THE IMPACT OF USING DHT IN 3-LAYERED MEDIATOR FRAMEWORK

Qasem Kharma, Raimund K. Ege

Secure Software Artechiture Laboratory
School of Computer Science, ECS234
Florida International University, Miami, FL 33199
{qkhar002 | ege }@cs.fiu.edu

ABSTRACT

Delivering multimedia data efficiently is the goal of our mediator framework. Before data can be served, it must be found. Since our data sources are subject to dynamic change and routes to them subject to quality of service (QoS) considerations, it is the purpose of this paper to investigate distributed searching techniques, known as distributed hash tables (DHT), and to arrive at a suitable search algorithm. The Chord, CAN and Pastry DHT algorithms are compared and a hybrid DHT algorithm is proposed that has the properties of efficient search while being able to handle dynamic change in the configuration and QoS situation.

Keywords: mediator, middleware, software architecture, integration, P2P, DHT.

1. INTRODUCTION

The context of the research reported here is our ongoing effort [1] to define and build a multi-layered mediator-based multimedia architecture that will provide a dynamic, scalable framework for telecommunications software environments. The architecture 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. XML request and their decomposition is done at the "integration" layer. The third layer is the "homogenization" where a connection to actual data sources is established. Figure 1 depicts the framework. The "integration" layer consists of mediators that successively decompose a XML request into smaller XML requests that are closer to the data sources that are served-up by the "homogenization" layer.

In this paper, the integration layer is the only layer which will be considered. The integration layer represents a special kind of knowledge which is the composing/decomposing of XML schemata and routes. Instead of maintaining a central Schema Repository Server which manages and handles all schemata, we opt for a distributed search mechanism that uses the mediators in the integration layer as nodes in a Distributed Hash Table (DHT) approach.

DHT algorithms can be classified into three categories [2]: Skiplist-like routing algorithms such as the Chord algorithm, Routing-in-Multiple dimensions algorithms such as the CAN algorithm, and Tree-like algorithms such as the Pastry algorithm. Another schema is to classify DHT algorithms according to their basic routing geometries [3], such as tree, hypercube, butterfly, ring, XOR, and hybrid. The general idea of DHT is that each node maintains information about its neighbors in the system: no node has all the information, and some information is duplicated so when a node fails, the whole system will not fail. This paper describes and evaluates three of these algorithms and proposes a DHT-based search algorithm for our mediator-based architecture.

The reminder of this paper is organized as follows: Section 2 illustrates three DHT algorithms and their characteristics. Section 3 compares these algorithms. Finally, Section 4 adapts an algorithm to be implemented in our mediator architecture.

2. DHT LOOKUP ALGORITHMS

In peer-to-peer (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 [2]: 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

This material is based on work supported by the National Science Foundation under Grant No. HRD-0317692.

Copy Rights

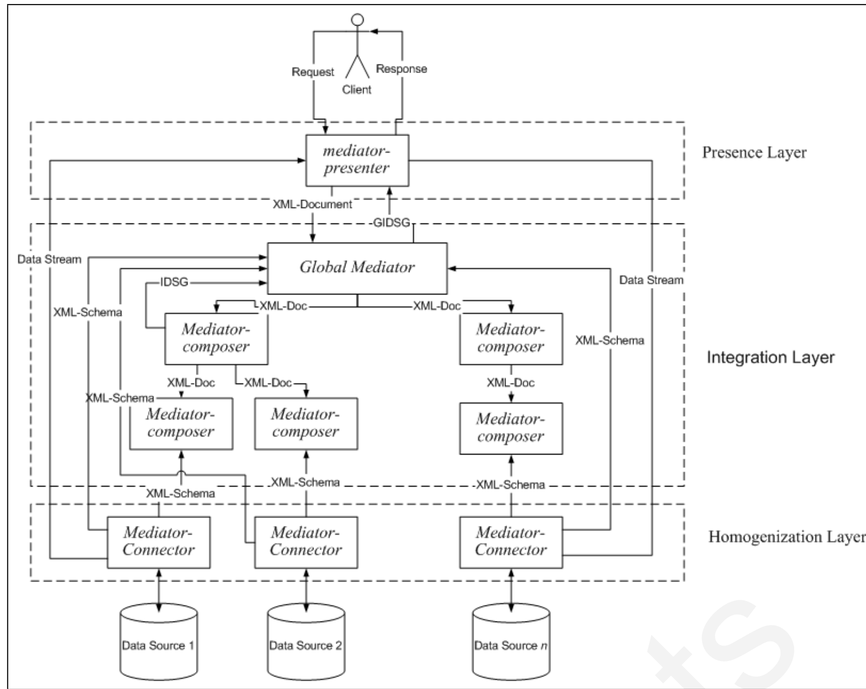

Fig. 1. Three-Layered Mediator Architecture

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.

[2] classifies DHT algorithms into three categories:

1. Skiplist-like routing algorithm: The Chord algorithm [4,5] 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 SHA-1 [6] 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. Section 2.1 discuss Chord algorithm in more detail.

2. Routing in multiple dimensions: The CAN [7] 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-dimension. In addition to storing a chunk of the DHT in the zone, each zone maintains information about its neighbors in the d-dimension. Section 2.2 describes this algorithm in more detail.

3. Tree-like algorithms: Tree-like algorithms, such as Pastry[1] [8, 9], Tapestry [10], and Kademlia [11], 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. Tapestry [10] maintains routing mesh which is an overlay network that links between the system's nodes that share prefix. Tapestry takes in the consideration the stretch which is the ratio of the actual distance to the shortest distance. In section 2.3 , Pastry algorithm will be discussed in detail as an example of the Tree-like algorithm category.

2.1. Chord Algorithm

Chord was proposed by [4, 5], and one of its advantages is its simplicity. The algorithm uses a consistent hashing function such as SHA-1 to assign a m-bit key or identifier to each node. The identifiers which represent nodes are ordered on an identifier circle or Chord ring modulo 2^m. Thus, Chord has a ring geometry [3].

Like adding a node, adding a key is done using a consistent hash function to assign a key identifier to the key to be added. Unlike a node identifier, a key identifier is generated by hashing the key itself while a node identifier is produced by hashing the IP address. After generating a key identifier, the key will be assigned to the first node whose identifier is equal to or greater than the key's identifier. This node is called

[1] Using Pastry in this paper as an example of the Tree-Like algorithms does not mean that Pastry has more or less feature than the other algorithms in this group. Each algorithm has its own features and comparing algorithm within the same category is beyond the scope of this paper.

Copy Rights

the successor node of the key k, and this function is denoted by successor (k).

The finger table is where the Chord algorithm maintains additional routing information in each node besides the successor of the node. Each node maintains a finger table which includes the identifiers of nodes that succeed the node n by at least 2^{i-1} on the Chord ring where $1 \leq i \leq m$ and each node maintains m entries which may have duplicated values in its finger table. Although the finger table contains m entries, only $O(\log_2 n)$ entries are distinct since some of the entries will be duplicated. The Chord algorithm maintains the following:

- finger[k]: first node on the circle that succeeds $(n + 2^{k-i}) \bmod 2^m$ and $1 \leq k \leq m$
- successor: the next node on the Chord ring
- predecessor : the previous node on the Chord ring

To locate a key, the node which received the request sends the request to the node that immediately precedes to that key. If the key is between the node id and the first successor, the request will be forwarded to the successor. Otherwise, the node checks the finger table to find an immediately predecessor to the key.

P2P assumes a dynamic joining and leaving of nodes. The Chord algorithm runs a "stabilization" protocol to ensure that each node's successor pointer is up to date, and fix_fingers function periodically to make ensure that the finger table is correct, and check-predecessor to ensure that its predecessor is available. Adding a new node requires the successor pointer to be fixed while the finger table will be fixed eventually by running the "stabilize" procedure periodically.

It is highly recommended that a list of $2(\log_2 n)$ successors be maintained in each node. Adding a new peer or removing an existing one can be achieved by $O(\log_2^2 n)$ messages to maintain the successors' lists [12, 13]. A "stable state" is a state in which the contents of all routing tables contain correct information (pointers and finger tables). Although the Chord system will not usually be in a "stable state", Chord algorithm can route the request correctly[2] in $O(\log_2 n)$.

2.2. CAN

The CAN [7] organizes nodes in virtual d-dimensional Cartesian coordinate space on d-torus; thus, CAN has hypercube geometry [3]. This space is divided among all nodes in the system, and each node maintains a chunk or zone of the hash table of its adjacent zones. Like Chord, CAN does not enforce any hierarchal structure of naming. The CAN's virtual space is used for storing the (key, value) pairs. The key is mapped in the space using a hash function. Then, the pair (key, values) will be stored in the node that owns the corresponding zone in which the mapped point is lying.

A node in the CAN network maintains a coordinate routing table of its immediate neighbors in the virtual space. Two nodes are neighbor in CAN if they share d-1 edges in the virtual space and are adjacent in one dimension.

When a node receives a request, it routes the request towards its destination by forwarding it to the closed neighbor to the destination. There are many available routes from a source to a destination, so the receiver node routes the request to the best available path using a greedy algorithm. Thus; the path is building dynamically, and a node failure will not affect the CAN algorithm. The greedy forwarding algorithm fails when a node looses all its neighbors in a certain direction. In this case, an alternative multicast algorithm such as expanding ring search can be used. The routing process can be achieved in $O(N^{\frac{1}{d}})$.

Adding a new node to a CAN system depends on the number of dimensions. Only $O(d)$ nodes, the split node and its neighbors, will be affected. The new node joins the system by finding a zone in the system and split that zone between the old and new node.

When a node wants to leave the system voluntary, it will return its zone to one of its neighbors. The neighbor which will take over must be either able to merge the two zones into a valid zone or handle the smallest origin zone.

In CAN, nodes update their zone coordinates by sending periodic update messages which include the sender's coordinates to its neighbors. Every node in a CAN system sends a periodic update message. A node failure can be discovered when a node update message is not received by its neighbor for long time. The node which discovered the failed node will set a takeover timer. When the timer is expired, it will send a "TAKEOVER" message including its own zone value to all the failed node neighbors. When a node receives a "TAKEOVER" message, it will check the zone value in the message. If the zone value in the message is smaller than its values, the receiver will cancel its own timer if it was already initiated. Otherwise, the receiver will start a "takeover" process.

In case of node failure, a "takeover algorithm" ensures one of the failed node's neighbors will take over. The data stored in the failed node's DHT will be lost temporary until the state is refreshed by the data holder. The data holders periodically refresh the pointers which are stored in the DHTs to their data.

2.3. Pastry Algorithm

Pastry [8, 9] maintains nodeIds of 128-bit. Each node in Pastry algorithm maintains a state table which has at most $(2^b) * \lceil \log_{2^b} n \rceil + L + M$ entries (when M and L = 2^b,then the state table will be at most $(2^b) * (\lceil \log_{2^b} n \rceil + 2)$ where n is the number of the nodes in the system, and b is a configuration number, usually

[2]the proof of Chord correctness can be found in "MIT LCS Tech Report" http://www.pdos.lcs.mit.edu/chord/

4). The nodeIds and the keys are sequences of digits of base 2^b.

Each node in Pastry maintains a state table which maintains three kind of information: a routing table, a neighborhood set, and a leaf set. A node's routing table maintains information about the other nodeIds that shares a prefix with the node. Each row in the routing table represents prefix of length i where $0 \leq i < \lceil \log_{2^b} n \rceil$. And, each row contains 2^b which are all possible values at the digit $i + 1$ where i is the row number. Some of those possible nodeIds do not exist; thus, each row will contain at most 2^b. Note that one entry in each raw will be the current nodeId. The nodeIds in the routing table are chosen according to a proximity metric which is used for determining the nodeId to be maintained in the current node according to a predefined criterion, such as the number of routing hops or geographic distance. Hence, the routing table is composed of $O(2^b * \lceil \log_{2^b} n \rceil)$ nodeIds. A node's leaf set contains L nodeIds such that L/2 nodeIds are the numerically closest largest nodeIds and L/2 are the numerically closest smallest nodeIds. A node's neighborhood set maintains information about M nodeIds which are closed to the nodeId according to proximity metric. The values of M and L can be either 2^b or $2 * 2^b$. The routing table maintains a tree geometry while the leaf set and neighborhood set maintains a ring geometry; thus, Pastry has hybrid geometry [3].

Unlike Chord and CAN, the Pastry algorithm routing is based on seeking a nodeId that shares as long as possible prefix with the request key. When node "A" receives a request with key K, the node "A" checks whether K is within its leaf set range or not. If K is within the leaf set range, the message will be forwarded to the minimal nodeId within the leaf set. If K is not within the leaf set range, the node will check its routing table for a nodeId which has at least one digit longer with K than the current nodeId. If no such nodeId exits, the message will be forwarded to the nodeId which is numerically closest to the K. Such nodeId can be found in either the routing table or neighborhood set. The routing in Pastry requires at most $\lceil \log_{2^b} n \rceil$ since at most cases the routing will use the routing table.

When a new node wants to join the system, it must know about a nearby node in the system. For example, the new node can find such a node using IP multicast or inquiring the system administrator. The new node sends a "join" message to the "nearby" node. Then, the "nearby" node routes the message to an existing node that has a closest numerical id to the new node. All nodes on the path from the "nearby" node to the closest numerical id to the new node send their state tables. The new node will build its state table based on those tables and may request some additional state tables. The new node's initial neighborhood set will be as same as the "nearby" node's neighborhood set, and the new node's initial leaf set will be as same as the closest numerical id node. The new node routing table can be constructed from the state tables which were sent to the new node. Assume that there is no common prefix between the new node and the "nearby" node; then, the first row in the new node routing table can be obtained from the "nearby" node. The second row can be obtained from the second node on the routing path since the Pastry algorithm routes the request to the node which has one more digit in its prefix with request key, and so forth. After constructing the new node's state table, the new node will inform any node which needs to be informed of its arrival.

The Pastry algorithm uses "lazily" repairing. In other word, when a node is discovered not to be available, an action will be taken to update the affected state table. Since the leaf sets of adjacent nodeIds overlap, the live node which discovers a failed node in its leaf set will update its leaf set by retrieving the leaf set of one of its neighbor nodeIds. If the failed node is in the neighborhood set, the live node will ask its neighbors for their neighborhood sets and measure the distance with the nodes in them and chooses the most appropriate one. If the failed node is in the routing table, the live node will ask another nodeId in the same row for an entry in its routing table in the same row and column of failed node. If no such node exists, the live node will contact a node in the next row.

3. COMPARING CHORD, CAN, AND PASTRY

The purpose of this section is to highlight the features of each of these algorithms and to analyze their complexity. Each of those algorithms has many good features. In general, it is impossible to describe a general algorithm as the best or ultimate solution.

Most of DHT algorithms share some common features such as scalability, maintaining knowledge about some neighbor nodes in the system, and trying to minimize the number of messages sent over the network to maintain systems. Each of the aforementioned algorithms maintains these properties with different approaches. Minimizing the number of messages is critical in P2P, and it is the criteria used to measure the system complexity. P2P complexity is considered by two issues: messages used to route a request and messages used to maintain the system.

The idea of DHT is that each node in a system maintains information about some other nodes in that system, and the system will use this stored information to route a request. As aforementioned, Chord maintains information about $\log_2 n$ finger entries and r successors (r is recommended to be $2 \log_2 n$). The only required information in Chord is the immediately successor ID, and the others are used to improve the performance. Each node in a CAN system maintains information about the nodes that share $d - 1$ edges in the virtual space and be adjacent in one dimension with the node; thus, each node needs to maintain infor-

mation about $O(2d)$ nodes. Pastry uses a different approach in which every node maintains a state table that includes three kind of information: a routing table– nodes which share a prefix with the node, a neighborhood set– the numerically closest nodeIds, and a leaf set– closed to the nodeId according to proximity metric; hence, each node in Pastry maintain $O(2^b * (\lceil \log_{2^b} n \rceil + 2))$ nodeIds.

Another important property is the degree of freedom in choosing nodes to build a routing table and in choosing the routing path. A node in a CAN system has only one possible neighbor set, but it can have several possible routing paths since CAN can route in any direction, unlike the tree geometry which limit the possible routing path to one path. On the other hand, nodes in Chord or Pastry have many possible valid neighbor sets. Pastry, as an example of tree geometry, can choose a node's routing table from $n^{(\log_2 n)/2}$ possible routing tables. However, after constructing the routing table, only one path can be selected. Finally, if a node in Chord has the freedom to choose its successors, for instance based on some proximity metric, $n^{(\log_2 n)/2}$ possible different successors can be chosen to construct the list. In chord, $(\log_2 n)!$ possible path exists between two nodes having distance $O(n)$. In a Pastry system, information maintained in each node is based on proximity selection to choose the "closest" nodeId. This approach can be implemented in Chord to select successors, but it cannot be implemented in CAN.

Not only is it important to maintain as little as possible information in each node, but also the cost of maintaining this information valid when a node is joining or leaving the system. This cost is measured by the number of messages needed to be sent to maintain the routing information. In Chord, $O(\log_2^2 n)$ messages to maintain the successor list which is sufficient to route a request. Also, Pastry needs $O(\log_2^2 n)$ messages to be sent to maintain its routing information. While, CAN needs $O(d)$ since the split node and its neighbors need to be updated.

Chord and Pastry route a user request to the destination in $O(\log_2 n)$ messages, but CAN depends on the number of the system's dimensions. The lookup cost in CAN is $O(n^{\frac{1}{d}})$ messages where d is the number of dimensions. Thus, if $d = \log_2 n$, Chord, CAN, and Pastry lookup cost are $O(log_2 n)$. However, it is impossible to maintain $d = \log_2 n$ since the number of nodes in P2P systems is not fixed and can be changed at any time.

4. THREE-LAYERED MEDIATOR ARCHITECTURE

4.1. The Architecture

A mediator handles the incongruence between client request and source data. For our target - telecommunication software - a single mediator does not suffice. The source data will be mediated by successive mediators into a form acceptable to the client: the mediation is done by chains of successively connected mediators. Since the source data can be collected from multiple sources, the chain will split at points: actually forming a tree. Since our target software must handle multi-media data in a secure and quality-oriented (QoS) fashion, the building of the mediation tree will not be static but rather it will grow and shrink, even data sources might change. In summary, our research is focused on a dynamic architecture for telecommunication multimedia delivery software. This architecture which features 3 layers: presence, integration, and homogenization. Presence layer is the interface to the client, which can be any computing device such as a computer, a PDA, or any special purpose device; the presence layer is also responsible for caching and buffering data streams. Integration layer analyses requests, finds connectors, and forms the Integration Data-Structure Graph - IDS [14–16]. Homogenization layer translate heterogeneous data sources into XML format [1].

Within this architecture, not all mediators are the same. We differentiate between three kinds of mediators: mediator-composer deployed in the integration layer, mediator-presenter deployed in the presence layer, and mediator-connectors in the homogenization layer. The client first connects to a mediator-presenter which will be responsible for caching data, converting from/to client format to/from composer format. Mediator-composers' functionalities include decomposing/composing schema and finding path(s) to the connectors. A special kind of mediator-composer is called the Global Mediator which will be responsible for receiving and handling a user request. A new Global Mediator is elected for every new request based on predefined QoS criteria. The Global Mediator forwards the request to other mediator-composers. At the lowest level of the mediator hierarchy, mediator-connectors are located. Data sources can be accessed through mediator-connectors only.

Unlike mediator-composers, mediator-connectors will not play any role in routing a request. The mediator-connector's role is an interface between the mediator system and data sources by mapping the data sources into XML schemata.

The user's request is sent to a Global Mediator [1]. The Global Mediator receives and responds to a XML request. More than one mediator-composer will need to cooperate to handle a single request. The DHT will be implemented in the mediator-composers of the system. Once the connector, which maps the requested data, is reached, the Integrated Data-Structure Graph will be composed by the Global Mediator and this tree will be forwarded to the mediator-presenter which will use it to stream data directly to/from the mediator-connectors.

Table 1. DHTs Properties (Note: Space complexity was calculated based on the recommended values. In CAN,assume that the space is partitioned into n equal zones.)

Property	Chord	CAN	Pastry
Number of nodes in a routing table	$3\log_2 n$	$2d$	$2^b * (\lceil \log_{2^b} n \rceil + 2)$
Number of possible routing tables	$n^{(\log_2 n)/2}$	1	$n^{(\log_2 n)/2}$
Messages to maintain the system	$O(\log_2^2 n)$	$O(d)$	$O(\log_2^2 n)$
Number of hops	$O(\log_2 n)$	$O(dn^{\frac{1}{n}})$	$O(\log_2 n)$
Symmetry	Asymmetric	Symmetric	Symmetric

4.2. Using a DHT in the architecture

All mediator-composers need to cooperate in order to find the connector(s) to the desired data source(s). In order to find the connector(s), the route from the Global Mediator through composers can be found using DHT instead of having a central repository of the connectors' XML schemata. Although it is possible to use one of the aforementioned DHT algorithms by defining what values will be mapped, we elected to build a hybrid algorithm of Chord and Pastry: this new algorithm maintains some features of both but adds important Quality of Service (QoS) criteria.

Unlike CFS [17] which is a file storage for blocks based on Chord, and PAST [18] 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 distributed pointers to the data which will be accessed through connectors.

Only mediator-composer plays a role in routing. The composers are distributed on a logical ring, like Chord. Unlike Chord, the composers maintain successor list, predecessor list, finger table, and a list of most recent reached composers. The list of most recent reached composers is to maintain information about the composers which could solve last few requests. Recall that although the 3-layer mediator system is flexible and can be run for any problem domain, in practical the system will be installed for a specific domain. Therefore, there are some keywords will be frequently repeated in clients' requests. For instance, if the system is installed in a medical system, words like patient, name, xray, insurance, and so on will be frequently repeated.

The first thing is to decide what values will be distributed. In our architecture, all messages between mediators are in XML format and each composer maintains some XML schema which will be used to decompose/compose the XML request in order to match a XML schema that is stored in a connector. When the system administrator adds a new data source by starting a new connector, the connector will convert the data source structure into XML schema and send the schema to a composer. Then, the composer coverts the XML schema into its corresponding tree. Next, the hash function maps the XML tags or elements which are now nodes in the tree onto keys which will be distributed over the peers.

The mediator-composers decompose the incoming request or simplify the incoming request by adding subtree(s) to the original request until all the tree leaves representing connectors or no further decomposing can be done. Once all the leaves refer to connectors, the composer creates dependency relationships between common nodes.

Second, when a new node (composer) wants to join the system, it will send a "join" message like Chord. The new node will join the P2P system if it is a composer. When a new connector joins the system, it will handle its XML schema to any composer on the ring. The connector will not be added to the ring since it will not play any role in finding paths.

Finally, the system maintains the validity of its peers using a "lazily" update mechanism, similar to Pastry. The system is not aggressive to maintain a stable state. When a node or a composer forwards a request to another node, it sets a timer to receive an acknowledgement message. The peer (composer) starts the recovery mechanism if the acknowledgement message is not received with the timeout.

In short, our algorithm looks similar to Chord, but employs a proximity metric for QoS consideration, which is an idea gleaned from Pastry. Applying this DHT algorithm will enhance the performance in the mediator framework.

4.3. Message Format

There are several benefits of using XML documents in the mediator architecture: it helps with integration and naming; follows standards such as JXTA–a collection of protocol for P2P with Java, and the ease of conversion from/to a graph to/from XML. In maintaining the P2P ring, the standard JXTA [19] is used. For detail about the JXTA protocols see [19]. In this section, the general XML [20,21] schema for messages among composers is explained as follows:

```
<request>
  <xs:attribute name = ”request-ID”>
  <xs:attribute name = ”client-ID”>
  <xs:attribute name=”presence-mediator-ID”>
  <xs:attribute name=”global-mediator-ID”>
  <xs:attribute name=”request-text”>
</request>
<token>
  <xs:attribute name=”token”>
  <xs:attribute name=”classification”>
  <xs:attribute name=”destination”>
  <xs:attribute name=”destinationID”>
</token>
<decompose>
  <xs:attribute name=”decompose-request-ID”>
  <xs:attribute name=” decomposer-ID”>
  <token>
   <xs:attribute name=”token”>
   <xs:attribute name=”classification”>
   <xs:attribute name=”destination”>
   <xs:attribute name=”destinationID”>
  </token>
</decompose>
```

All the elements in the request section are mandatory. The client will send the request to a presence–mediator-presenter which will send an election request to a composer. Once the global-mediator is chosen, the mediator-presenter will fill all the attributes. The mediator-presenter generates a unique request ID, adds the client ID which also will be used for security and privilege checking, adds the presence (mediator-presenter) unique identifier ID, adds the global-mediator ID, and attaches the request as a string. This section will not change unless the global-mediator or the mediator-presenter fails; otherwise, it will be the header for any further messages related to this request.

The global mediator will tokenize the request. Initially, all the tokens' types will be marked as ”unknown”, their destinations' types are nil, and their destinations' IDs are nil. These values will be filled while the request is in process. The token classification can be unknown, keyword, or value. An ”unknown” type has no match in either composers or connectors. A ”keyword” has a match in either composer or connector. In short, ”keywords” at the lowest level point to element identifiers in a connector's XML schema, while ”values” point to values stored in the data source. The token section does not need to be in all messages regarding the request. However, the final response from the global-mediator to the presence-mediator must have the token section. The ”destination” can be either connector or composer that stores the corresponding XML schema to solve or handle that keyword.

The ”decompose” section is a composition of the first two sections. This section is used when a composer, including the global mediator, wants to decompose or break a request into sub-requests. A new sub request ID is generated for each decomposition message. Recall that the original request ID will not be affected since it is maintained by the request section. The ID of the composer which generated this composition message will be maintained in the ”decomposer-ID” which plays a similar role to the global mediator to this sub-request, except it will not generate an IDS graph. The decomposer returns the XML schema for the sub request which is a part of the IDS but does not generate it. Note that we don't maintain information about the target since it will be maintained by the P2P messages.

5. CONCLUSION

Implementing the DHT concept in our mediator architecture improves the scalability and fault-tolerance in the system. Besides, some QoS criteria, such as choosing the best neighbor based on the bandwidth, can be embedded in the system to improve its performance. The Chord algorithm provides a simple way to implement the DHT and adds provable correctness (reliability) to the system, but it requires too many messages to maintain the system. The CAN algorithm routes message in multiple, logical dimensions. The problem in CAN is that the space-versus-number of the hops (the path length) decision must be made before implementing the system. Although the Pastry algorithm needs more information to be maintained in each node, it supports an interesting use of proximity metric. Thus, a hybrid version of Chord and Pastry algorithm is chosen for our architecture. Like Pastry, a proximity metric will be implemented to maintain QoS and to use the lazily update mechanism to maintain the system. Besides, other characters such as adding a node, finger tables, and successor lists will be maintained.

Acknowledgment

The mediator system was implemented by five undergraduate students:
Erik Hamlin<erik5@myrealbox.com>;
Juan Braceras<fijijuan@aol.com>;
Levi Teitelbaum<levibaum@yahoo.com>;
Roberto McQuattie<quattie@yahoo.com>;
Eduardo Mosaihuate<emosaihuate@msa.com>

6. REFERENCES

[1] R. K. Ege, L. Yang, Q. Kharma, and X. Ni, “Xml based multimedia delivery framework for telecommunications environments,” Florida International University, SSA LAB, Miami, FL, Tech. Rep. 2003-1, July 2003, (to appear in I-SPAN 2004). [Online]. Available: mediate.cs.fiu.edu

[2] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, “Looking up data in p2p

systems," *Communications of the ACM*, vol. 46, no. 2, Feb. 2003.

[3] K. P. Gummadi, R. Gummadi, S. D. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica, "The impact of dht routing geometry on resilience and proximity," in *Proc. of ACM SIGCOMM*, Aug. 2003.

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

[5] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for internet applications," *IEEE/ACM Trans. Networking*, vol. 11, no. 1, Feb. 2003.

[6] U. D. of Commerce, "Secure hash standard," NIST National Technical Information Service, Tech. Rep. FIPS 180-1, Apr. 1995. [Online]. Available: www.itl.nist.gov/fipspubs/fip180-1.htm

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

[8] A. Rowstron and P. Druschel, "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.

[9] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron, "Topology-aware routing in structured peer-to-peer overlay network," Microsoft Research, Tech. Rep. MSR-TR-2002-82, 2002. [Online]. Available: ftp.research.microsoft.com/pub/tr/tr-2002-82.pdf

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

[11] P. Maymounkov and D. Mazieres, "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: Springer-Verlag version, Mar. 2002.

[12] F. Dabek, E. Brunskill, M. F. Kaashoek, D. Karger, R. Morris, I. Stoica, and H. Balakrishnan, "Building peer-to-peer systems with chord, a distributed lookup service," in *Proc. of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, May 2001.

[13] D. Liben-Nowell, H. Balakrishnan, and D. Karger, "Analysis of the evolution of peer-to-peer systems," in *ACM Conf. on Principles of Distributed Computing (PODC)*, Monterey, CA, July 2002.

[14] P. Buneman, S. B. Davidson, and D. Suciu, "Programming constructs for unstructured data," in *Proceedings of the Fifth International Workshop on Database Programming Languages*, Gubbio, Umbria, Italy, Sept. 1995.

[15] S. Abiteboul, S. Cluet, and T. Milo, "Correspondence and translation for heterogeneous data," in *Proceedings of Database Theory - ICDT '97, 6th International Conference*, ser. Lecture Notes in Computer Science, vol. 1186. Delphi, Greece: Springer, Jan. 1997.

[16] S. Abiteboul, "Querying semi-structured data," in *Proceedings of Database Theory - ICDT '97, 6th International Conference*, ser. Lecture Notes in Computer Science, vol. 1186. Delphi, Greece: Springer, Jan. 1997.

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

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

[19] The jxta project. [Online]. Available: http://www.jxta.org/

[20] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. (2001, May) Xml schema part 1: Structures. [Online]. Available: http://www.w3.org/TR/xmlschema-1/

[21] P. V. Biron and A. Malhotra. (2001, May) Xml schema part 2: Datatypes. [Online]. Available: http://www.w3.org/TR/xmlschema-2/

Copy Rights