A New Framework to Design Distributed Query System

Yangming Zhao, Sheng Wang, Huacheng Cai, Xiong Wang
School of Communication and Information Engineering
University of Electronic Science and Technology of China,
Chengdu, China
zhaoyangming@uestc.edu.cn, wsh_keylab@uestc.edu.cn, chcacy@163.com, wangxiong@uestc.edu.cn

***Abstract*—This paper proposes a new framework to design query systems which could be used in DNS systems. An efficient ILP model, which is able to take network layer information into count, is formulated to design such query systems. By simulation, we demonstrate that query system in our framework can relax unnecessary constraint and trade off between query delay and storage cost.**

Keywords-query system design; ILP (Integrate Linear Programming); system availability

I. INTRODUCTION

These days the query system plays a vital role in our daily life[1]. For example, in the Internet which greatly affects our daily commercial, social and culture activities, DNS (Domain Name Server) query system is used to resolve the domain name. Recently, L.stoica proposed LISP (Locator Identifier Separation Protocol) [1] to improve the network security and scalability. In LISP, a query system must be set up to map the EID (Endpoint Identifier) to the RLOC (Routing Locators) and vice versa. Query system is also deployed in mobile management system. In a GSM system, if a terminal wants to call the others, the mobile management system should query HLR/VLR (Home Location Register/Visitor Location Register) to find out where the callee is.

To build a query system, we can employ centralized or distributed solution. In most cases, the data quantity in the networks is too large to be stored in a single database and the query delay will be too long if all the query requests are queuing at a single node, therefore the distributed solutions are preferred.

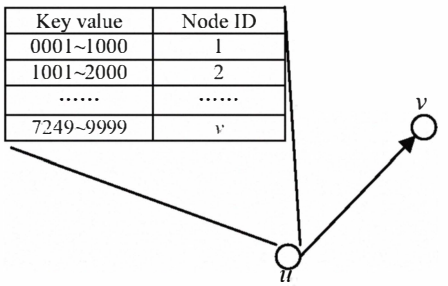


Figure 1 An example of mapping table

This work was partially supported by the 973 Program (2007CB307104), NSFC Funds (60972030, 60872032), Fundamental Research Funds for the Central Universities and Ph.D. Programs Foundation of Ministry of Education of China.

As to the distributed query system, DHT (Distributed Hash Table) is a general choice, e.g. Chord [1]. Chord has many advantages to be designed as a query system, such as the routing table entry size at every node is only $O(\log N)$, where N is the size of the identifier space. The small size of routing table greatly simplifies the system management. However, a query request on the Chord ring will be routed only depending on the destination identifier. It may incur query path to be suboptimal because part of the path may suffer from high delay. Another reason leading to the sub-optimization of the design is the ring structure of the query system. It is an unnecessary constraint which can be relaxed. The Chord is a self-healing ring. When a node fails or is deleted from the ring, the failed/deleted node will sent its data to its successor (the next node with the closest identifier). But Chord can't ensure the reliability of the query system during the ring is healing itself. On the other hand, if any node fails suddenly and there is not enough time to transfer data to its successor, the query system will never recover by itself. When we analyze the scenarios proposed in the 1st paragraph, no matter what scenario the mapping system is employed in, it is not necessary to be scalable, because the size of the query system (or the mapping system) is settled by the operators when the system is designed. The scalability of the system is realized by reserving more storage capacity on each node, so as to induct more terminals into the system. Hence, the query system can be designed without any consideration of scalability.

A paradigm to design query system considering no scalability and relaxing the topology constraint is distributed match making [2], which is created by Mullender. Two node sets $P(v)$ and $Q(v)$ are defined for each node v. $P(v)$ is the set of nodes where data replications of node v stored, while $Q(v)$ is the set of nodes which node v should lookup when a query is generated. For every node pair (v,v'), the intersection $|P(v)\cap Q(v')|=1$ will ensure the query from v' to v being responded correctly. The objective of distributed match making is to 1) minimize the average of $|P(v)+ Q(v')|$ taken over all pairs (v,v') and 2) find trade-offs between the lower bound of question 1) and the average number of elements (or worst case number of elements) in $S(v)$, where

$\boldsymbol{S}(v)=\{j: v \in \boldsymbol{P}(j)\}$. In the query system based on distributed match making, every query request can be accomplished within only one hop on the overlay, and it makes the query request be responded quickly. But there are also some drawbacks: 1) Only one hop paths are used in this paradigm, however the direct link on the overlay is not always the quickest path; 2) To make all the queries be able to be matched within only one hop, we need large storage capacity or amount of lookups for a single query; 3) It can not trade off between the used storage capacity and the response delay.

As a well-designed query system for the conditions explained above (in 1st paragraph), it should satisfy the ability below: a) responding quickly when a query request generated; b) correctly responding the query even if there are some failure nodes or links; c) we can easily trade off the query delay and the amount of data replications in the query system; d) relax all unnecessary constraints. As far as we know, there is no work to design a query system considering all the four elements.

A simple design considering no unnecessary constraint is shown in Fig.1. There is a mapping table at every node, which maps key values to the node storing the data related to required keys. Assuming a query about key value K generates at node u, node u will go to its mapping table to see where the key value K is stored (say it is node v) and then sends the request to node v directly. For example, when a query for key value 8000 is generated at node u, the node u will see its mapping table and find that the key value 8000 is stored at node v. Then it will send the query to node v directly. To improve the reliability of the system, each data can be replicated at more than 1 node, and then we lookup all the nodes containing our desired data simultaneously when a query is generated. Clearly, the objective of this framework above is to minimize the storage capacity being used by query system, while the constraints are the storage capacity and the required query system reliability. The query system may be improved when the network layer information, such as delay between each node pair and SRLG (Same Risk Link Group) information, is available. In this case, the total delay between all the node pairs can be optimized and reliability of the query system can be further improved by making the query paths between each node pair SRLG disjoint and node disjoint. However, these objectives (storage capacity and the response delay) can not be ensured to be minimized at the same time, a scheme should be proposed to trade off them. In this paper, we formulate ILP model to design such query system.

The rest of the paper is organized as follows. Section II introduces the framework of our query system and analyzes the query system design problem in detail. An ILP model is formulated in Section III and The numeric simulation and discussion presented in Section IV followed by conclusion in Section V.

II. PROBLEM ANANLYSIS

A. *Framework of the query system*

1) *Node structure*

In the query system, each node has 3 tables: key-value mapping table, key-node mapping table and routing table.

Key-value mapping table is used to obtain the value required by users. There are two parts in every key-value mapping table: primary key-value mapping table and back up mapping table. In primary key-value mapping table, all the data belong to the node itself (we call these data *primary data* for simplicity) are stored. All the replications of other nodes (*backing up data*) are stored in the back up mapping table. If a node stores all the primary data of one node, we say that this node is using 1 *storage*. Whenever a node revises its primary data, it will advertise all the nodes contain its replication to revise their backing up data.

Key-node mapping table is launched to find the destination of a given query. When a node receives a query, it will go to its key-node mapping table to lookup where the destination is. Every key value must be mapped to an exclusive node, so the key-node mapping table may be too large to be stored. In order to reduce key-value mapping table size, we store the value with k as its key on node $\left\lceil \frac{kN}{T} \right\rceil$, where N and T is the number of nodes and identifiers in the query system respectively. In this case, key-node mapping table size will be reduced to N.

After a node obtains which node is the query destination, it will see the routing table and set the route of all the queries for the same request. If query system will generate K queries for a single request in order to ensure system reliability, there will be K disjoint routes for each destination.

2) *Node function*

Any node in the query system has 3 functions, starting, relaying and ending of a query. When a node receives a *query request* (without any ambiguity, the query request sent by terminal will be called request hereafter) from terminal, a query is triggered in the query system. The node receiving request will execute its starting query function: 1) generates *query packets* (for simplicity, we say queries instead of query packets hereafter); 2) goes to the key-node mapping table to find the *query destination* (the node store the required value as its primary data); 3) obtain route of each query from routing table; 4) send the generated queries simultaneously. If a node receives a query and it is not very the *query sink* (the last node of a query's route. The required value of the query must be stored in the key-value mapping table of this node. The query with the sink node u will *terminated* at node u), it will relay the query through the route carried by the query. Otherwise, if it is the query sink, it will go through its key-value table to find the value associating with required key.

3) *Query process*

The query process of a given query is shown in Fig.2. We assume that terminal T wants to lookup the value associating with key 8000. A request will be sent to node u (node u is called the *local access node* of terminal T) by terminal T. When node u receives request from T, it generates a query (node u is call the *home node* of this query) immediately and then goes to key-node mapping table to see in which node the required value is stored. After the

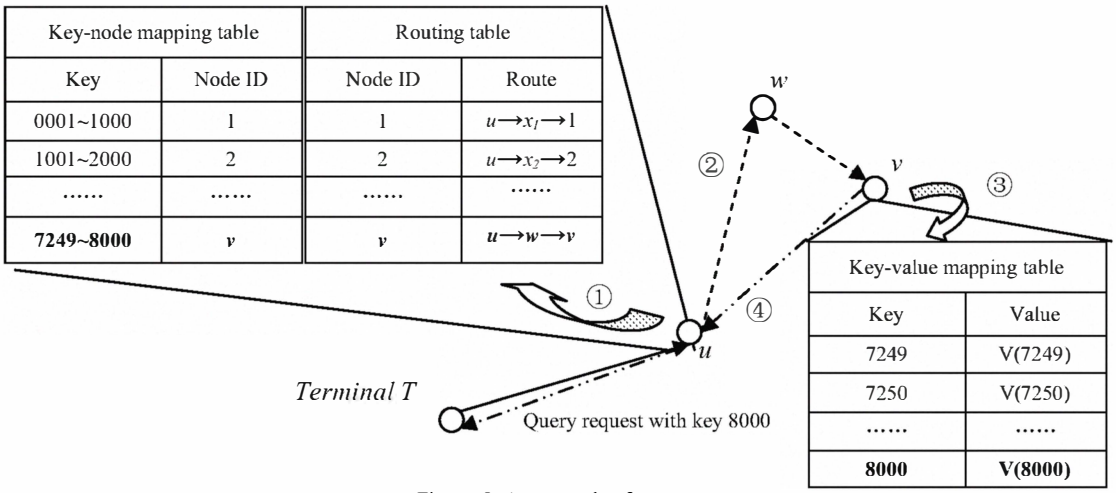


Figure 2 An example of query process

destination node v is obtained, node u goes through the routing table to find pre-computed route from node u to node v and writes it into the generated query (we simply say it is a query from home node to destination, even if query sink is different from destination. It is a query from node u to node v in this example). Then the query is delivered through its route $u \rightarrow w \rightarrow v$. When node v receives the query from u, it looks up its key-value mapping table and sends value associating with key 8000 back to node u. Finally, Node u responses terminal T with value queried from v. The duration from the query generated to home node receiving response from query sink is called *response time*.

B. Using Network Layer Information to Optimize Query System

Fig.3 shows virtual topology of a query system on the overlay. The delay cost of each virtual link is in the bracket. All the links will spend 10 units delay cost to send the query request from one node to the other one, except link (1,4) and link (4,5) spend only 4 units and 2 units delay respectively. For convenience, we assume that the delays on both directions of each link on the overlay are symmetric. If there is a query from node 1 to node 5, at least 2 paths exit to deliver it: $1 \rightarrow 5$ with 10 units delay and $1 \rightarrow 4 \rightarrow 5$ with 6 units delay. Obviously, path $1 \rightarrow 4 \rightarrow 5$ is preferred for its lower delay.

Another issue is that backing up data at different node can not only ensure the reliability of the query system, but also quicken the response to a query request. See Fig.3 again, if node 4 stores all the primary data of node 3 (without ambiguity, we say data of node u instead of primary data of node u), when there is a query from node 1 to node 3, this query can be sent to node 4 with only 2 units delay cost.

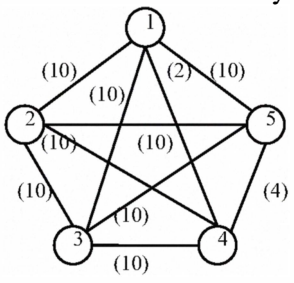


Figure 3 Overlay structure of a query system

Intuitively, every node keeps the replication of all other nodes can minimize delay cost. But it is not reasonable for the storage constraint. Backing up data on different node will spend more storage which increases the cost to set up a query system. To quantify the optimization of a query system, we formulate a *query system cost* given in the following:

$$\begin{aligned}\text{Query System Cost} &= \text{storage cost} + \text{delay cost} \\ &= \gamma \times \text{number of storage} \\ &\quad + \text{delay cost} \end{aligned} \tag{1}$$

The cost ratio γ determines the relative importance between the storage cost and the delay cost. In Fig.3, we set the γ to be 5. If we deliver all the query requests directly and every node stores its primary data, the query system cost will be 197 (25 storage cost and 172 delay cost). If the delivering path for the query from 1 to 5 and 5 to 1 are set to be $1 \rightarrow 4 \rightarrow 5$ and $5 \rightarrow 4 \rightarrow 1$ respectively, the query cost will be reduced to 189 (25 storage cost and 164 delay cost). If we back up node 3 on node 4, the minimal query system cost will be reduced to 156 (30 storage cost and 126 delay cost).

The other benefit of backing up data on multiple nodes is that when failure occurs on the home nodes (i.e. node 3 fail in this example), we can still get the response from other backing up nodes which contain its replication (i.e. node 4 in this example). It is a good characteristic to improve the reliability of the query system. In order to ensure the reliability of the query system, more than one query for the same request is needed. When queries are generated, they will be delivered through disjoint paths in parallel (We say all the disjoint paths are the *group* of paths for these queries). If only there is one query can be delivered to its destination, the request can be responded correctly. If there are k requires for the same request delivered through disjoint path, the query could still be responded correctly even there are k-1 failures in the network.

Due to more than one virtual link on the overlay may traverse the same links on the physical topology, the common link's failure will lead all the links on the overlay traversing it to collapse. All the links on the overlay traversing common physical links will be put into the same SRLG, and all the queries generated for the same request should be delivered by SRLG disjoint path. For the same reason, paths used to deliver the queries generated for the same request should be node

disjoint.

In the next section, we will formulate an ILP model to design query system, in which the network layer information is used to further optimize the query system.

III. ILP FORMULATION

A. ILP with Information about Network Layer

Notation List

- N The number of nodes in the network
- K The number of paths should be prepared for a pair of query
- γ Predefined cost ratio of a storage to a unit delay cost
- ε Predefined some number, which used to minimize the number of virtual links in the query system
- C The maximize storage of every node
- M The number of SRLG on the application layer virtual topology
- l_{ij} The delay cost of link (i, j) on the application layer virtual topology
- e_{uv}^{m} Binary constant. It takes 1 if link (u,v) belongs to the m^{th} SRLG, and 0 otherwise.
- p_{ijk}^{u} Binary variable. It takes 1 if the k^{th} query path from node i to node j traverse node u, and 0 otherwise
- f_{uv}^{ijk} Binary variable. It takes 1 if the k^{th} query path from node i to node j traverse link (u,v), and 0 otherwise
- d_{uv} Binary variable. It takes 1 if the link (u,v) is used to propagate query request, and 0 otherwise.
- x_{ij}^{uk} Binary variable. It takes 1 if the destination of the k^{th} path from node i to node j is node u, and 0 otherwise.
- s_{ij} Binary variable. It takes 1 if node j abide itself on node i, and 0 otherwise.
- r_{ijk}^{m} Binary variable. It takes 1 if the k^{th} path from node i to node j traverse at least one link belongs to the m^{th} SRLG, and 0 otherwise.

B. ILP Formulation with Information about Network Layer

Given query system size N, the delay cost of each virtual link on the overlay and the cost ratio γ of the unit storage cost to the unit delay cost, the ILP formulated in (2)-(16) can generate an optimal query system with minimal query system cost in (1). See (2)-(16) at the bottom of this page.

Objective (2) aims at minimizing the query system cost in (1), the first term is the total delay cost of the query system, the middle one is the storage cost and the last term is added to reduce the number of virtual links constructed in the system, which can reduce the cost in the real engineering. Constraint (3) means that if there are queries with destination node j terminating at node k, the replication of node j 's primary data must be stored at node k as backing up data. Constraints (4)-(5) formulate the flow conservation property for every path. Constraint (6) indicates that the k^{th} path prepared for the query from node i to node j traverses link (u,v) and the link (u,v) belongs to the m^{th} SRLG, then the path will use the SRLG. Constraint (7) says that for each SRLG, there is at most one path in the group of paths for the queries from node i to node j traversing the links in this SRLG. This constraint ensures the SRLG disjoint of the paths prepared for the query between any node pair. If only there is one path traverse link (u,v), an application layer virtual link from node u to node v must be set up. This constraint is shown in (8). Constraint (9) allows that each node can store C replications at most. Constraint (10) specifies that if the required data is on the local node, no virtual link will be used on the overlay. Constraint (11) stipulates that every query path has one sink. Constraint (12) identifies that for a group of paths for a query between a given

---

Objective:

Minimize
$$\sum_{i}\sum_{j}\sum_{u}\sum_{v}\sum_{k} f_{uv}^{ijk} \bullet l_{uv} + \gamma \sum_{i}\sum_{j} s_{ij} + \varepsilon \sum_{i}\sum_{j} d_{ij} \quad (2)$$

Subject to:

$$\sum_{i} x_{ij}^{uk} \le N \bullet s_{uj} \quad \forall u, j, k \quad (3)$$

$$\sum_{u} f_{uv}^{ijk} - \sum_{u \ne i} f_{vu}^{ijk} = x_{ij}^{vk} \quad \forall i, j \ne i, v \ne i, k \quad (4)$$

$$\sum_{u} f_{iu}^{ijk} - \sum_{u \ne i} f_{ui}^{ijk} = 1 \quad \forall i, j \ne i, k \quad (5)$$

$$\frac{e_{uv}^{m} + f_{uv}^{ijk} - 1}{2} \le r_{ijk}^{m} \quad \forall i, j, k, u, v, m \quad (6)$$

$$\sum_{k} r_{ijk}^{m} \le 1 \quad \forall i, j, m \quad (7)$$

$$d_{uv} \ge f_{uv}^{ijk} \quad \forall i, j, u, v, k \quad (8)$$

$$\sum_{j} s_{ij} \le C \quad \forall i \quad (9)$$

$$f_{ij}^{uuk} = 0 \quad \forall i, j, u, k \quad (10)$$

$$\sum_{u} x_{ij}^{uk} = 1 \quad \forall i, j, k \quad (11)$$

$$\sum_{k} x_{ij}^{uk} \le 1 \quad \forall i, j \ne i, u \quad (12)$$

$$s_{ii} = 1 \quad \forall i \quad (13)$$

$$x_{ii}^{uk} = 0 \quad \forall i, u \ne i, k \quad (14)$$

$$p_{ijk}^{v} \ge \sum_{u} f_{uv}^{ijk} \quad \forall i, j \ne i, v \ne i, k \quad (15)$$

$$\sum_{k} p_{ijk}^{v} \le 1 \quad \forall i, j \ne i, v \quad (16)$$

node pair, each node can be selected as the sink of at most one path. Constraint (13) ensures that every node must store its primary data. Constraint (14) forbids every node query its primary data from other node. If a path traverses the link with the end node i, the node is on this path. This rule is formulated by (15). The last constraint (16) ensures the paths designed for query from node i to j are node disjoint.

It should be declared that if the variable $f_{ii}^{ijk}=1$ means node i has backed up the data of node j. So the k^{th} path for query from i to j will be terminated at node i. We say this query is passing through the link (i,i) in our ILP model.

There are $KN^4+KN^3+(KM+2-K)N^2+(K-1)N$ variables and $(K+KN)N^4+(5K+2)N^3+(4K-2)N^2+(3K-1)N$ constraints in the model formulated above.

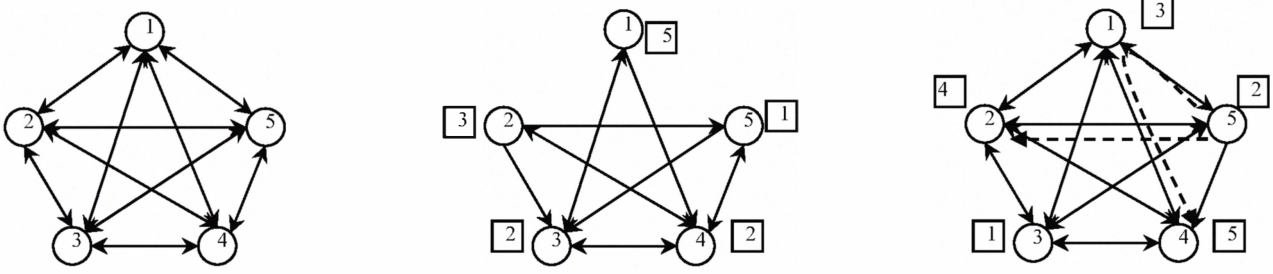
This ILP model above is flexible. For example, when the information about SRLG is unavailable, we can remove the constraint (6) and constraint (7). If we want to prepare k disjoint paths for query whose home node and destination are the same node, we can remove the constraint (10) and constraint (14), and change the constraint (12) to be

$$\sum_{k} x_{ij}^{uk} \le 1 \quad \forall\, i,j,u \tag{17}$$

IV. Simulation and Analysis

We use ILOG CPLEX 10.0 to implement the ILP on a computer with Duo-Core 1.86 GHz Intel CPU.

Fig.4 (a)-(c) show the results of ILP in 5 nodes query systems with different parameters, the data replication stored at each node is shown in the rectangle beside it. In all the query systems, we assume that every node has 2 storage capacities. It is required that each query should be responded from only 1 node in the first two query systems while be responded from at least 2 nodes for each query in the last query system, i.e., the last query system should be reliable when there is a single failure in the query system.



(a) delay cost between any 2 nodes is 1, $\gamma=2$

(b) delay cost between any 2 nodes is 1, $\gamma=0.5$

(c) delay cost between any 2 nodes is 1, except delay on (4,5) is 100, $\gamma=2$

Figure 4 The result of ILP in 5 nodes networks

In case (a), using the direct link between any node pair will pay 1 unit delay cost while pay 2 units storage cost for a back up. ILP mode designs a full mesh query system without using any extra storage capacity. The solution use 20 delay cost with 10 storage cost.

In case (b), the only parameter differing from case (a) is the cost ratio, only 0.5 units storage cost should be paid for backing up a node. The result of the ILP uses all the residual storage capacity to reduce the query system cost. At the same time, the virtual link (1, 2) and (1, 5) need not be set up any more, because there will be no path using them. In this case, the solution uses 10 storage capacities with 5 storage cost and 15 delay cost. The total cost is reduced because of the reduction of the cost ratio.

In case (c), we set the delay cost on link (5,4) to be 100 and link (5,2) and (3,4) to be in the same SRLG. We hope the query system is still available even if a failure occurs in the network. Fig.4 (c) shows the solution of our ILP models. Each node is backed up on one another node and link (5, 4) is not used for its large delay cost. Two query paths from node 5 to node 4 are 5→2 and 5→1→4, which are shown as dotted line. Path 5→3→4 can not be used due to the SRLG constraint. This query system spends 36 delay cost and 20 storage cost.

Though there are 5 nodes in all of the three networks, none of them have same virtual topology. It comes over the topology constraint, hence our design will obtain more optimized solution than the framework constrained to the topology such as Chord and LISP-tree [3].

The framework we used to design the query system can easily trade off the delay cost and the storage cost through adjusting the cost ratio. From the case (a) and case (b), we can see that the lower cost ratio is, the more storage we will use to reduce the query system cost.

V. Conclusion

Static query system is needed in DNS and mobile management systems. But there has been no work on designing such a static query system. In this paper, we propose a new framework to design static query system and formulate an ILP model to design it, which uses network layer information to optimize query system. From simulation, we demonstrate that our method relax topology constraint of query system and realize trade off between delay cost and storage cost.

References

[1] L. Stoica, R. Morris, D. Darger M. F. Kaashoek and H. Balakrishman, "Chord: A Scalable Peer-to Peer Lookup Service for Internet Applicaiton," *SIGCOMM 2001*

[2] Sape J. Mullender and Paul M.B. Vitanyi, "Distributed Match-Making," *Algorithm*, 1988, vol. III, pp. 367–391.

[3] Jakab L., Cabellos-Aparicio A. Coras F., Saucez D. and Bonaventure O. "LISP-TREE: A DNS Hierarchy to Support the LISP Mapping System," *IEEE JSAC* 2010, Vol 28, pp. 1332.