E-Chord: Keyword-Based Search Algorithm Based on DHT in Mediation Architecture

QASEM KHARMA[1],RAIMUND EGE[2],SAID ABU SHAAR[3]

[1],[3]Al-Ahliyya Amman University
Faculty of Information Technology
Amman, Jordan
[1]Qasem.Kharma@gmail.com,[3]said@ammanu.edu.jo
[2]Northern Illinois University
Department of Computer Science
DeKalb, IL 60115 USA
[2]ege@cs.niu.edu

Abstract. E-Chord is a Distributed Hash Table algorithm (DHT) inspired from Chord, Pastry and CAN algorithms. It meant to provide a keyword-based search in the Three-Layer Mediation Architecture. The E-Chord is deployed in the middle layer (Integration Layer) of the architecture and provides services to the higher layer (Presence Layer) and the lower layer (Homogenization Layer).

Keywords: DHT, Mediation, Heterogeneous Data Integration

(Received June 25, 2006 / Accepted November 01, 2006)

1 Introduction

In the last few decades, the dependency on accessing data from distributed data sources has increased because of the distributed nature of data and breakthroughs in communication. Existing enterprises were merged. New international businesses are established. Governmental organizations need to share information. Advancements in communication, such as wireless, cable, satellite, and fast Internet access make accessing heterogeneous data viable. Moreover, the Internet, which is a network of networks of computers, provides rich data sources. Therefore, sharing data is essential nowadays. Unfortunately, in most cases each individual data source has its own data structures, platform, and design. As a result, the integration of distributed heterogeneous data sources is not a simple task. Huge investments were made in each of those data management systems; hence, the decision to build a new integrated data management system or enforce changes is often not a practical solution.

A mediation architecture [25] was proposed as a solution to integrate heterogeneous data sources in a specific domain of knowledge by adding a layer between the application layer and the data sources in the system. A mediation layer which handles the responsibilities of accessing the heterogeneous data sources and presents the integrated data is placed between the data sources and the application layers; therefore, no changes need to be done in the layer of the data sources. A client can query the system by exploring the schema generated by the mediation system. The data schema of a specific source is called a local schema; the mediation schema is called a global schema, and it integrates and transforms several local schemata. Although this solution is viable and cost-effective, it is not very reliable since it maintains either a global schema in a central unit or a specific-domain schema in each mediator. The existence of such a central unit makes the system vulnerable to failure.

Our mediation architecture [8, 10] adopts a distribution technique from Peer-to-Peer (P2P) architectures called Distributed Hash Table (DHT) algorithm in order

to avoid having a central failure unit in the system. The proposed algorithm is a Chord-like algorithm. We will refer to our enhanced skip-list algorithm as E-Chord which is deployed in the Integration layer.

2 Background

The background section covers two things: the Three-Layer Mediation Architecture and DHT algorithms.

2.1 Three-Layer Mediation Architecture

The Three-Layer Mediation architecture [8, 9, 10, 14, 15, 27, 26] which was designed by the Secure System Architecture (SSA) laboratory at Florida International University (FIU) was a group research project directed by Dr. Ege. Members. The architecture is to define and build a multi-layered mediator-based multimedia architecture that provides a dynamic, scalable framework for telecommunications software environments. It is capable of handling complex data types and providing services to various devices including mobile devices. The architecture (see Figure 1) is based on three layers: a "presence" layer takes requests from clients and is responsible for caching and buffering of streams that it receives from the "integration" and "homogenization" layers. The second layer is the "integration" layer which is responsible for decomposing requests, searching for the sources, integration. The "integration" layer is composed of a set of mediators called "composers". For each new session created by a "presence" mediator, a composer is elected to be a "global mediator" which is responsible for communication with the presence mediator and managing the request processing. The third layer is the "homogenization" where a connection to actual data sources is established. The "homogenization" layer is composed of a set of connectors. On top of each data source, a connector is placed to manage the access to the data source. The common data model in the architecture is based on XML. XML is a semi-structured model that is capable of handling structured and unstructured data. XML request and its decomposition is done at the "integration" layer which consists of mediators that successively decompose an XML request into smaller XML requests that are closer to the data sources that are served-up by the "homogenization" layer.

2.2 DHT algorithms

In the decentralized P2P systems where there is no centralized unit and all nodes (peers) have the same computation power, many problems arise such as security, scalability, administration, and more. Locating data files in the distributed P2P environment is essential since many systems are naturally distributed. The most difficult challenge in P2P is how data can be found in a large, scalable P2P system without relying on a central server [1]. If this server fails, the system will fail. To avoid having a central failure scenario, many algorithms based on DHT were proposed in the past few years. Instead of having a central server, those DHT algorithms use a DHT in which each node maintains some knowledge about some other nodes (but not all). The general purpose of these algorithms is to map a value onto a key using a hash function. Although the general format of the value is a node IP address, the value can be any meaningful value for the system to be built such as document name.

Balakrishnan et al. [1] classifies DHT algorithms into three categories:

1. Skiplist-like routing algorithm: The Chord algorithm [23, 24] is an example of skiplist-like routing algorithm. In Chord, the hash function assigns a m-bit (where m is the number of the bits used for storing the key in binary) identification key using Secure Hash Algorithm (SHA-1)[17] as a base function to map an IP address onto a key. The nodes in the system are arranged in an identifier circle. Each node on this circle maintains a finger table containing the IP addresses of $n + 2^{i-1}$ successors where n is the node ID and $1 \leq i \leq m$. In other words, this finger table maintains the IP addresses of halfway, quarter-of-the-way, eighth-of-the-way, and so forth.

2. Routing in multiple dimensions: The scalable Content Addressable Network (CAN) [18, 19] is an example of routing in multiple dimensions. Each node in CAN maintains a chunk of the DHT called zone. These zones are distributed in d-dimensions. In addition to storing a chunk of the DHT in the zone, each zone maintains information about its neighbors in the d-dimensions.

3. Tree-like algorithms: Tree-like algorithms, such as Pastry [5, 21], Tapestry [13], and Kademlia [16], use a structured prefix to maintain the location of nodes. Each node maintains IP addresses of some other nodes in its leaf. For instance, Kademlia algorithm assigns 160-bit IDs to the nodes in the P2P system and treats those nodes as leaves in a binary tree.

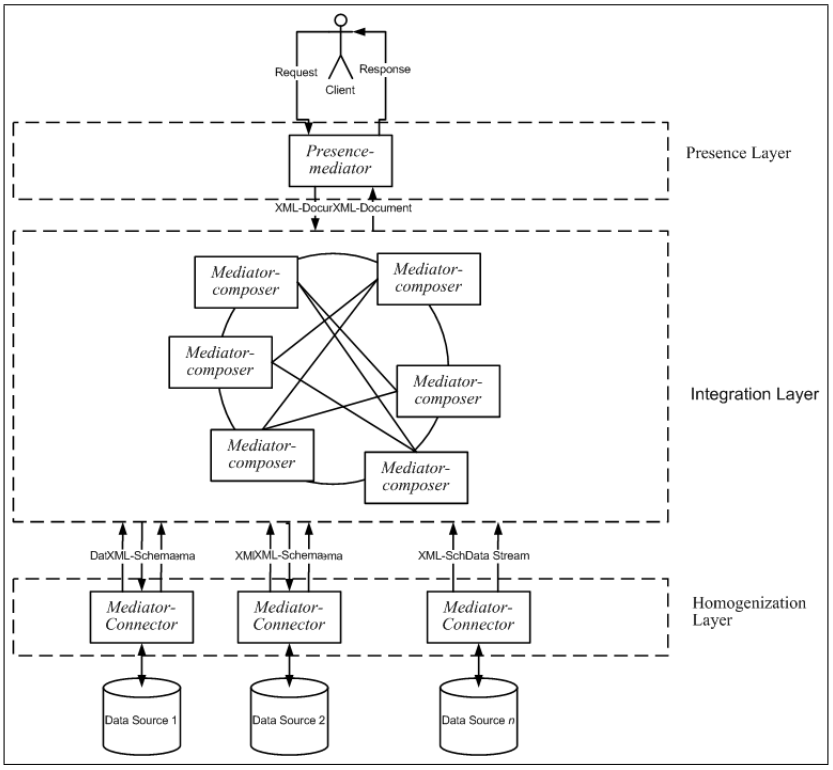

Figure 1: The Three-Layer Mediation Architecture

3 Overview of Searching in the Three-Layer Mediation Architecture

The design goal of our mediation architecture is to avoid having any component that constitutes a central point of failure. Unlike MIX [2, 3] and Garlic [4, 20], our system does not maintain a global schema, which is a global view of the integrated data sources in the system, in a central repository. Although TSIMMIS [6, 12] uses a distribution strategy for its schema over specific domain chain of mediators, each mediator maintains its own global schema. [8, 10] adopts a distribution technique from Peer-to-Peer (P2P) architectures called Distributed Hash Table (DHT) algorithm using Enhanced Chord-like (E-Chord).

Although the mediator-composers are connected in a P2P fashion, the mediation system has different characteristics from the standard P2P systems. First, mediation systems are usually domain specific systems. They are deployed in a specific knowledge domain to provide decision-makers with information in that domain which may be integrated from several sub-domains. Each subdomain may be composed of several heterogeneous data sources. Some keywords, which are key search criteria, may be repeated. For instance, if the system is deployed in a medical domain, the system might be composed of data sources that contain medical records, and insurance information. In this scenario keywords, such as patient, name, and SSN, will be frequently repeated in queries. Second, the mediator-composers are more stable than standard peers in a P2P system. P2P systems were originally intended for music files sharing over the Internet. They created networks in which peers join and leave arbitrarily. Mediator-composers are more stable since they could be run by either the data source administrators or service providers.

The key operation in DHT algorithm is the “lookup”. DHT algorithms are structured in the sense that each node in the system is responsible for a range. Before any action can be taken, the node which is maintaining the range of the desired action must be found. For instance, when a new node wants to join the system, its successor must first be looked up. Then, the new node can join the system.

The E-Chord algorithm is a relaxed version of the Chord algorithm. It combines features from Chord [23, 24], CAN [18, 19], and Pastry [5, 21]. The general structure is based on the Chord structure. Like Pastry, the adapted algorithm updates its routing information when a node is discovered not to be available. A frequency list is added to each node instead of a neigh-

borhood set. Finally, like CAN, the data sources are responsible for their pointers.

4 Routing Information

Like DHTs, the E-Chord algorithm uses the Secure Hash Algorithm (SHA-1) as a hash function. The SHA-1 generates nodes' identifiers and keywords' identifiers. The node identifiers are generated by running the SHA-1 on a composition of the IP address and the port number of each node while the keyword identifiers are generated by running SHA-1 on the keyword to be indexed in the system. The keywords' identifiers are distributed over the first successor of each keyword. As a result of using SHA-1, the generated identifiers have 160 bits, and the input is limited to 2^{64} bits.

The E-Chord algorithm maintains three sets of routing information in each node. The first set, like Chord, is the finger table which contains 160 entries. The entries are points to halfway, quarter-of-the-way, eighth-of-the-way, and so forth. The entries in the finger table can be found by first calculating the estimated value from the formula for $1 \leq i \leq 160$ $estimated\_value = (current\_node\_identifier + 2^{i-1}) \ mod \ 2^{160}$ and then finding the first actually successor of the estimated value. The second set, like Pastry's leaf set, maintains the immediate successors and predecessors. The third set maintains entries of the most frequent used keywords. The connector on the top of the data source maintains counters of the number of times it was queried for its keywords. When it sends the schema to the composer, the frequency of each keyword in the schema is sent along with the schema. Although the connector is responsible for counting the frequency, the entries in the frequency set points to the composer which is responsible for the range of the keyword, not to the connectors. The composer checks the frequencies assigned to the schema with the ones stored in its set and adds any keywords that have higher frequencies than the existing frequencies in its set.

When a global mediator receives a request from a presence mediator, it adds to each element in the request an identifier attribute which contains the hashed value of the keyword. Then, the elements are sorted in the request according to the identifiers. After that, the composer, either the global mediator or a cooperating composer mediator, searches its sets for each keyword in the following order:

1. The composer checks first if the keyword is within its range. If the keyword is within its range, it adds the connector_id to the request and returns it to the global mediator.

2. The most frequently used keywords set: the composer checks the set for an exact match. If an exact match is found, the composer forwards the request to the composer responsible for the range of that keyword.

3. The composer checks the set of the immediate successors and predecessors. If the keyword is within the range of the largest node identifier in the successor list and the composer identifier, it means that the keyword must be maintained by one of the successors. In a similar approach, the composer checks whether the keyword is within the range of its predecessor list. If the keyword is within the successor/predecessor list, the composer forwards the request to the composer that maintains the range of the keyword.

4. Finally, if non of the previous steps finds the composer that maintains the requested range, the composer forwards the request to the first node with an identifier that immediately precedes the keyword's identifier in the finger table.

Before the global mediator looks up the next keyword, it checks whether the next keyword identifier is within the range of the composer maintaining the previous keyword. The global mediator groups the set of the keywords within the same range and forwards them in one request to the composer. The composer adds the connector_ids to the request and returns it to the global mediator.

The E-Chord is a relaxed version of Chord; it has a similar structure to the Chord algorithm. It is a skip-list like algorithm. The finger table and the frequency set help the composer to skip as many as possible composers that cannot help in solving the request. The successor and predecessor set compose the logical ring of the system; therefore, the system has a ring geometry. Since the system is using the SHA-1 algorithm, the composers in the system are distributed over the logical ring from range 0 to $2^{160} - 1$.

The size of the finger table is fixed while the other two sets are runtime parameters. The size of the finger tables is fixed to 160 entries in each node because it is related to the identifier size. The SHA-1 generates identifiers of length 160 bits, so the maximum number of entries that can be maintained in the finger table is 160. The size of the successor and predecessor set and the frequency set are determined by the system administrator. The size of successor and predecessor set should be a reasonable size according to the expected number of the composers in the system. In the Chord algorithm the successor list is recommended to be of size $2log_2n$

where n is the expected number of the nodes in the system. The size of the frequency set should be reasonable according to the expected number of the keywords in the system.

5 Joining the System

There are two cases that affect the DHT in the integration layer. The first case is when a new composer joins the system. The second case is when a new connector joins the system. The former case affects the system by adding a new composer to the P2P system; as a consequence, this new composer will play a role in routing and maintaining routing information while the later case, adding a new connector, adds new indexes to the DHT tables. The connectors in the system do not play any role in routing requests.

When a new composer wants to join the system, it must first find another composer which is already in the P2P system. There are many bootstrapping techniques [11] such as simple broadcast, selective broadcast, and adaptive broadcast. In the simple broadcast, the new peer sends a message to every peer in the system. This technique overloads the system with messages; as a result, it consumes the bandwidth of the system. Unlike simple broadcast, in the selective broadcast the new peer sends a message to selective peers in the system based on predefined criteria such as trust relationship in the selective broadcast technique. Adaptive broadcasting is similar to the selective one in the sense that both of them try to minimize the consumption of the network resources. However, adaptive broadcasting needs to keep monitoring the system for changes. The CAN algorithm [18, 19] uses a selective techniques in which a list of nodes are maintained in a registered domain. When a new node wants to join the system, it retrieves the list and sends a message to a node in the list. In our system, we opt to enter an existing node IP address as a parameter when creating a new composer.

Once the new node finds a composer in the system, it sends a "join" message to that node. The node treats the "join" message as a "lookup". It searches for the node responsible for the range of the new node in a similar fashion to finding a keyword (See Section 4). Unlike searching for keyword, the process will terminate once the composer maintaining the range of the new composer is found, not the connector. After finding the composer, the new composer will be the immediate predecessor of that node.

The next step is to initialize the lists. The most important list for the system is the successor/predecessor list. The importance of this list is to keep the system's logical ring connected all the time; so that, the routing to the destination can be guaranteed. Once the new node finds its position in the ring, it obtains the successor/predecessor list from the composer which is responsible for the range of the new node. The new composer will have exactly the same predecessor list as the old one. The successor list is almost the same except that the first successor is the old composer. The new composer informs the composer in the successor/predecessor list of its arrival. If the size of successor/predecessor list is $2r$ where r is a parameter defined by the system administrator, the new node will sends r messages to its successors to update their predecessor lists and r messages to its predecessors to update their successor lists. The nodes in the successor/predecessor list send their frequency list, so the new node can construct its frequency list from those lists. Finally the finger table is initialized from the finger table of the predecessor. Unlike Chord, our algorithm does not lookup all the entries in the new finger table. The new composer obtains a copy of the finger table of its predecessor. Then, the new composer checks whether the ranges of the entries are within the current pointer values. The pseudo code for initializing the finger table is listed in Figure 2. The new composer notifies the nodes which are expected to point to it in their finger table using Chord's "update_others" method [23]. The idea of the "update_others" is to find the nodes that precede the new composer by 2^{i-1}. That can be done by finding the predecessor of $n - 2^{i-1}$, and then update the i entry in that node to point to the new composer if the "i" entry must point to the new composer.

```
for i←1 to m do
    finger[i].start ← node_identifier+2^(i−1) mod 2^160
    if finger[i].start > finger[i].node
      finger[i].node=find_successor(finger[i].start)
    %else no action is needed
```

Figure 2: Pseudocode for initializing the finger table

When a new connector joins the system, it will find an existing composer in the ring of the system. An existing composer can be found using the bootstrap techniques explained earlier in this section. Unlike adding a new composer, adding a new connector will not affect the routing information, but it adds new indexes to the system. Once the connector finds a composer, it sends its XML document. The composer called distributor generates identifiers for the keywords in the XML document. Then, it sorts the keywords according to their identifiers. After that, the distributor finds the composer which is responsible for the range of the keyword. This

process is similar to finding a keyword, but instead of returning a connector it adds the keyword. The composer which will index the keyword returns its range for the distributor. The distributor groups the keywords with the composer range and sends them in one message. The distributor repeats this process until all the keywords are distributed.

Unlike CFS [7] which is a file storage for blocks based on Chord and PAST [22] which is a file storage for files based on Pastry, the mediator does not distribute the data in the data sources among the composer-mediator. The mediator system needs only to distribute pointers to the connectors on top of data sources which will be accessed through connectors. In other words, the composers only cache the identifiers and the connectors' IP addresses and socket numbers, and the data can be accessed and retrieved directly from the source through the connector.

6 Replication and System Recovery

The system has a self-recovery mechanism when a composer fails. In order to preserve the indexing from being lost, the system maintains a replication of each composer's indexes in its successors which are in its successor list. Therefore, the system administrator needs to consider a reasonable size for the successor list. Assume that the system administrator chooses size l for the successor list, then the probability that all the successors fail is $1/2^l$ since the probability of a successor failing is independent from the others. Not only does composer failure affect the keyword indexing, but also it affects the routing in the system. Each node in the DHT algorithm maintains information about a set of other nodes in the system. When a node in that set fails, the other nodes assist in routing and recovering the set.

In most cases, when a composer leaves the system, it will not notify the others. As a consequence, the routing information maintained in the finger table, successor/predecessor list, and frequency list may not be valid. However, the system is not aggressive in maintaining all the pointers valid. The system will only keep the successor list having valid pointers by sending messages periodically to its immediate successor to check its availability. If the immediate successor does not respond, the composer will contact the next successor in the list. Then, the successor list will be updated by receiving the successor list from the first composer that responds to the message.

If an entry in the frequency list was found not to be valid, the composer removes this entry from the list. The frequency list is meant to help the composer to find a short-cut to the destination composer based on historical requests. However, since for each request the global mediator is elected, the path from the global mediator to the connector is built dynamically. The connectors in the system maintain counters of the frequencies of each keyword in its source. The frequency counters are returned with the schema to the composer, so the composer will maintain the most frequently used keywords from its frequency list and the received schema. The frequency list is built dynamically and may differ from a composer to another.

The last case is an invalid entry in a finger table. The system will lookup the successor of the start interval of the failed entry. The start of the interval is calculated by the equation:
$finger[i].start = (composer\_identifier + 2^{i-1}) \bmod 2^{160}$. Then, the node looks up this value by invoking "lookup(finger[i].start)". The "lookup" method returns the identifier of the node which maintains the range of the requested interval.

The system needs to have at least one valid pointer in each composer. If all the pointers in the finger table fail, the composer can forward the request to its successor using the successor list. Even if the request is not within the successor range, eventually the destination will be reached. If all the finger tables in the system fail, the lookup can be done in $O(n)$ messages by forwarding the messages from a composer to its successor. However, having the finger table can reduce the number of messages to $O(\log_2 n)$ by skipping half the distance closer to the destination each time.

7 E-Chord Simulation

A Java based simulation of E-Chord was developed. The simulation creates logical nodes instead of physical nodes. The logical node simulation is a simulation in which the system runs on a single machine instead of set of computers. Logical node's information can be either randomly generated or loaded from a file. The generated or loaded information simulates the physical nodes' IP address and port numbers. Figure 3 shows the menu of the simulation. The user can either choose to load nodes from a text file or to generate IP address and ports randomly. The maximum number of nodes that can be loaded depends on the simulation machine configuration (hard disk capacity, memory, softwareĚ). The simulation was run on a machine with 2.0 GB memory. 10,000 nodes were randomly generated (Figure 4). Increasing the number or running applications in addition to the simulation on the same machine may give a running error because of the memory limitation.

After loading the nodes' information the "join" option, which is choice number 2 in the menu, must be

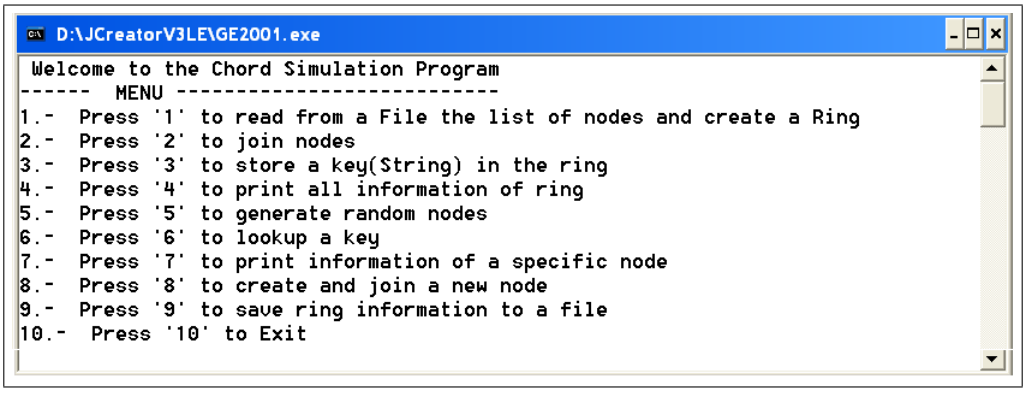

Figure 3: The main menu of the simulation

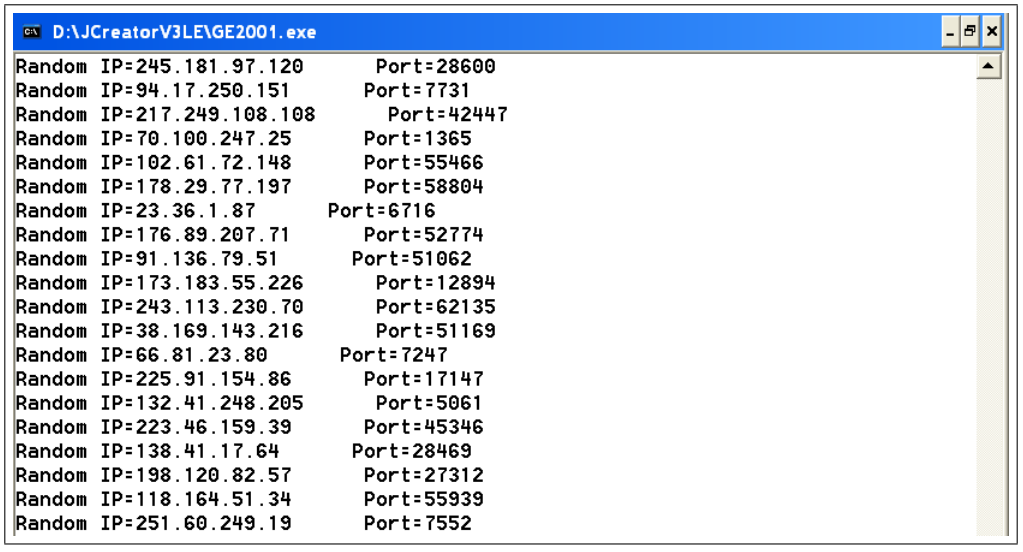

Figure 4: Random generation of nodes

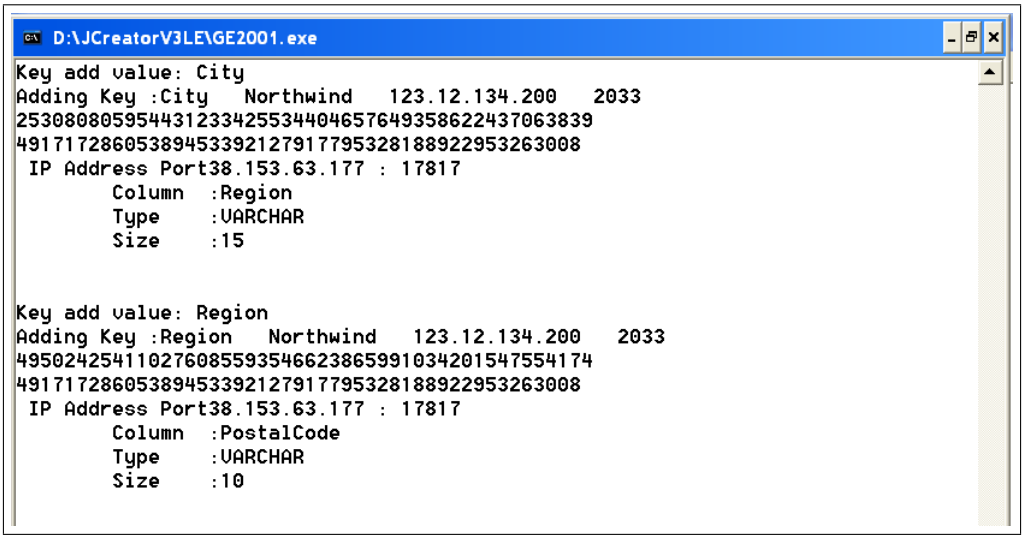

Figure 5: Adding keywords to the system

chosen to fix the finger tables. E-Chord after running the "join" option is in a stable state that all the lists maintain valid information. By adding a new logical node, which is option 8 in the menu, the system state will change to unstable state which will eventually be fixed by serving request.

Keywords can be added to the system by choosing option 3 from the menu which will load and convert a data source into an XML structure. This option currently supports only MS ACCESS data sources; however, this will not affect the fact that the system is capable of handling heterogeneous data source since the input to the E-Chord is always XML structures. Loading keywords option uses JDOM library (www.jdom.org) to generate XML tree from the data source. Figure 5 shows some keywords added to the system. It illustrates the key value to be added, its connector information, the identifier of the key, the identifier of the composer maintaining the range, and the composer information.

8 Summary

We introduce in this research an enhanced skip-list algorithm which is based on DHT to cache the keywords in the mediation architecture. The algorithm is a relaxed-version of Chord algorithm called E-Chord which uses features from CAN and Pastry algorithms. A frequency list and successor / predecessor list are added to each node (composer) to enhance the routing information. Unlike the Chord algorithm which maintains only successor list, the proposed algorithm maintains successor/predecessor list. Moreover, each node maintains a frequency list which is composed of a list of most queried keywords. A new initializing finger table method was designed to minimize the number of message in construction the finger table of a newly joined node. Using E-Chord in mediation is effective when data sources are disjoined.

Although the new algorithm needs less message in maintaining the system and routing, it needs $O(log_2 n)$ messages to find each keyword. If a composer wants to look up a keyword which does not exist within its range or the frequency list, the composer uses the finger table to route the request. Therefore, it jumps half the distance closer to the target, like Chord. Unlike Chord, the enhanced algorithm uses relaxed repairing mechanism for the finger table entries instead of periodical checks. Our algorithm is not aggressive in maintaining the finger table. An action is taken if an entry in finger table is found to be invalid.

The most expensive operation from the time complexity viewpoint is the sorting. The global mediator performs sorting twice: keyword identifiers sorting and connector_id sorting. It takes $O(e\ log_2\ e)$ to sort "e" elements in the XML document using merge sort or binary sorting algorithm. The selection algorithm is not as expensive as the sorting. The selection can be done in $O(l)$ where "l" is the size of the lists in the composer.

References

[1] Balakrishnan, H., Kaashoek, M. F., Karger, D., Morris, R., and Stoica, I. Looking up data in P2P systems. *Communications of the ACM*, 46(2), Feb. 2003.

[2] Baru, C., Chu, V., Gupta, A., Ludäscher, B., Marciano, R., Papakonstantinou, Y., and Velikhov, P. XML-based information mediation for digital libraries. In *DL '99: Proceedings of the fourth ACM conference on Digital libraries*, pages 214–215, Berkeley, California, Aug. 1999. ACM Press.

[3] Baru, C., Gupta, A., Ludäscher, B., Marciano, R., Papakonstantinou, Y., Velikhov, P., and Chu, V. XML-based information mediation with mix. In *SIGMOD '99: Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, pages 597–599, New York, 1999. ACM Press.

[4] Carey, M. J., Haas, L. M., Schwarz, P. M., Arya, M., Cody, W. F., Fagin, R., Flickner, M., Luniewski, A. W., Niblack, W., Petkovic, D., Thomas, J., Williams, J. H., and Wimmers, E. L. Towards heterogeneous multimedia information systems: the Garlic approach. In *RIDE-DOM '95 :5th Int'l Workshop on Research Issues in Data Engineering: Distributed Object Management*, pages 124–131, 1995.

[5] Castro, M., Druschel, P., Hu, Y. C., and Rowstron, A. Topology-aware routing in structured peer-to-peer overlay network. Technical Report MSR-TR-2002-82, Microsoft Research, 2002.

[6] Chawathe, S. S., Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J. D., and Widom, J. The TSIMMIS project: Integration of heterogeneous information sources. In *16th Meeting of the Information Processing Society of Japan*, pages 7–18, 1994.

[7] Dabek, F., Kaashoek, M. F., Karger, D., Morris, R., and Stoica, I. Wide-area cooperative storage with cfs. In *SOSP '01: Proc. of the 18th ACM Symposium on Operating Systems Principles*, Chateau Lake Louise, Alberta, Canada, Oct. 2001.

[8] Ege, R. K., Yang, L., Kharma, Q., and Ni, X. Three-layered mediator architecture based on dht. In *ISPAN: 7th International Symposium on Parallel Architectures, Algorithms, and Networks*, pages 313–318, Hong Kong, May 2004.

[9] Ezenwoye, O., Ege, R. K., Kharma, Q., and Siddique, S. Electing a global mediator in a three-layer mediator. In *IEEE SoutheastCon 2005 Conference*, Ft. Lauderdale, Florida, Apr. 2005.

[10] Ezenwoye, O., Ege, R. K., Yang, L., and Kharma, Q. A mediation framework for multimedia delivery. In *MUM2004: Third International Conference on Mobile and Ubiquitous Multimedia*, Maryland, Oct. 2004.

[11] Flenner, R., Abbott, M., Boubez, T., Cohen, F., Krishnan, N., Moffet, A., Ramamurti, R., Siddiqui, B., and Sommers, F. *Java P2P Unleashed*. Sams, first edition, 2002.

[12] Garcia-Molina, H., Quass, D., Papakonstantinou, Y., Rajaraman, A., Sagiv, Y., Ullman, J. D., and Widom, J. The TSIMMIS approach to mediation: Data models and languages. *Journal of Intelligent Information Systems*, 8(2):117–132, 1997.

[13] Hildrum, K., Kubiatowicz, J., Rao, S., and Zhao, B. Distributed object location in a dynamic network. In *Proc. of 14th ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, Aug. 2002.

[14] Kharma, Q., Ege, R. K., Ezenwoye, O., and Yang, L. Data integration in a three-layer mediation framework. In *IEEE SoutheastCon 2005 Conference*, Ft. Lauderdale, Florida, Apr. 2005.

[15] Kharma, Q. and K.Ege, R. The impact of using DHT in 3-layered mediator framework. In *ICTIT: International Conference on Telecomputing and Information Technology*, Amman, Jordan, Sept. 2004.

[16] Maymounkov, P. and Mazieres, D. Kademlia: A peer-to-peer information system based on the XOR metric. In *Proc. of the 1st International Workshop on Peer-to-Peer Systems*, Cambridge, MA, Mar. 2002. Springer-Verlag version.

[17] of Commerce, U. D. Secure hash standard. Technical Report FIPS 180-1, NIST National Technical Information Service, Apr. 1995.

[18] Ratnasamy, S. *A Scalable Content-Addressable Network*. PhD thesis, University Of California at Berkeley, 2002.

[19] Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Shenker, S. A scalable content-addressable network. In *Proc. of ACM SIGCOMM*, San Diego, CA, Aug. 2001.

[20] Roth, M. T., Arya, M., Haas, L., Carey, M., Cody, W., Fagin, R., Schwarz, P., Thomas, J., and Wimmers, E. The Garlic project. In *SIGMOD '96: Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, page 557. ACM Press, 1996.

[21] Rowstron, A. and Druschel, P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. of the 18th IFIP/ACM Int'l Conf. on Distributed Systems Platforms*, Heidelberg, Germany, Nov. 2001.

[22] Rowstron, A. and Druschel, P. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *SOSP '01: Proc. of the 18th ACM Symposium on Operating Systems Principles*, Chateau Lake Louise, Alberta, Canada, Oct. 2001.

[23] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of ACM SIGCOMM*, San Diego, Aug. 2001.

[24] Stoica, I., Morris, R., Liben-Nowell, D., Karger, D. R., Kaashoek, M. F., Dabek, F., and Balakrishnan, H. Chord: A scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, Feb. 2003.

[25] Wiederhold, G. Mediators in the architecture of future information systems. *Computer*, 25(3):38–49, 1992.

[26] Yang, L. and Ege, R. K. Dynamic integration strategy for mediation framework. In *SEKE: Software Engineering and Knowledge Engineering*, Taipei, Taiwan, Republic of China, 2005.

[27] Yang, L., Ege, R. K., Ezenwoye, O., and Kharma, Q. A role-based access control model for information mediation. In *IRI: Proceedings of the 2004 IEEE International Conference on Information Reuse and Integration*, pages 277–282, Las Vegas, NV, Nov. 2004.