Integration of Shareable Containers with Distributed Hash Tables for Storage of Structured and Dynamic Data

Eva Kühn, Richard Mordinyi,
Hannu-D. Goiss, Thomas Moser
Complex Systems Design & Engineering
Vienna University of Technology
Favoritenstr. 9-11, 1040 Vienna, Austria
{eva,richard}@complang.tuwien.ac.at
{hdg,tm}@complang.tuwien.ac.at

Sandford Bessler, Slobodanka Tomic
Telecommunications Research Centre Vienna (FTW)
Donau-City 1, 1210 Vienna, Austria
{bessler, tomic}@ftw.at

Abstract

Structured, spatial-temporal data arises in many applications areas such as transportation, sensor networks or mobile services. Peer to peer networks are the natural choice for the distributed architecture required by these applications. However, a closer analysis of the available distributed hash table (DHT) based approaches shows their inefficiency since the data structure gets lost and the short liveness of the data leads to a high signalling traffic. In this work we propose a novel storage network based on an overlay of Space-based Computing Containers for storing, accessing and manipulating dynamic data.

1 Introduction

As wireless access becomes ubiquitous, and peer mobility increases, the Internet architecture faces many new challenges. Emerging mobility scenarios often require novel opportunistic routing, node-by-node reliable transport and disruption-tolerant behaviour, since the assumption of stable end-to-end connections is not true anymore. To meet these new requirements intermediate nodes must be able to cache large amounts of data. The trend for ”storage in the net” has been efficiently addressed by peer to peer storage networks such as PAST [10] [26], OceanStore [17], cooperative file system CFS [8]. However, when structured content such as contextual data including complex temporal attributes as well as geographical location attributes need to be efficiently stored, queried and acted-upon, the former storage network designs (supporting mainly files) renders unsuitable, since the information structure gets lost. A few P2P techniques such as the intentional naming system INS/Twine [3] or P-Grid [2] can maintain a structure with a hierarchical DNS-like addressing. However, the costs for such solutions are high, since keeping a P2P entry for each data causes a large traffic overhead, especially when the data elements are mutable and short lived.

In this paper we address the need for efficient storage of complex structured data by proposing a novel architecture which unifies adaptiveness of the P2P approach, particularly Distributed Hash Tables (DHT), with the query expressiveness and coordination support of the space-based computing paradigm [7]. This architecture is an enabler for applications that operate on distributed storage of structured data and require application-based routing and distributed coordination. It uses extended (virtual) shared memory [18], a middleware technology to store data objects in a space that can be shared among many applications/services, and an underlying location and mediation layer. The main contribution of this work is therefore a novel storage architecture in which the tuple space entities are uniquely addressable via an overlay network based on DHT technology. This approach has two advantages: firstly, it keeps the message traffic on the DHT level low and secondly, it allows remote client applications to directly manipulate the data entries in so called containers [11] by using advanced query expressiveness of the spaces technology.

The rest of the paper is organized as follows. In Section 2 we present an application scenario describing the requirements and motivating our approach. In Section 3 we introduce the two relevant technologies, that is, distributed hash tables and space-based computing. In Section 4 the proposed architecture is presented together with related design problems including addressing, replication, and handling of node arrivals and departures. A global API that hides these aspects from applications is presented. Section 5 illustrates the use of the architecture to support a transport telematics application and Section 6 contains concluding remarks.

978-0-7695-3575-3/09 $25.00 © 2009 IEEE
DOI 10.1109/CISIS.2009.53

2 Application scenario

A motivating study-case that we use to identify requirements and to illustrate the benefits of the proposed architecture is an intelligent transportation system (ITS) scenario. In this scenario fast moving vehicles communicate with a fixed, but geographically distributed infrastructure, as illustrated in Fig. 1.

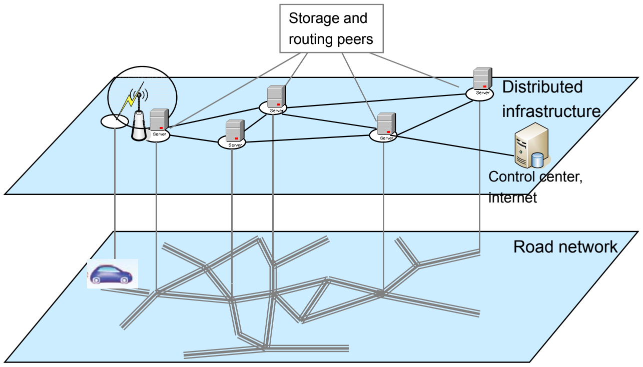

Figure 1. Intelligent transportation scenario

For this purpose so-called road side units (RSUs) are installed at some of the junctions or segments of the road network. On the one interface, RSUs are connected in a meshed wired broadband network, on the other interface they provide wireless access for the vehicles in their communication range, via dedicated short range communication protocols (DSRC [28]). In one type of application, vehicles act as providers of geo-temporal context data: they report on safety events measured by sensors along their paths, e.g. sudden use of breaks, humidity, temperature of the road, etc. Another service aims to distribute traffic events from the road provider towards *the relevant* RSUs. These events include alerts about incidents, road works, setting speed limits, etc. In both cases the data is geo-located and its relevance in space and time is limited to a certain region, moving direction and period of time. In Section 5 we describe how the event dissemination can be realized with the proposed architecture and the application interface.

3 Related Work

The following section introduces the DHT and the space-based computing technologies stressing the aspects needed in the system architecture presentation.

3.1 Space-based Computing

The idea of a shared data space was first created by David Gelernter in the 1980s [13]. He introduced a coordination language called Linda which operates on an abstract computation environment called tuple space. In the tuple space approach, processes communicate with the other entities in the environment by writing tuples (ordered sequences of data) into the tuple space. Sharing of data via spaces [6] is not a novel paradigm. It comes from parallel processing and was later considered for distributed environments. Due to its high-level abstraction of communication by simply reading and writing data from/into a shared space this paradigm fits to growing dynamics and collaboration in the network [23]. The processes interested in retrieving information useful for coordinating their activities perform blocking read, **rd**, or consuming read, **in**, operations specifying via a template. In case several tuples match the template of a data-retrieval operation, only one of them is selected non-deterministically. The communication always takes place between the processes and the space. This way the sending process does not need to know about the receiving process and there is no need for both processes to be connected at the same time. This decoupling of processes in both time and space takes away a lot of complexity in creating distributed applications. Gelernter calls this communication paradigm which is both decoupled in space and time generative communication.

There are a lot of implementations, like LighTS [24], JavaSpaces [12], TSpaces [22], GigaSpace [15], that follow the concept of the tuple space with associative search for the stored tuples. In our approach we use a space-based architecture, called XVSM (extensible virtual shared memory [20], [1]), that generalizes Linda tuple based communication [14] as well as several extensions to it – like reactions [25], programmable behaviour [5] to make the space extensible, and addition of other/more expressiveness to the coordination laws like the introduction of priority and probability tags [4] – in one single concept, namely the abstraction of *shared containers*. Entries are chosen to represent data in a shared container. An entry is a multiset of labelled values called tags. A tag can be used to represent either a payload or meta-data relevant for coordination. This way the entry offers a clear separation between user data (cf. message body) and coordination data (cf. message head). Like in the Linda [14] model, containers can be accessed by the operations **read**, **take**, and **write**. Each container has one or more coordination types that define the exact semantics of each operation. The different coordination types can be classified into *implicit order*, *direct access* and *content matching*. Implicit order refers to e.g. FIFO and LIFO order. Using FIFO coordination corresponds to seeing the container as a queue where **take** will always read and remove the latest written entry. Direct access allows selection of entries via their tags directly, using relational operators [19] (e.g. select entries with a certain tag that has a value less than 100) or range operators (e.g. select entries with a certain tag that has a value between 100 and 1000). Content matching allows to define user defined match makers [24], e.g. for Linda, RDF, or XML querying facilities.

A shared data space is a collection of containers which can be addressed via URLs in the internet. A lookup mechanism is assumed that resolves a published container name to its URL and that makes use of DHT techniques as decribed in section 4.1.

A container is extensible in a sense that its behaviour can be extended through aspects comparable to aspects in object-oriented programming languages [16] and space-based programming extensions for aspects [21]. Aspects are code fragments that react on certain events, in XVSM on operations carried out on a container. Interception points for aspects are pre and post of every operation, e.g. pre-read, post-write etc. The operation context can be selected to be the same transaction as of the base operation or not. Aspects serve to build higher level behaviour and APIs e.g. for dedicated replication strategies (see section 4.2) on top of a container.

A container can refer to other (sub)containers forming a more complex coordination data structure. The asynchronous and blackboard based space-based communication model allows programmers to explicitly control interactions among processes via shared data. It avoids a coupling through direct interactions between the agents, especially when mobile devices are assumed ([5], [24]).

3.2 Distributed hash table

The second technology in our solution approach is a structured peer to peer network. Structured P2P networks (Chord, Pastry, CAN) use Distributed Hash Tables and have been extensively studied in the last decade and successfully deployed to create scalable applications. The basic functions a DHT provides are to store a data object and to retrieve efficiently that data object from any peer in the network. For this, a very large identifier space (e.g. all binary combinations of 128 bits) is defined onto which data objects and node identifiers are mapped using a hash function. The design options to create the *key* for a data object vary from hashing the (string) name of the object, its location (URI), or the content itself. The storage/publishing function is not affected by the selected alternative *publish(key, value)*. A DHT offers basic self organization functions such as adapting the topology when a node arrives or leaves, and a better reliability since it may have a mechanism in place to replicate DHT entries. The lookup effort is for most DHTs O(log N) where N is the number of nodes. Among the various DHT algorithms, Pastry [27] with its implementation FreePastry deserves a more detailed discussion. The identifiers are selected from a 128-bit ID circular space. The routing of messages between the nodes is based on maximum prefix matching. For this purpose the routing table of a node has several rows, so that the n-th row lists those nodes that have a matching prefix of n with the current node ID. The leaf set is also a part of the routing table and contains nodes whose node ID are numerically close to the current node ID.

DHT entries are replicated by Pastry on a subset of the leaf set. The Pastry replication mechanism is responsible that the parameter *r*, the number of life replicas, remains invariant under node churn conditions. During a lookup operation, the routing protocol will search the node with the closest ID to the key K, however a neighbor node holding *replica(K)* might be found earlier on the routing path. Therefore, the *lookup(k)* operation might not deliver deterministically the same replica.

4 Architecture and design considerations

As mentioned before, the proposed system should keep the structure of information entities, e.g. maintain messages that are grouped and ordered according to certain criteria. The grouping can be achieved by putting the information entities in a tuple space container and, at retrieval time, applying filter and ordering selectors to that container. The addressing granularity in the overlay is therefore the spaces container. A XVSM container entity can be addressed as any other resource in the internet using the domain name of the host as a part of the resource URI. The addressing scheme for a container in the XVSM is an URI of the form *"tcp-java://mycomputer.mydomain.com:1234/ContainerName"*. The container reference is therefore dependent of the IP address of the localhost node. The indirection layer added by means of the DHT makes possible to hide IP address changes due to mobility, replication, etc. In case the IP address changes, the node fails or the containers are redistributed to equally share the load, a redirection at the DHT level would be appropriate. Once the current IP address is obtained via DHT, the container is addressed directly via IP routing. It would allow clients to lookup containers by *container name* only.

4.1 Managing the container replication through Pastry

The Pastry replication algorithm is based on an invariant replication of an object addressed by a key *K* on *r* nodes that are alive and nearest to key *K*. Similarly to the PAST storage system built on top of Pastry [27], replicas have to be managed in order to fulfill the invariant rule mentioned above. In the following discussion we will distinguish between replicas of DHT entries (DHT replicas) and replicas of the shared space containers (container replicas). An key-value pair entry in DHT looks like: *key = H(containerName)*, *value = containerReferenceURI*. The Pastry routing protocol makes sure that an alive replica is found. However, finding the DHT entry is not enough, its value has to point to the correct container replica. We describe how to correctly maintain the replicas. Let's consider first the replication of DHT entries: Pastry can always return the *replicaSet* for a certain key, that is a set of nodes that can be used to store replicas of that key. On each of these nodes a replica of the "stripped" container reference is stored under the key, that is the *ContainerReferenceURI* without the first part that defines the domain name. When reading this Value, the local node (the destination node of the lookup) completes the URI with the IP of the localhost, before returning it to the requesting client. This solution has two consequences:

- The DHT entries are the same for all replicas, thus they do not have to be manipulated or rewritten in failure case - a substantial advantage.
- Replicated DHT entries and the corresponding containers have to be on the same node. This constraint might limit the freedom in sharing the load between nodes, an assumption to be further checked in our simulations.

In case a node fails, the lost DHT entries are replicated in order to keep the number of copies for each entry equal to *r*. Specifically, in case the number of replicas is less than r, a DHT entry replica is automatically created by Pastry on those nodes from the set *replicaSet(K)* that have no replica. In addition to this DHT mechanism, the container corresponding to the newly created replica has to be copied as well on that same node.

4.2 Handling container replicas

Containers are passive components in which the entries change following operations such as write, take, or destroy. Therefore, the container replicas have to be updated, although the DHT pointers remain unchanged.

The design of the interactions related to a WRITE operation has to consider the replication mechanisms specific for the DHT implementation - in our case - Pastry. In the normal case the DHT operation *lookup(K)* will access the root node, but if the path to destination hits first a replica node, it is this node that is returned. Therefore, in this system we do not have a single master replication. We propose to use a delegation pattern as shown in figure 2. The first interaction *lookup(K)* will return the full address of a container replica (or the root). Within a container transaction, the requester peer proceeds with a direct container *write()* operation, followed by an *aspect* program that calls *replicaSet()* in order to obtain the other replica nodes, and repeats the *write* operation at the respective containers. The transaction to the first accessed container ends with the result. Figure 2 shows the interactions.

Since the scenario contains several classes of information, each of them with different requirements towards the replication strategy in sense of e.g. consistency, this kind of replication pattern is very suitable for e.g the information class where inconsistencies do not need to be handled immediately, like weather data. This also means, that the strategy needs less overhead and is therefore more efficient than strategies which must keep the data distributed always in a consistent way. Those inconsistencies can be solved locally at predefined time frames.

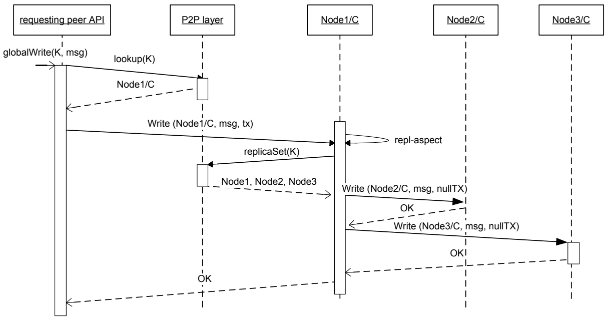

Figure 2. Handling container replicas

4.3 Handling joining and leaving nodes

If a node holding a key K is non responsive for a certain period of time, e.g 10 seconds, then Pastry triggers an adjustment of the leaf set routing entries in all affected nodes. Each node removes the failed node from the leaf set and includes the live node with the closest *nodeId*. The responsibility for the key K moves from the failed node to another near node that however has first to acquire a DHT replica of K. As we have seen in the previous section, a new DHT replica of K triggers the creation of new container replica on the same node.

If a new node joins the system, this node is included in the leaf set of L neighbors, and other nodes are removed from those leaf sets. If the key K points to one of these nodes, there remain only *r-1* replicas, and the new node has to require a replica for K. Therefore, the overall effect of a "join" is to move the container from the node dropped from the leaf set, to the new node.

4.4 Enhanced container application Interface

The mechanisms described above enable us to define a powerful application interface that on the one side hides the node lookup in the network, the replication management of

DHT entries and of containers. On the other side the tuplespaces paradigm offers the features described in Section 3.1, it supports advanced queries of data items in a collection, allows the modeling of queues and stacks, supports transactions, etc.

For application programmers the API [9] below is a compact, elegant abstraction that needs only the application specific container name for addressing it network-wide. The core methods are given below, whereas the whole framework handles in addition transactions and creation of aspect programs. (see [19] for more examples).

- **Create (containerName)**: publishes the name of the used container. Every time Pastry creates a replica entry on a node, a replica container is created as well. The process consists of three steps: 1) a DHT publish method call with the name of the container is executed; 2) Pastry forwards the value to those peers where the original entry and the replica of it should be placed; 3) on those peers containers with the provided name are created. Additionally, replication aspects are installed.

- **Read (containerName, timeout, transaction, selector)**: The client addresses the DHT first to lookup for the container URI. The returned value is then used to address the container directly. The selector allows ordering and filtering of tuples/container entries for reading.

- **Write (containerName, timeout, transaction, entries)**
  Take (containerName, timeout, transaction, selector)
  Destroy (containerName, timeout, transaction, selector)
  For each of the operations, first a DHT lookup is performed in order to retrieve the container reference. Using this, one of the replica containers can be manipulated within a transaction. By means of the installed replication aspects, the updates done on that container are propagated to the other replica containers automatically.

5 Application in an intelligent transportation scenario

Let us consider again the scenario in Figure 1. The safety and traffic information events are collected into the storage system, for example by defining containers for every road segment. The goal of an intelligent dissemination of these events is to deliver them only at the road side unit nodes for which the event is relevant, for instance only for the vehicles located upstream of an accident.

Without going into details about the algorithm to find the RSUs located upstream of an accident event, let us denote with S the set of road segments and their associated containers, and with R the set of road side unit containers.

An event injected into the container $i \in S$ is disseminated by algorithm to a set of RSU node containers R^i. As a consequence the vehicles approaching one of those RSUs (direction dependent) will receive the events from road segment i. The steps in writing such an application are sketched as follows:

- for the container $i \in S$ a so called aspect program (see section 2) is created. The aspect is triggered whenever an entry E is written in the container.

- based on the type/priority of the entry, the triggered aspect program fetches (or calculates) a set $R^i \subset R$ to which the entry E should be copied as follows: for each container j $\in R^i$ do *write (j, timeout, transaction, E)*

6 Conclusion

Distributed hash tables have a few known drawbacks: similarity and locality of data e.g. Meier vs. Meyer cannot be easily modelled, lookup without specifying exactly the searched name or range searches cannot be performed, without special handling in the application. We presented an system approach that largely compensate for these disadvantages. Using shared space containers we can preserve the spatial proximity of the stored events, an important requirement for the management of geo-temporal data. Since the data objects are enriched with keywords and metadata in form of Linda tuples, LIFO / FIFO ordering or data filtering according to any of these tuples can be applied when reading from a container. Most important, manipulation of data in a container does not create DHT signalling traffic since the container links do not change. For retrieval of a number of k entries as in the sketched transportation application, one single DHT lookup is necessary to locate the container, instead of k lookup queries in a plain DHT storage approach.

A further contribution in the paper is a simple solution for the integrated replication of both DHT entries and containers, which in addition exploits the pastry replication mechanism. The PAST system stores immutable files, whereas the proposed approach deals with active container objects in which content is being steadily updated. The location of possible replicas for an DHT entry are dictated by pastry/past getReplicas() function. Similar to PAST, for updating the replicas, the first accessed replica peer is made responsible for the update process (see Figure 2).

Further work will provide performance measurements on a prototype that will show the behaviour of the DHT and space container storage network in presence of node fluctuation (churn).

7 Acknowledgment

This work has been supported by the Austrian Government and by the City of Vienna within the competence center program COMET.

References

[1] Xvsm website, http://www.xvsm.org, 2008.

[2] K. Aberer. P-grid: A self-organizing access structure for p2p information systems. In *CooplS '01: Proceedings of the 9th International Conference on Cooperative Information Systems*, pages 179–194, London, UK, 2001. Springer-Verlag.

[3] M. Balazinska, H. Balakrishnan, and D. Karger. Ins/twine: A scalable peer-to-peer architecture for intentional resource discovery. pages 195–210, 2002.

[4] M. Bravetti, R. Gorrieri, R. Lucchi, and G. Zavattaro. Quantitative information in the tuple space coordination model. *Theor. Comput. Sci.*, 346(1):28–57, 2005.

[5] G. Cabri, L. Leonardi, and F. Zambonelli. Mars: a programmable coordination architecture for mobile agents. *Internet Computing, IEEE*, 4(4):26–35, Jul/Aug 2000.

[6] N. Carriero and D. Gelernter. Linda in context. *Commun. ACM*, 32(4):444–458, 1989.

[7] P. Ciancarini. Distributed programming with logic tuple spaces. *New Gen. Comput.*, 12(3):251–284, 1994.

[8] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with cfs. *SIGOPS Oper. Syst. Rev.*, 35(5):202–215, 2001.

[9] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica. Towards a common api for structured peer-to-peer overlays. In *Peer-to-Peer Systems II*, pages 33–44, 2003.

[10] P. Druschel and A. Rowstron. Past: A large-scale, persistent peer-to-peer storage utility. In *HOTOS '01: Proceedings of the Eighth Workshop on Hot Topics in Operating Systems*, page 75, Washington, DC, USA, 2001. IEEE Computer Society.

[11] eva Kühn, J. Riemer, and G. Joskowicz. Xvsm (extensible virtual shared memory) architecture and application. Technical report, Space-Based Computing Group, Institute of Computer Languages, Vienna University of Technology, November 2005.

[12] E. Freeman, K. Arnold, and S. Hupfer. *JavaSpaces Principles, Patterns, and Practice*. Addison-Wesley Longman Ltd., Essex, UK, UK, 1999.

[13] D. Gelernter. Generative communication in linda. *ACM Trans. Program. Lang. Syst.*, 7(1):80–112, 1985.

[14] D. Gelernter and N. Carriero. Coordination languages and their significance. *Commun. ACM*, 35(2):97–107, 1992.

[15] Gigaspaces. *GigaSpaces Enterprise Application Grid Version 4.1 Documentation*, 2005.

[16] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier, and J. Irwin. Aspect-oriented programming. In *ECOOP'97 Object-Oriented Programming*, pages 220–242. Springer-Verlag, 1997.

[17] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells, and B. Zhao. Oceanstore: an architecture for global-scale persistent storage. *SIGOPS Oper. Syst. Rev.*, 34(5):190–201, 2000.

[18] E. Kühn. *Virtual Shared Memory for Distributed Architecture*. Nova Science Publishers, 2001.

[19] E. Kühn, R. Mordinyi, and C. Schreiber. An extensible space-based coordination approach for modeling complex patterns in large systems. *3rd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation, Special Track on Formal Methods for Analysing and Verifying Very Large Systems*, 2008.

[20] E. Kühn, J. Riemer, R. Mordinyi, and L. Lechner. Integration of xvsm spaces with the web to meet the challenging interaction demands in pervasive scenarios. *Ubiquitous Computing And Communication Journal (UbiCC), special issue on "Coordination in Pervasive Environments"*, 3, 2008.

[21] E. Kühn and F. Schmied. Xl-aof: lightweight aspects for space-based computing. In *AOMD '05: Proceedings of the 1st workshop on Aspect oriented middleware development*, New York, NY, USA, 2005. ACM.

[22] T. J. Lehman, A. Cozzi, Y. Xiong, J. Gottschalk, V. Vasudevan, S. Landis, P. Davis, B. Khavar, and P. Bowman. Hitting the distributed computing sweet spot with tspaces. *Comput. Netw.*, 35(4):457–472, 2001.

[23] Z. Li and M. Parashar. Comet: A scalable coordination space for decentralized distributed environments. In *HOT-P2P '05: Proceedings of the Second International Workshop on Hot Topics in Peer-to-Peer Systems*, pages 104–112, Washington, DC, USA, 2005. IEEE Computer Society.

[24] G. P. Picco, D. Balzarotti, and P. Costa. Lights: a lightweight, customizable tuple space supporting context-aware applications. In *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, pages 413–419, New York, NY, USA, 2005. ACM.

[25] G. P. Picco, A. L. Murphy, and G.-C. Roman. Lime: Linda meets mobility. In *ICSE '99: Proceedings of the 21st international conference on Software engineering*, pages 368–377, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[26] A. Rowstron and P. Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 188–201, New York, NY, USA, 2001. ACM.

[27] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329–350, London, UK, 2001. Springer-Verlag.

[28] Q. Xu, T. Mak, J. Ko, and R. Sengupta. Vehicle-to-vehicle safety messaging in dsrc. In *VANET '04: Proceedings of the 1st ACM international workshop on Vehicular ad hoc networks*, pages 19–28, New York, NY, USA, 2004. ACM.