewness. (See again Fig. 13(a).) We also can conjecture that the overloaded nodes are more highly loaded in the highly skewed case. Therefore, the total number of handovers becomes almost same even though workload skewness increases.

An interesting observation is that the ratio of global handover compared to local handover increases with the workload skewness. It is because, with a highly skewed workload, multiple adjacent nodes in the hotspot get overloaded at the same time. Therefore, the overloaded nodes prefer to perform global-handovers since local-handovers may cause cascading load propagation.

Fig. 14 shows the cost of the load balancing schemes. First, Fig. 14(a) summarizes the message overhead, i.e., the average number and volume of messages per handover. We do not explicitly simulate leaves, joins and lookups in DHT during a local- or global-handover. Rather, we examine the number of messages upon those events. Theoretically, $O(\log S)$ messages are generated upon a leave, a join or a lookup, where S is the number of nodes. However, it is known that $(1/2)\log S$ messages are generated in average [37]. Based on the above fact, we derive the average number of generated messages for DHT routing table updates due to a leave followed by a join, i.e., 13.29. Due to victim probing, a global-handover additionally generates 53.16 messages in average. In our simulation, we set S to 10^4 and $\boldsymbol{k}$ to 8. In addition, we estimate the size of DHT message and load information message. For a DHT message, 25 bytes are reasonable since a DHT message contains only an N-bit key and a message tag representing an event type. Also, we assume the 256 bytes of load information message which contains the load values of 64 sub-ranges; the value is represented by a 4-byte integer. Therefore, the volume of messages per both local- and global-handover is small (<4KB), and hence the message cost is not significant.

Fig. 14(b) shows the object transfer overhead, i.e., the average number of objects transferred per handover under different workload sizes and skewness. As expected, a global handover transfers more objects than a local handover. In all cases, the number of objects transferred per handover is less than 20 (0.2% of the total objects). In general, the object transfer overhead highly depends on the object sizes.

Let us see the graphs in Fig. 14(b) more closely. First, as the workload size increases, the average number of transferred objects decreases. It is because the load given to each object gets larger with a high update rate, thereby a large workload size. Thus, transferring a small number of objects has a relatively big impact in decreasing the load level. Second, the number of transferred objects increases with workload skewness. However, its amount of change varies with workload size. That is, with a small workload size, e.g., $|W| = 24\%$ or 48%, the number of transferred objects increases with the skewness. (See the change of heights of graphs for different α values in Fig. 14(b).) However, with a large workload size, e.g., $|W| = 72\%$, the number does not increase.

Such a delicate trend is because our method tries to avoid cascading load propagation. As mentioned before, overloaded nodes are more heavily loaded

a

	Local-Handover		Global-Handover	
	# of messages	message size (byte)	# of messages	message size (byte)
DHT routing table update	13.29	25	13.29	25
Victim probing	-	-	53.16	25
Load information collection	2	256	8	256
Total volume	0.8 KB		3.7 KB	

b

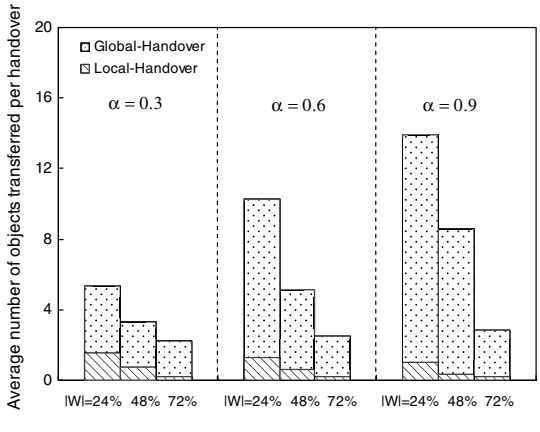


Fig. 14. Cost of load balancing: (a) Message overhead and (b) object transfer overhead.

with a more highly skewed workload. Thus, with small workload sizes, a handover transfers more objects as the skewness increases. In contrast, when the workload size is large, the load caused by an object gets large. Thus, transferring even a small number of objects shifts a large amount of load. Since our load balancing method avoids cascading load propagation, the number of objects transferred is limited even though the skewness increases.

5.2.2. Comparison with virtual server-based methods

In order to show the benefit of the proposed cluster-preserving load balancing method, we compare it with the virtual server-based load balancing method [7,10,30,36]. In the simulation with the virtual server-based method, the number of virtual servers which can be allocated to a physical peer node is set to 2, 5, and $\log N$, where N is the total number of physical peer nodes [7,10,30,36]. We randomly assign a node identifier to each virtual server. This is to enforce the virtual servers disperse to different physical nodes. Thus, the whole key space is partitioned to, at the maximum, $2N$, $5N$ and $N\log N$ virtual nodes. For a meaningful comparison with CISS, the virtual server-based method has been adapted for range query and update processing.

We examine how well the clusters are preserved with the two methods by comparing two metrics, i.e., the number of peer nodes involving in processing a range query and the hit ratio of the key range cache for updates. These metrics are measured assuming that two load balancing methods have been applied to the systems. The skewed key range arrangements after load balancing is simulated with the key range distribution following the Zip distribution with α of 0.7.

Fig. 15 presents the number of nodes required for processing a query. For this, we randomly generate 5000 queries of Q_7 type, which has been described in Section 5.1.1.1. As shown in Fig. 15(a), more peer nodes involve in processing a query with the virtual server-based method than with CISS. This is because the key ranges are more partitioned in the virtual server-based method. As the number of virtual server per a physical node increases, the key range of each virtual server becomes shorter and the number of nodes for processing a query increases.

Fig. 15(b) compares the two in terms of the cumulative density function (CDF). With no more than five nodes, more than 70% of the queries are resolved in CISS, while merely 20% are resolved in the virtual server-based method. In conclusion, the proposed cluster-preserving approach better maintains object clustering, and thus better support range query processing.

Fig. 16 shows the hit ratio of the key range caches. Each simulation follows the previous MMOG scenario in Section 5.1.2 for its update workload, and runs for 5000 s. Fig. 16(a) shows that the average hit ratio with CISS is higher than that with the virtual server-based method. It is because key range in CISS is much longer than that in the virtual server-based method. Fig. 16(b) compares the two in terms of cumulative density function (CDF) of the average hit ratio. No more than 1% of nodes in CISS show less than 60% of hit ratio, whereas 15% show less than 60% of hit ratio in the virtual server-based method. In conclusion, CISS also outperforms the virtual server-based method in update processing by better preserving object clustering.

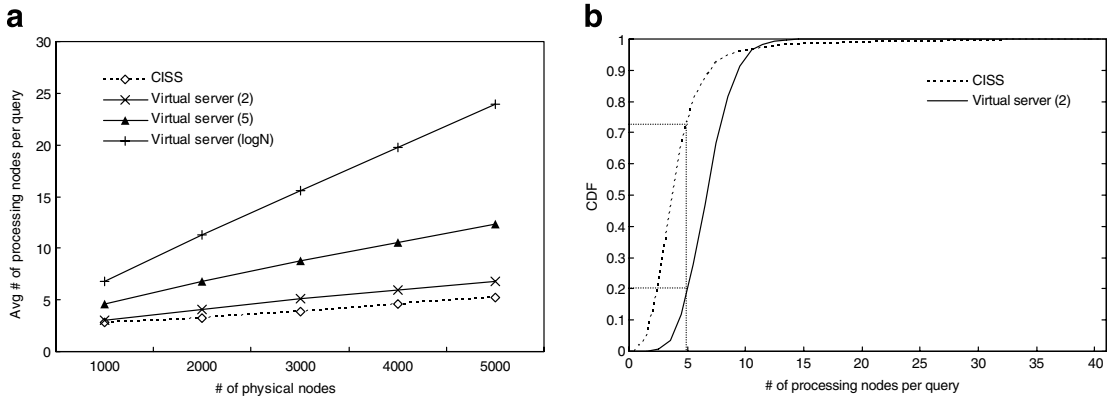


Fig. 15. (a) Average number of processing nodes per query. (b) CDF of the processing nodes per query; the number of physical nodes is 5000.

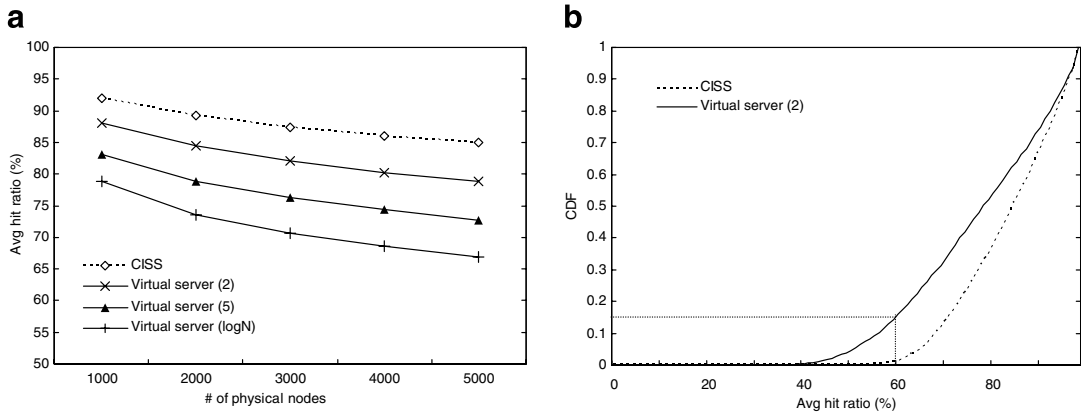


Fig. 16. (a) Average hit ratio of the key range cache and (b) CDF of the average hit ratio; the number of physical nodes is 5000.

5.2.3. Side effect on DHT lookup performance

In this section, we identify the side effect of load balancing on DHT lookup performance. As mentioned before, performing cluster-preserving load balancing may cause highly uneven key range distribution among peer nodes. Such an uneven range partition might affect the basic lookup performance of the underlying DHT since a DHT originally assumes evenly partitioned ranges by using a consistent hashing.

To analyze the side effect, we consider two types of DHT models: one with skiplist-like routing and the other with tree-like routing, which are representative DHT models supporting one-dimensional lookups [4]. As mentioned, CISS is designed to run on top of any DHTs supporting one-dimensional lookups. Chord [36] is chosen for skiplist-like routing, and Pastry [33] is for tree-like routing. For simulation, we extend the basic Chord code available from [39]. For Pastry code, we use SimPastry [40] with b of 4 and L of 8.

To simulate a skewed distribution of range sizes, the key ranges are assigned using a Zipf distribution. We also use the same distribution for the query workload, i.e., key lookup. As a metric for DHT lookup performance, we measure the number of hops needed to resolve a DHT lookup. We present the simulation results for three Zipf distributions with different skew parameters and also a uniform distribution for comparison.

Fig. 17 shows results for Chord. In Fig. 17(a), the average number of hops needed for a DHT lookup increases logarithmically with the number of nodes for both uniform and skewed distributions. Basically, a DHT lookup in Chord can be resolved with $O(\log S)$ messages, where S is the number of

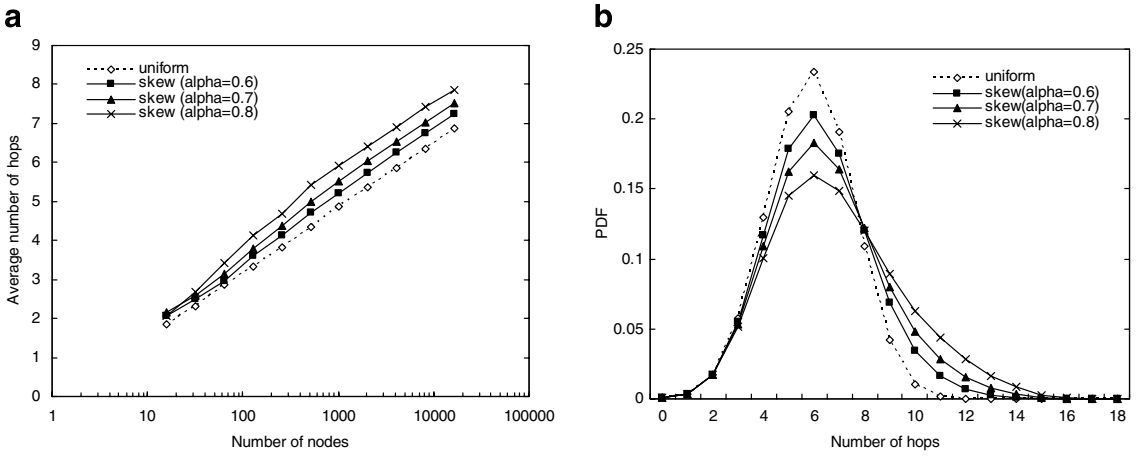


Fig. 17. Chord: (a) average number of hops and (b) PDF of the number of hops.

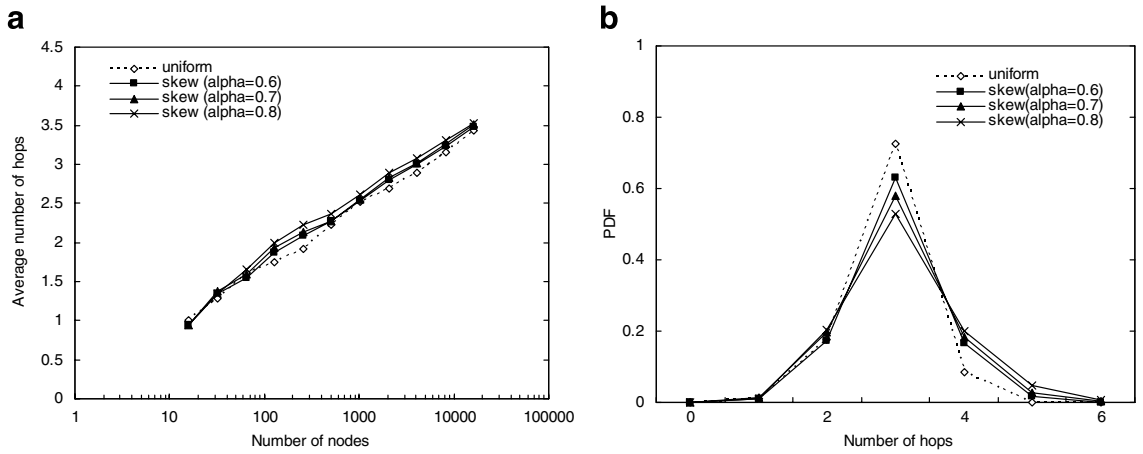


Fig. 18. Pastry: (a) average number of hops and (b) PDF of the number of hops.

nodes in the network. The result shown in the figure supports such a logarithmic hop routing performance. Also, with the same number of nodes, the average number of hops slightly increases with the skewness. However, the difference between the uniform and the skewed cases is not considerable and remains constant, i.e., almost one hop. Fig. 17(b) presents the probability density function (PDF) of the number of hops where the total number of nodes is 2^{12}. In the cases with skewed distribution, the maximum number of hops is larger than that with the uniform distribution. However, it is no more than 18 hops even in the worst case and 13 hops with a 99th percentile.

Fig. 18 shows results with Pastry as the underlying DHT. Fig. 18(a) presents that, as in Fig. 17(a), the average number of hops increases logarithmically with the number of nodes for uniform and skewed distributions. Also, the difference between uniform and skewed distributions is very small. The result approximates the expected hop of Pastry, $\log_{2^b} S$. Fig. 18(b) presents the PDF. Although the peak PDF value decreases and the tail increases slightly as the skewness increases, the maximum number of hops is 6 regardless of the skewness. In conclusion, we expect that uneven range partition due to the load balancing does not have much effect on the lookup performance when using Chord or Pastry as an underlying DHT.

6. Conclusion

In this paper, we have presented CISS, a framework for efficient object clustering for DHT-based P2P applications. While DHTs [31,33,36,38] have a high potential as a scalable infrastructure for Internet-scale data management, it is still deficient in supporting advanced access operations such as multi-dimensional range queries or continuous updates. It is mainly because it organizes objects in a highly declustered fashion. The proposed framework effectively supports object clustering, complimenting such deficiency of the DHT-based approach. As semantically related objects are clustered, multiple key values can be searched for via a single DHT lookup, thereby efficiently supporting range queries. It is also advantageous upon continuous updates as successive updates are often semantically close. Therefore, the cost as well as the latency of range queries and updates can be considerably reduced.

CISS addresses several technical challenges, i.e., Locality Preserving Function, query and update routing protocols, and cluster-preserving load balancing. A high level of object clustering is achieved by using the LPF on top of the existing DHTs without requiring much change to the DHTs. Efficient query and update routing protocols maximize the benefit of object clustering by significantly reducing the number of DHT lookups. The load balancing schemes well distribute loads among peer nodes while preserving the clustering property.

Our extensive simulation studies show that CISS is very beneficial to support advanced access operations. Also, the cluster-preserving load balancing works well even under highly skewed workload, and has negligible side effects on the lookup performance of underlying DHT. We believe that CISS forms an important building block to efficiently support DHT-based P2P applications with advanced access operations.

References

[1] A. Andrzejak, Z. Xu, Scalable, Efficient range queries for grid information services, in: Proceedings of IEEE P2P, Sweden, September 2002.

[2] T. Asano, D. Ranjan, T. Roose, E. Welzl, P. Widmaier, Space filling curves and their use in geometric data structures, Theoretical Computing Science 181 (1997) 3–15.

[3] J. Aspnes, G. Shah, Skip graphs, in: Proceeding of SODA, 2003.

[4] H. Balakrishnan, M. Kaashoek, D. Karger, R. Morris, I. Stoica, Looking up Data in P2P Systems, Communications of the ACM 46 (2) (2003).

[5] A. Bharambe, S. Rao, S. Seshan, Mercury: a scalable publish-subscribe system for internet games, in: Proceedings of NetGames, Germany, April 2002.

[6] A. Bharambe, M. Agrawal, S. Seshan, Mercury: Supporting scalable multi-attribute range queries, in: Proceedings of ACM SIGCOMM, August 2004.

[7] J. Byers, J. Considine, M. Mitzenmacher, Simple load balancing for distributed hash tables, in: Proceedings of IPTPS, CA, USA, February 2003.

[8] K.S. Candan, D. Agrawal, W. Li, O. Po, W. Hsiung, View invalidation for dynamic content caching in multitiered architectures, VLDB'02.

[9] S. Choi, J. Lee, S. Kim, S. Jo, J. Song, Y. Lee, Accelerating database processing at multi-tier web sites, EC-Web'04.

[10] F. Dabek, M.F. Kaashoek, D. Karger, R. Morris, I. Stoica, Wide-area cooperative storage with CFS, in: Proceedings of SOSP, Canada, October 2001.

[11] K. Fall, K. Varadhan, NS Manual.

[12] P. Ganesan, B. Yang, H. Molina, One torus to rule them all: multidimensional queries in P2P systems, in: Proceeding of WebDB, Paris, France, June 2004.

[13] P. Ganesan, M. Bawa, H. Molina, Online balancing of range-partitioned data with applications to peer-to-peer systems, in: Proceeding of VLDB, Toronto, Canada, September 2004.

[14] A. Gupta, D. Agrawal, A. El Abbadi, Approximate range selection queries in peer-to-peer systems, in: Proceedings of CIDR, CA, USA, January 2003.

[15] M. Harren, J.M. Hellerstein, R. Huebsch, B.T. Loo, S. Shenker, I. Stoica, Complex queries in DHT-based peer-to-peer networks, in: Proceedings of IPTPS, MA, USA, March 2002.

[16] N.J.A. Harvey, M. Jones, S. Saroiu, M. Theimer, A. Wolman, Skipnet: a scalable overlay network with practical locality properties, in: Proceeding of USITS, 2003.

[17] R. Huebsch, J.M. Hellerstein, N. Lanham, B.T. Loo, S. Shenker, I. Stoica, Querying the internet with PIER, in: Proceedings of VLDB, Berlin, September 2003.

[18] H.V. Jagadish, Linear clustering of objects with multiple attributes, ACM SIGMOD, 1990.

[19] B. Knutsson, H. Lu, W. Xu, B. Hopkins, Peer-to-peer support for massively multiplayer games, in: Proceedings of INFOCOM, Hong Kong, China, March 2004.

[20] A. Kothari, D. Agrawal, A. Gupta, Subhash Suri, Range addressable network: a P2P cache architecture for data ranges, in: Proceedings of IEEE P2P, Sweden, September 2003.

[21] B. Krishnammurthy, C.E. Wills, Piggyback server invalidation for proxy cache coherency, WWW'98.

[22] J. Lee, H. Lee, S. Kang, S. Choe, J. Song, CISS: an efficient object clustering framework for DHT-based peer-to-peer applications, in: Proceeding of DBISP2P, collocated with VLDB, Toronto, Canada, August 2004.

[23] J. Lee, S. Kang, S. Choi, H. Jin, S. Choe, J. Song, LARI: locality-aware range query index for high performance data stream processing, Technical Report CS-TR-2004-202, August 2004.

[24] J. Mischke, B. Stiller, A Methodology for the Design of Distributed Search in P2P Middleware, IEEE Network 18 (1) (2004) 30–37.

[25] A. Misra, P. Castro, J. Lee, CLASH: a protocol for internet-scale utility-oriented distributed computing, in: Proceedings of ICDCS, Japan, March 2004.

[26] A. Muthitacharoen, R. Morris, T.M. Gil, B. Chen, Ivy: a read/write peer-to-peer file system, OSDI 2002.

[27] A. Oram (Ed.), Peer-To-Peer: Harnessing the Power of Disruptive Technologies, O'Reilly & Associates, Sebastopol, USA, 2001.

[28] S. Padmanabhan, B. Bhattacharjee, T. Malkemus, L. Cranston, M. Huras, multi-dimensional clustering: a new data layout scheme in DB2, in: Proceeding of SIGMOD, USA, June 2003.

[29] V. Papadimos, D. Maier, K. Tufte, Distributed query processing and catalogs for peer-to-peer systems, in: Proceedings of CIDR, CA, USA, January 2003.

[30] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, I. Stoica, Load balancing in structured P2P systems, in: Proceedings of IPTPS, CA, USA, February 2003.

[31] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, A scalable content-addressable network, in: Proceedings of SIGCOMM, CA, USA, August 2001.

[32] S. Ratnasamy, J.M. Hellerstein, S. Shenker, Range queries over DHTs, IRB-TR-03-009, June 2003.

[33] A. Rowstron, P. Druschel, Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems, in: Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Germany, November 2001.

[34] O. Sahin, A. Gupta, D. Agrawal, A. El Abbadi, A peer-to-peer framework for caching range queries, in: Proceedings of ICDE, MA, USA, March 2004.

[35] C. Schmidt, M. Parashar, Flexible information discovery in decentralized distributed systems, in: Proceedings of HPDC, WA, USA June 2003.

[36] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnan, Chord: a scalable peer-to-peer lookup service for internet applications, in: Proceedings of SIGCOMM, CA, USA, August 2001.

[37] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, H. Balakrishnan, Chord: A scalable peer-to-peer lookup protocol for internet applications, IEEE/ACM Transactions on Networking (TON) 11 (1) (2003).

[38] B.Y. Zhao, J. Kubiatowicz, A. Joseph, Tapestry: an infrastructure for fault-tolerant wide-area location and routing, UCB Tech. Report UCB/CSD-01-1141.

[39] http://www.pdos.lcs.mit.edu/chord.

[40] http://research.microsoft.com/~antr/Pastry/.

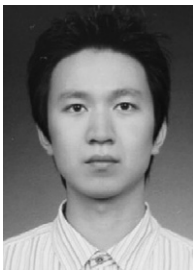
[41] http://www.lineage.com.

[42] http://www.worldofwarcraft.com/.

[43] http://www.idsoftware.com/business/techdownloads.

[44] http://gnutella.wego.com/.
[45] http://freenet.sourceforge.net.

Jinwon Lee is a PhD candidate at the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea. He received the B.S. degree in Electrical Engineering from the KAIST in 2001, and M.S. degree in Computer Science from the KAIST in 2003. His research interests include data stream processing system, peer-to-peer overlay network, and high-performance Internet cache system. He received the best paper award at the 7th International Conference on Mobile Data Management (MDM 2006).



Hyonik Lee is a graduate student at the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea. He received the B.S. degree in Electrical Engineering from KAIST, in 2002, and M.S. degree in Computer Science from KAIST in 2005. His research interest lies in Internet Technologies, Ubiquitous Computing, Distributed Systems and Overlay Networks.

Seungwoo Kang is a graduate student at the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea. He received the B.S. degree in Electrical Engineering from the Yonsei University, Seoul, Korea, in 2000, and M.S. degree in Electrical Engineering from KAIST in 2002. His research interest lies in Internet Technologies, Internet-scale distributed computing systems.

Su Myeon Kim received his BSc and PhD in Electronic Engineering and Computer Science from the Korea Advanced Institute of Science and Technology in 1997 and 2006. He is currently a research and development staff member at Samsung Advanced Institute of Technology. His research interest include ubiquitous systems, stream processing, service oriented computing, and internet computing.

Junehwa Song is an associate professor, Department of EECS, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea. Before joining KAIST, he worked at IBM T.J. Watson Research Center, Yorktown Heights, NY as a Research Staff Member from 1997 to September 2000. He received his Ph.D. in Computer Science from University of Maryland at College Park in 1997. His research interest lies in Internet Technologies, such as intermediary devices, high performance Web serving, electronic commerce, etc., and distributed multimedia systems.