Performance Evaluation of a Distributed Lookup System for a Virtual Database Server

Simon Oechsner, Tobias Hoßfeld, Phuoc Tran-Gia
Chair of Distributed Systems, Department of Computer Science, University of Würzburg, Germany
Email: {oechsner|hossfeld|trangia}@informatik.uni-wuerzburg.de

***Abstract*—Services offered today rely on large amounts of data that can be accessed fast and reliably. One technical solution providing both acceptable speed and high reliability are distributed databases, which can be accessed from an application as one virtual database server. The virtualization here hides complexity introduced by distributing the service to a set of machines. In this paper, we will present a DHT-based architecture implementing a lookup layer for such a database, which preserves important features such as self-organization from its DHT roots, but still offers a good performance for time-critical applications. Additionally, first analytical results are given, which show some of the basic mechanisms at work in such systems.**

I. Introduction

In todays services, large amounts of data, e.g., video and audio content, subscriber management and accounting data, have to be stored and made available to applications. One example area for this are mobile networks, where user data is stored and accessed very often during the time a user is booked in. The number of database accesses is potentially very large and therefore poses a challenge to the system, which has to show a high availability and performance while being at the same time resource-efficient and scalable.

Examples exist where the amount of data to be stored is large enough to warrant the deployment of a distributed database. This is the case, e.g., for mobile subscriber databases such as the Home Location Register (HLR) or a similar subscriber database of a large provider. Distributing the database system however adds additional complexity to the architecture. It means a basic searching and routing mechanism has to be implemented to look up single data sets. Also, the content should be partitioned in a way that distributes the load on the nodes. Thus, storage space consumption and the load on individual nodes is reduced, which is traded off with longer access times and a higher total load due to the forwarding.

In this paper, we will describe a component of such a distributed database which was designed to offer good performance while being resource-efficient. We have implemented a prototype of this system and will present a first analysis based on the mechanisms implemented in this prototype.

The paper is structured as follows. Section II reviews related work. In Section III we will describe the considered architecture, together with a discussion on the design aspects of the system. Analytical results are presented in Section IV. Section V concludes the paper.

II. Background and Related Work

The system presented in this work is to be placed in the context of a mobile subscriber database, such as the HLR. Architectures like the IP Multimedia Subsystem (IMS) include similar databases, so that more than one possible application exists [1]. There are even activities aiming at unifying and generalizing the storage of user data for mobile networks. The Common Profile Store (CPS) is an example for these efforts [2]. Here, one encapsulated database is used for a number of applications or services, instead of several proprietary smaller ones. This only emphasizes the need for a scalable system with respect to the approach described here, since the size of the system grows with the number of smaller, specialized databases that are integrated.

The architecture evaluated here belongs to the class of one-hop DHTs. This class of overlays has been investigated in a number of papers. For example, Gupta et al. [3], [4] evaluate a fully meshed overlay architecture. They show that the additional overhead needed to keep the complete routing table in each node up-to-date can be handled even for large overlays.

In [5], a different scheme based on tokens for maintaining the global routing table in a one-hop overlay was presented and compared to the mechanism of Gupta et al. Leong and Lik showed here that less bandwidth was used in their architecture, resulting in a more efficient system.

Another architecture and its analysis of a one-hop DHT was presented in [6]. Again, it was shown that this kind of system is indeed feasible in terms of message overhead and response times to keep the complete routing tables needed at each peer up-to-date.

A distributed database similar in spirit and with some of the same design issues was published under the name of Dynamo [7]. While our system implements the lookup layer of a distributed database, Dynamo constitutes a complete DB which is also based on a ring structure and follows similar design principles. The architecture differs in details because of the different applications. For example, since Dynamo is used as a service for the Amazon S3 business platform, it has to offer write access at all times for the end user. In contrast, the system presented here is allowed to block write requests in order to protect the data integrity. Additionally, we provide analytical results for such a system.

III. Architecture Description

We consider a distributed database which stores user information, such as the subscriber database in a mobile network operator domain described earlier. Due to the large amount of data stored, the database is distributed among several dedicated database servers, which form the back-end. To locate specific data entries in the back-end, a front-end layer offers the necessary lookup and forwarding functionality, i.e., the front-end resolves queries to the database and forwards them to the correct back-end server. This lookup layer basically stores pointers to the entries in the back-end, one pointer for each database key. These pointers take the form of $< key, value >$ pairs, with the key being a reference to the database key queried and the value holding the address of the back-end server where the associated database entry is stored.

Applications, such as accounting, location management, etc., issue queries to this database system, normally searching for one entry at a time. These queries may be LDAP messages or conform to other protocols suitable for accessing a database. Application queries therefore do only see one virtual database system, the internal complexity is hidden. They are received by a front-end server first, which then resolves the contained key to the address of the back-end server holding the associated entry, cf. Figure 1. Basic load distribution may be conducted before forwarding queries to the individual front-end servers, however, this work focuses on the lookup layer itself.

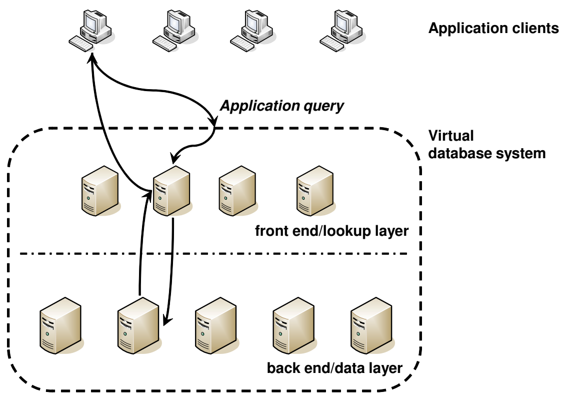

Fig. 1. The basic system architecture considered in this paper.

Mobile subscriber databases have several requirements which have to be taken into account when evaluating a possible instantiation of this architecture. Queries to such a database are frequently part of a longer process involving a current action of the subscriber, such as booking in, requesting a special service or changing the access network (horizontal or vertical handover). In order to keep the service quality experienced by the user high, the response times of the provider system as a whole and therefore also of the database have very tight timing bounds. For the same reason, the availability must be as high as possible.

Since the user data stored is critical for many services such as accounting, reliability and correctness is also required. This is prioritized sometimes even over availability. The system may even be unreachable or entries may be write-locked for short times as long as the correctness of the stored data can be guaranteed.

In order to facilitate a fast lookup process, the lookup information is not stored on disk at the front-end nodes, but is held in the memory of these nodes. Depending on the size of the database tables stored at the back-end, and on the number of searchable keys, this lookup table as a whole can reach sizes of several dozen GB. The simplest implementation of such a front-end system is to install a number of fully redundant lookup servers, each storing the full lookup table. Consequently, this means the amount of installed memory needed on each machine is in the range of the size of the lookup table. The number of servers depends on the system load, i.e., the arrival rate of the application queries.

In the following, we will describe a different, resource saving and self organizing implementation of this front-end system, based on the DHT principle.

A. Problem formulation

The application described above places several requirements on a lookup system that resolves queries for a back-end database. Since the database query is only a part of a larger process in the mobile environment which has severe timing constraints, the lookup itself must be kept as short as possible while being also highly reliable and available. If the user database at the back-end is not reachable, the experienced service quality for the user is diminished. On the other hand, the system should be as resource-efficent as possible. In this case, the enormous amount of RAM needed to store the lookup data is one of the highest cost factors when the data is stored fully redundant on several nodes. Therefore, saving memory consumption is a good approach to make the system more cost-efficient. Scalability is another issue that has to be addressed. The database size may grow over time, necessitating an expansion also of the front-end system. It should therefore be possible to add new servers and to integrate them seamlessly into the system. As a last consideration, the system should need only a minimal amount of human control and intervention. This saves costs, but also may lead to shorter response times in case of certain events if the system can react automatically.

The problem with these requirements, especially with speed and resource-efficiency, is that not all of these objectives can be accomplished at the same time. There is an implicit tradeoff between a fast search, where a fully redundant storage of the lookup data would be the best case, and a smaller lookup table per node, which necessitates a time-consuming internal routing process. This is complicated by the fact that lookup data distribution is not efficient for all resources, since internal traffic means more consumed bandwidth and a higher query load that has to be processed on the servers.

Also, if the data is not stored fully redundant, it has to be distributed in a way that still allows for load distribution. Moreover, this balancing should be resilient to node failures or to system expansions. At least, mechanisms should be implemented that allow for a correction of temporary imbalances.

B. Implementation

In order to fulfil these requirements, we have implemented a one-hop DHT, based on the Chord DHT. The interface of a DHT, namely the ability to store and retrieve pairs of keys and values, is perfectly suited to a lookup system. The original database search keys can be hashed to serve as hash table keys, and the storage locations of the associated data sets are the stored values.

Since short sojourn times are required, the routing paths through the overlay introduced by the DHT have to be as short as possible. On the other hand, if the memory consumption is to be lowered in comparison with a fully redundant system, the lookup table has to be partitioned in some way among the nodes. This necessitates at least a basic routing process. Combining these two requirements leads us to a fully meshed overlay structure, where at most one hop is necessary to locate a given entry in the DHT. This means that each node participating in the system has to know exactly what partition of the content is stored on each of the other nodes. Since the complexity of storing this information rises with the number of nodes, this potentially does lead to scalability problems. For our application however, we can assume system sizes in the range of 100 nodes for the considered application, where this can be handled easily. However, as described in Section II, similar systems (known under the term one-hop DHTs) have already been considered also for larger system sizes, and shown to be feasible.

Another important consideration concerns the load distribution in the system. If the lookup data is distributed unevenly among the nodes, and if we assume that in general each entry is queried with the same frequency, then the load distribution is also skewed. Moreover, the nodes storing more lookup data have to provide more memory. If it can not predicted how the load will be distributed, all nodes have to be outfitted with enough resources to handle the worst case, resulting in higher costs than necessary. To circumvent this problem, the random overlay ID assignment known from most DHTs is replaced by a deterministic positioning of the nodes in the overlay, assuring that each node is responsible for the same amount of data (cf. Figure 2).

While the considered application normally warrants the deployment of dedicated hardware with long mean time between failures (MTBF) intervals, it is nevertheless

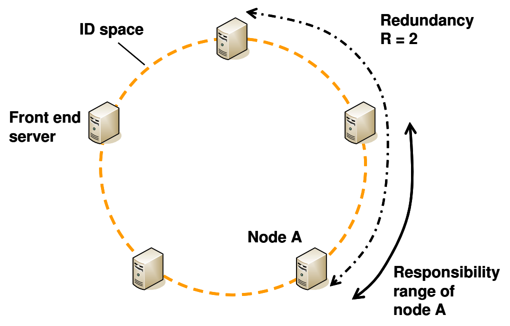

Fig. 2. Equidistant placement of nodes on the identifier ring

possible that a node or one of its components fails during normal operation. Node failures greatly upset the even load distribution in the system, leading to overload and/or congestion and should therefore be acted upon immediately. In order to keep the need for manual intervention low, an automatic reorganization algorithm is used to reassign the IDs of the remaining nodes, thus again placing them equidistantly on the identifier ring. Since this also includes a change in the responsibility ranges for each node, a redistribution of the lookup data is necessary in this case. A simple heuristic is used to keep the amount of data that has to be transmitted over the network low. Starting from an anchor node, each node is positioned in the correct distance (computed with the new number of active nodes) in the same order as before. Thus, there is a high probability that the old range of a node has a large overlap with its new range, meaning that the node already stores much of the data it needs in the new situation (cf. Figure 3).

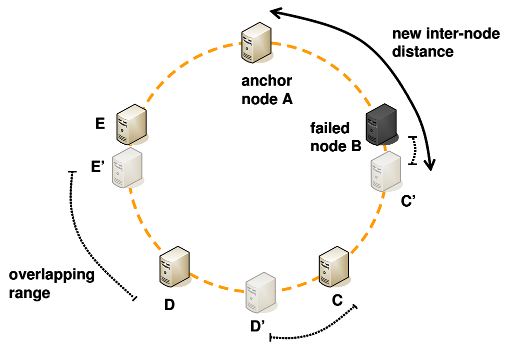

Fig. 3. Reorganization after one node failure

The same algorithm is used to add new nodes to the system, enabling an easy expansion and scaling with load. If more than one node is to be added at the same time, the new nodes are inserted equally distributed into the system, so that again the overlap between old and new responsibility ranges is high for the 'old' nodes.

To ensure that the probability for data loss is low even in case of node failures, each entry of the lookup table is stored on more than one node. Similar to the replication group in Chord, it is copied on the R successor nodes of the key of the entry, with the first successor being the node primarily responsible for the entry. Here, R is a tunable parameter that enables a tradeoff between resource savings on one hand and system load and availability on the other hand, as will also be shown in Section IV.

With these mechanisms, the normal system operation is as follows. An application issues a query to the database of which the lookup system is part of. These queries are distributed evenly among the lookup nodes, e.g., by means of a simple round robin load distributor. When a query reaches a lookup node A, this node first hashes the database key of the query. It then checks whether it stores the lookup entry associated with this key locally, including the redundantly stored data. If this entry is not stored locally, then the node determines which other node is responsible for it via its routing table, and requests the entry from that node. In either case, the original query is then forwarded to the according back end database server, whose address is the stored value of the $< key, value >$ pair of the lookup entry. The response to this query can then again be forwarded to the application.

In case of a remote lookup, the node with the lowest ID higher than the hashed key searched for is always selected as the responsible node. This eases the routing process in case of node failures, and should still return valid results as long as no data loss has occurred, which is only possible if R consecutive nodes fail in a short time interval. While the query load for one entry could also be distributed among the R nodes storing that entry, we assume that each entry is queried with the same frequency, and therefore no additional gain can be achieved by this measure.

IV. Analysis

In this section, we present a first analysis of the described system. This is based on a methodology presented in [8], where we considered a more general system and the parameters that influence its behaviour. We model the front end servers as queuing systems, with the complete network being a queueing network. Thus, we can analyze system characteristics like the sojourn time.

Each node is modeled as a M/GI/1 waiting system. In this first model, we make some simplyfying assumptions. First, the processing times for queries are assumed to be independently and identically distributed (iid), i.e., we do not discern the processing of external queries, internal queries or response forwarding. Second, we assume that the popularity of each database entry is the same. To conduct the analysis, we have to compute the arrival rate at each node, which is a function of the number of nodes and of the number of hops that are made internally.

A. Node load

Each node initially receives its fair share of the application queries. If we define the total initial arrival rate as λ_0, and the system consists of N servers, this initial load at a specific server corresponds to $\frac{\lambda_0}{N}$, cf. Figure 4. Of these queries, the server can only resolve a fraction locally, depending on the partition of lookup data it stores. Ideally, this fraction should be $p = \frac{R}{N}$, where R is the redundancy factor. The rest of the queries has to be resolved on a second server, and is consequently forwarded. This outgoing query flow is equally split among the $N-1$ remaining nodes. Due to the symmetry of the traffic flows, the same amount of queries (with rate $p \cdot \frac{\lambda_0}{N}$) is received. This traffic is therefore added to the external query flow.

All of the internal queries are answered, effectively doubling the internal traffic. This is due to the fact that the first node receiving the request is responsible for resolving the complete query and forwarding the response to the querying application. Some traffic could be saved if the database virtualization allows for the second node or even a back-end server to answer the query instead of the first reached node. However, we describe the most general and worst case here with the least assumptions about the interface between the external application and the virtual database system.

Finally, all application queries received by that server are forwarded to the back end database, and the answers are again forwarded to the first node to be forwarded to the application. Therefore, the full rate of $\frac{\lambda_0}{N}$ is again received from the back end and processed, leading to a total arrival rate of $\lambda^* = (2+2p) \cdot \frac{\lambda_0}{N}$.

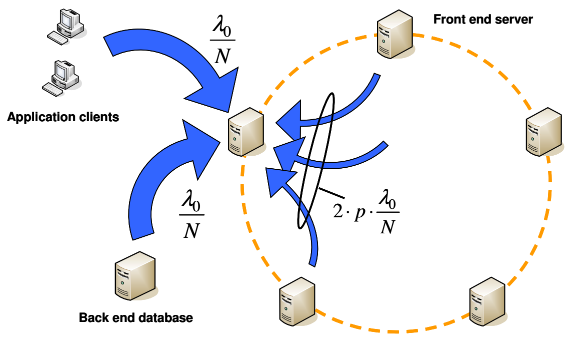

Fig. 4. The traffic flows influencing the node arrival process

B. Sojourn time

Based on this model, we can compute important characteristics like the mean sojourn time of queries or its coefficient of variation. For the following results, we assumed a service process with mean $E[B] = 1\,\mathrm{ms}$ and a coefficient of variation of $c_B = 0.5$. To model the internal network transmission, i.e., query forwarding from front-end server to front-end server, we use an exponential distribution with mean $0.3\,\mathrm{ms}$. We do not consider the querying of the back end database here, but focus on the time a query spends in the lookup system itself.

We will now present selected results from this analysis, which provide some basic insights into the presented system. One of the important parameters that influences the system behaviour is the redundancy factor. It has a direct influence on the probability that a query has to be forwarded internally in the lookup layer, and therefore on the total load a front end server experiences. Figure 5 shows the mean sojourn times of queries normalized by $E[B]$ for a system with 20 nodes and different redundancy factors, ranging from $R = 1$ (no redundancy) to $R = 20$ (full redundancy, no forwarding is required).

Due to the fact that the internal load is up to four times larger than the external load (i.e., the query load produced by the applications exclusively), only values up to 0.25 are experienced for the externally seen utilization $\rho = \frac{\lambda_0}{N} E[B]$ per front-end server.

The mean sojourn times increase for lower redundancy factors, which is due to the higher fraction of queries that have to be forwarded in the lookup layer, resulting in a higher load at the nodes and therefore longer waiting times. On the other hand, a higher redundancy also means that more data has to be stored on each node.

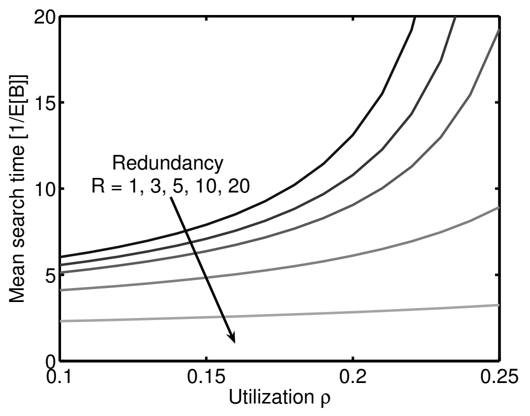

Fig. 5. Mean sojourn time for different redundancy factors for a system consisting of 20 nodes

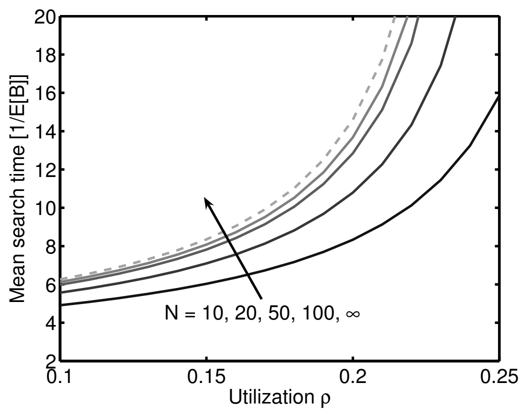

Fig. 6. Mean sojourn time for different system sizes

While this should be of no concern where cheap disk space can be used, it is a tradeoff to be considered for our application, as described earlier. In any case, the redundancy factor can be used as a parameter to tune the system to the needs of the operator.

C. System size

Another parameter influencing the system performance is its size in terms of the number of front-end servers. The larger the system is, the higher is the forwarding probability and therefore the internal load. Figure 6 shows this by again comparing the mean sojourn times for numbers of front end servers ranging from $N = 5$ to $N = 30$. The redundancy factor is set to $R = 3$. Additionally, the hypothetical case where every query is forwarded internally, i.e., $p = 1$ is represented as an upper bound (dashed line).

Again, the mean sojourn times increase with a larger forwarding probability. However, the load increase resulting from more front end servers diminishes for already large systems. This is due to the fact that the forwarding probability p grows fast for smaller systems, but is soon close to 1, e.g., $p = 0.85$ already for 20 nodes and $R = 3$.

D. Reorganization effort

An important aspect for the presented system design was its ability to react to changes in the topology, especially node failures. Since load distribution is a critical characteristic for the efficiency of the described architecture, it has to be restored as soon as possible after one or several node failures. The same is true for inserting additional nodes into the system, however, we assume that an expansion is executed in a more planned and controlled manner. Apart from this, the reduction of servers also increases the load on the remaining servers, even if an equal load distribution can be achieved. Therefore, it is a kind of worst case for the reorganization algorithm.

In order to conduct a first evaluation of the rather straightforward method implemented in our prototype, we conducted a Monte-Carlo simulation for different node failure scenarios. We varied the number of nodes that fail concurrently in a system consisting of 40 nodes. For a given number f of node failures, we selected a random subset S of the nodes with $|S| = f$. This experiment was repeated 10,000 times for one value of f, the shown intervals are for a confidence level of 99%. We recorded the amount of data that had to be moved during the reorganization phase in order to achieve equal load distribution again, relative to the total amount of data stored. We assumed here that the data is placed with roughly equal density on the identifier ring. Also, we neglected cases where enough successors of a failed node also fail, which results in the loss of data. Since in this case less data has to be transmitted over the network, the presented results are an upper bound, even if data loss is an undesirable event.

Figure 7 shows these results for different grades R of replication, ranging from $R = 2$ to $R = 4$. The amount of data that has to be transmitted increases for higher replication grades and a larger number of node failures. Also, the maximum amount of data moved in the worst case equals $R \cdot 100\%$, meaning that more data has to be moved than there is in the ring. This initially counter-intuitive characteristic stems from the fact that the responsibility areas of the reorganizing nodes may overlap, meaning that several nodes have to retrieve the same data sets if they have not stored them before the reorganization.

Due to fact that each node is responsible for a single, continuous range of the id space, the number of data entries that have to be moved to a new node is not equally distributed among all nodes. Especially the successors of a failed node normally have to be moved a larger distance in the identifier space than the following nodes. It is to be expected that different schemes, such as the one proposed in [7], might be able to reduce this unfairness, lowering the amount of data that has to be moved, and consequently also the time it takes to reorganize.

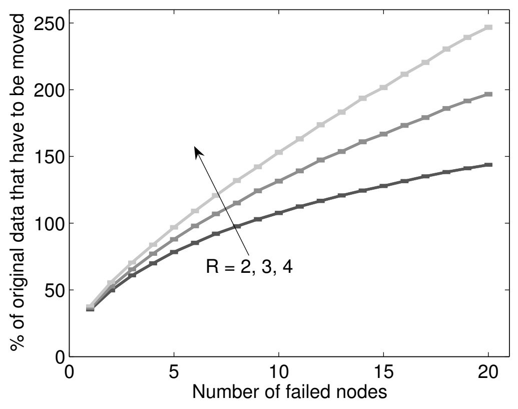

Fig. 7. Relative amount of data that has to be moved during reorganization

V. Conclusion

In this paper, we presented and evaluated an architecture for a virtual mobile subscriber database based on a one-hop DHT. The requirements and design issues influencing the system layout are described, and a basic overview on the developed architecture is presented. It is shown that the design caters to the necessities of the application. It provides a trade-off between the system sojourn time on one hand and resource consumption on individual nodes on the other.

A performance evaluation model was presented, the results derived from it giving some insights into the basic workings of the lookup system. The underlying tradeoff of less storage space per node against longer search times and a higher load on the system is illustrated. As future work, measurement results from an implemented prototype will be used to verify the analysis, as well as simulations. More emphasis will be placed on the system behaviour during transient phases, e.g., reorganization times.

Acknowledgments

This work has partly been performed in a project with Nokia Siemens Networks. The authors would like to thank especially Frank-Uwe Andersen and Kurt Tutschku for their help and efforts.

References

[1] 3GPP, 3rd generation Partnership Project, “IMS Specification,” http://www.3gpp.org/specs/numbering.htm, 2008.

[2] Franz-Josef Banet, Rodolfo López Aladros, and Stephan Rupp, “Common Profile Storage für Telekommunikationsnetze,” 8. Workshop des GI-Arbeitskreises Mobile Datenbanken und Informationssysteme, 2005.

[3] Anjali Gupta, Barbara Liskov, and Rodrigo Rodrigues, “One hop lookups for peer-to-peer overlays,” in *Proceedungs of the Ninth Workshop on Hot Topics in Operating Systems*, Lihue, Hawaii, May 2003.

[4] Anjali Gupta, Barbara Liskov, and Rodrigo Rodrigues, “Efficient routing for peer-to-peer overlays,” in *Proceedings of the First Symposium on Networked Systems Design and Implementation*, San Francisco, CA, March 2004.

[5] Ben Leong and Ji Lik, “Achieving one-hop dht lookup and strong stabilization by passing tokens,” 12th International Conference on Networks 2004 (ICON 2004), Singapore, November 2004.

[6] Luiz R. Monnerat and Claudio L. Amorim, “D1ht: A distributed one hop hash table,” Rhodes, Greece, April 2006, IEEE International Parallel & Distributed Processing Symposium.

[7] Giuseppe DeCandia, Deniz Hastorun, Jampani Madan, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels, “Dynamo: Amazon’s highly available key-value store,” in *Proceedings of the 21st ACM Symposium on Operating Systems Principles*, 2007.

[8] Simon Oechsner and Phuoc Tran-Gia, “Performance evaluation of a reliable content mediation platform in the emerging future internet,” in *Proceedings of the 20th International Teletraffic Congress (ITC20)*, 2007.