Journal of Computer and System Sciences ••• (••••) •••–•••

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Using tuple-spaces to manage the storage and dissemination of spatial–temporal content

Sandford Bessler [a,*], Alexander Fischer [b], Eva Kühn [b], Richard Mordinyi [b], Slobodanka Tomic [a]

[a] FTW Telecommunications Research Center Vienna, Donau-City 1, A-1220 Vienna, Austria
[b] Vienna University of Technology, Argentinierstrasse 8, A-1040 Vienna, Austria

ARTICLE INFO

Article history:
Received 7 June 2009
Received in revised form 22 September 2009
Available online xxxx

Keywords:
Tuple spaces
Content centric dissemination
DHT
Intelligent transportation
Service platform

ABSTRACT

Structured, spatial–temporal content arises in application areas such as mobile computing, intelligent transportation, urban mobility, and ubiquitous sensing. For the distributed storage and dissemination of such content, peer-to-peer solutions appear to be the natural choice. However, a closer analysis shows that distributed hash tables (DHT) alone are not enough: firstly, they do not maintain the original data structure needed to efficiently access the comprising attributes, and secondly, they lead to high signaling traffic when the data is short lived, such as in high mobility scenarios. In order to address these two problems we propose a novel content dissemination architecture based on an overlay of space-based containers. Furthermore, we apply the proposed concept to realize a concrete application in the field of intelligent transportation, and present the results of the performance evaluation conducted with the system prototype.

1. Introduction

As wireless access becomes ubiquitous and peer mobility increases, the internet architecture faces many new challenges. Emerging mobility scenarios often require disruption-tolerant behavior, novel opportunistic routing and node-by-node reliable transport, since the assumption of stable end-to-end connections is not true anymore. The future internet architecture is currently developed in projects such as 4WARD [1], FIND-Postcards from the Edge [2], or Content Centric Networking of the internet pioneer Van Jacobson [3] and enables content-centric instead of device-oriented information dissemination. An essential requirement in this architecture is that intermediate nodes must be able to cache large amounts of data. The technology for "storage in the net" has been efficiently addressed by peer-to-peer storage networks such as PAST [4,5], OceanStore [6] and the cooperative file system CFS [7]. However, in case structured content, such as context data with temporal and geo-location attributes needs to be efficiently stored, queried and acted-upon, a storage network that supports mainly files renders unsuitable, since the information structure gets lost. A few P2P techniques such as the intentional naming system INS/Twine [8] or P-Grid [9] can maintain the content structure with a hierarchical DNS-like addressing. In [10] the authors map a multidimensional keyword space originating from Linda tuples to a hash space and use DHT techniques to retrieve the tuples. However, the costs for such solutions are high, since keeping a P2P entry for each data item causes a large traffic overhead, especially when the data elements are mutable and short lived.

In this paper we address the network storage and distribution of structured content by proposing an architecture that unifies the flexibility of the P2P approach with the query expressiveness and coordination support of the tuple space-based

* Corresponding author.
E-mail address: bessler@ftw.at (S. Bessler).

0022-0000/$ – see front matter
doi:10.1016/j.jcss.2010.01.010

Please cite this article in press as: S. Bessler et al., Using tuple-spaces to manage the storage and dissemination of spatial–temporal content, J. Comput. System Sci. (2010), doi:10.1016/j.jcss.2010.01.010

computing paradigm [11]. Supported by a geographical hash function, the addressing of the tuple spaces is content driven and not related to the node identity.

In order to show the benefits from our architecture solution we describe an application that disseminates the content in designated geographical areas. The required support for application-based routing and distributed coordination is offered by a tuple-spaces technology called extended virtual shared memory (XVSM) [12,13].

Our approach has two advantages: firstly, it keeps the message traffic low on the DHT level and secondly, it allows applications to directly manipulate the data entries of fine granularity stored within the space containers.

The rest of the paper is organized as follows. In Section 2 we briefly introduce the two constituent technologies of our solution, that is, distributed hash tables and the space-based computing. In Section 3 we discuss the key features of the proposed solution including addressing and lookup, replication, and handling of node arrivals and departures. A global API that hides implementation aspects from the applications is presented as well. In Section 4 we show, how the proposed architecture is applied to realize a distributed platform for intelligent transportation applications, and in Section 5 we present message throughput and delay measurements. Finally, Section 6 contains concluding remarks.

2. Technologies

The proposed content storage and distribution platform is based on two constituent technologies, which we discuss briefly in this section: the tuple spaces and the distributed hash tables.

2.1. Space-based computing

The idea of a shared data space was first created by David Gelernter in the 1980s [14]. 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 data tuples into the space. Sharing data via spaces [15] is not new; it originates from parallel processing and was later considered for distributed environments. The processes interested in retrieving information from the space perform blocking read (Rd/In) operations specifying data 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. The loose coupling between processes both in time and space reduces complexity in creating distributed applications [16].

There are a lot of implementations that follow the concept of the tuple space and provide associative search for the stored tuples, such as LighTS [17], JavaSpace [18], TSpaces [19], GigaSpace [20], Blitz [21]. We developed a space-based architecture called XVSM (extensible virtual shared memory [12,13]), that generalizes the Linda tuples [22] and some later extensions to one single concept of *shared containers*. A (data) entry in a shared container is a set of labeled values called tags. The payload and meta-data relevant for coordination are differently tagged, allowing to distinguish between user data (message body) and coordination data (message header). Similarly to the Linda model, containers can be accessed by the operations *read*, *take*, and *write*. Each container has one or more coordination types [23] 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 [24] (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 provides support for user defined match makers [17] 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 container is extensible in a sense that its behavior can be extended through "aspects" comparable to the "aspects" in object-oriented programming languages [25] or to space-based programming extensions for aspects [26]. XSVM aspects are code fragments that are executed, when an operation (e.g. read, write, etc.) is carried out on a container. Interception points for aspects are set before or after the operation is executed, e.g. pre-read, post-write, etc. Aspects are used to build higher level logic such as dedicated replication strategies.

2.2. Distributed hash tables

The underlying concept of scalable P2P networks such as Chord, Pastry or CAN is the distributed hash table (DHT). The basic functions of a DHT are to store a data object and to efficiently retrieve that data object from any peer in the network. For this purpose, a very large identifier space (e.g. all binary combinations of 128 bits) is defined and data objects and node identifiers are mapped on it using a hash function. The design options for creating the *key* for a data object include hashing the (string) name of the object, its location (URI), or the content itself. The storage/publishing function *publish(key, value)* is not affected by the selected alternative. A DHT offers basic self-organization functions such as adapting the topology when a node arrives or leaves, as well as a better reliability due to mechanisms to replicate DHT entries. The lookup complexity is for most DHTs $O(\log N)$ where N is the number of nodes. Among the various DHT algorithms, Pastry [27] with its

Please cite this article in press as: S. Bessler et al., Using tuple-spaces to manage the storage and dissemination of spatial–temporal content, J. Comput. System Sci. (2010), doi:10.1016/j.jcss.2010.01.010

implementation FreePastry deserves a more detailed discussion. In FreePastry the identifiers are selected from a 128-bit ID circular space. The routing of messages between nodes is based on maximum prefix matching. The routing table of a node defines its rows, such that the ith row contains those nodes that have a common prefix of length i with the current node ID. The leaf set is also a part of the routing table and contains nodes with node ID numerically close to the current node ID.

DHT entries are replicated in Pastry using a subset of the leaf set nodes. The Pastry replication mechanism guarantees that the number of live replicas r remains invariant under node churn conditions. During a lookup operation, the routing protocol conducts the search for the node with the closest ID to the key K. It is however possible that another node is found, namely a node that is located on the routing path and that holds a replica of K. Therefore, the *lookup(k)* operation might not deliver deterministically the same replica.

3. Architecture and design considerations

3.1. Naming and addressing

As mentioned before, the proposed system aims to keep the structure of information entities, for instance by maintaining grouping and ordering of messages according to certain criteria. This is achieved by putting the information entities in tuple space containers and, at retrieval time, applying filter and ordering selectors. Thus, the addressable entities at the DHT level are not individual content objects, but space containers. This design decision accounts for the requirement to handle structured and ephemeral content in an efficient way. The given addressing scheme for XVSM space containers is a URI with the format *"tcpjava://mycomputer.mydomain.com:1234/ContainerName"*. If *ContainerName* is unique on that host, a remote application can directly address the container using the DNS. However, the fixed identifier-locator (IP address) binding leads to known problems in case the IP address changes (due to mobility of the node), the node fails or if containers are moved to other nodes to better distribute the load.

In order to achieve identifier-locator separation we proceed as follows: we assume that the key–value pair entry in DHT looks like: *key* = *H(containerName)*, *value* = *containerReferenceURI*. The value stored under *containerReferenceURI* has the format "tcpjava://localhost:1234/randID", where localhost is a place holder for the IP address, and randID uniquely identifies the container. As we will see in Section 3.3, the container replicas maintained on other nodes must have the same randID.

When a remote application client requests the value in *H(containerName)* using the DHT overlay, the destination node of that lookup replaces "localhost" in the *containerReferenceURI* with its own IP address, before returning to the requesting client.

3.2. Location aware hash function

In FreePastry, data set placement and routing is done by defining and slicing a 128-bit ID space. A hash function is used to map nodes and entries to these IDs. The default implementation is to utilize the well-known SHA1 hash function to encode the keys of an entry and the random number which is by default assigned to new node to obtain the necessary IDs.

We focus on spatial–temporal content that, in addition to being a time-stamped event, contains a geo-reference to a point, a pair of points or a circular zone. A point is defined by a pair of WGS84 coordinates. In some services it is required that a piece of content is routed to a node responsible for those coordinates.

To realize this kind of functionality, a hash function that takes geographical location of a point into account is superior to the random behavior of standard SHA1 hash function. In order to build the location-aware-key, an identifier called *Cname* is added to the coordinate pair, because the coordinates alone cannot qualify collocated objects. This concept leads to a hash function signature similarly to [28]: $H(x, y, Cname)$, where x is the concatenation of 8 longitude digits $o1, o2, \ldots, o8$, y is the concatenation of the first eight latitude digits $a1, a2, \ldots, a8$ and *Cname* is the identifier mentioned above.

3.3. Resilience considerations

The spectrum of fault tolerance design solutions in distributed systems is large and depends of course on the requirements and the solution costs. A full backup of services following a node failure does not solve the problem if those services are location aware, because, in general, the service functionality cannot be delivered by another node with the same quality. Therefore, by using replication techniques we want to avoid at least the loss of data after a node failure.

At the overlay level, the replication of DHT entries is a common functionality. The Pastry replication algorithm is based on the replication of an object addressed by a key K on (invariant) 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. Our contribution is to connect and synchronize two mechanisms: the replication of DHT entries (DHT replicas) and the replication of shared space containers (container replicas). In Section 3.1 it was stated that the DHT entries were made of container URIs that look 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 also to point to the correct container replica. Pastry can always return the so-called *replicaSet* for a certain key, that is a set of nodes that can be used to store replicas of that key. 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

Please cite this article in press as: S. Bessler et al., Using tuple-spaces to manage the storage and dissemination of spatial–temporal content, J. Comput. System Sci. (2010), doi:10.1016/j.jcss.2010.01.010

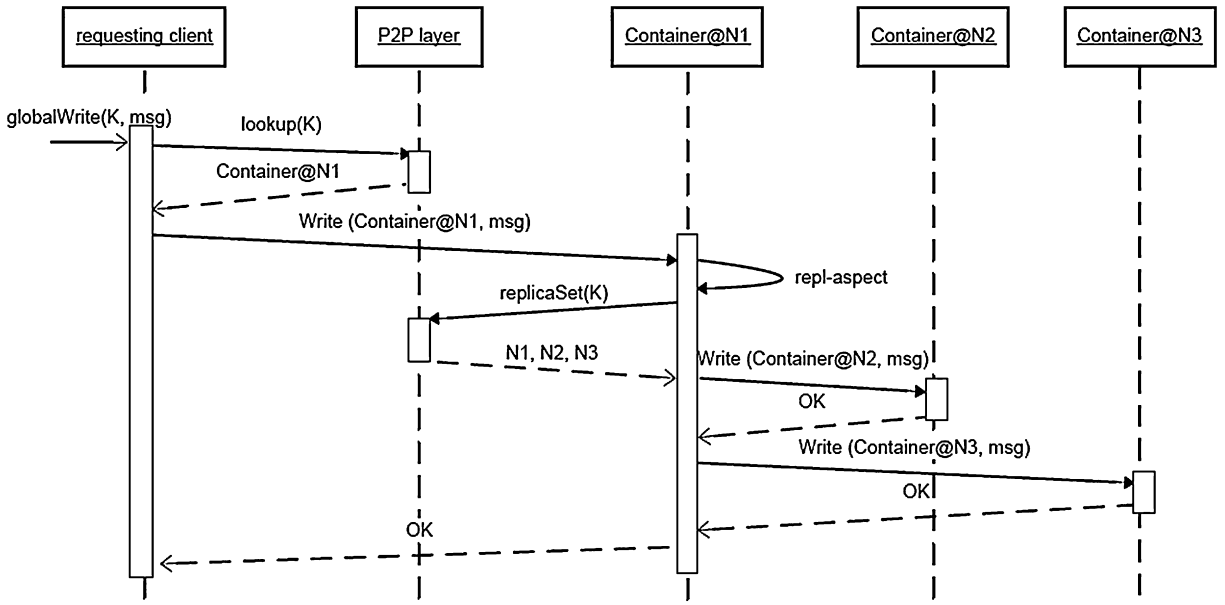

Fig. 1. Handling container replicas.

created by Pastry on those nodes from the set *replicaSet(K)* that have no replica. In addition to the DHT mechanism, the container corresponding to the newly created replica has to be copied as well on that same node.

Containers are active components in which the entries change following operations such as write, take, destroy or shift. 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 to the DHT implementation – in our case – to 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 cannot use a single master replication scheme. We propose to use a delegation pattern as shown in Fig. 1. The first interaction *lookup(K)* will return the full address of a container replica (or the root). The requesting peer proceeds with a direct container *write()* operation, followed by a replication *aspect* program that calls *replicaSet()* in order to obtain the other replica nodes, and repeats the *write* operation at the respective containers. The first accessed container returns the final result of the replication operation.

3.4. Handling join and leave requests

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 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 sets of L neighbors, whereas other nodes are removed from those leaf sets. If a key K points to one of these removed nodes, then only $r-1$ replicas for K can be accessed, and the new node has to acquire a replica for K as well. The overall effect of a "join" is therefore moving the container from the node dropped from the leaf set to the new node.

3.5. Distributed 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 tuple-spaces paradigm offers the features described in Section 2.1, 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 below is a compact, elegant abstraction that needs only the application specific container name for addressing it network-wide. The main methods are given below, whereas the whole framework handles in addition transactions and creation of aspect programs (see [24] for more examples).

- *globalCreate(LocationAwareKey key)*: 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

Please cite this article in press as: S. Bessler et al., Using tuple-spaces to manage the storage and dissemination of spatial–temporal content, J. Comput. System Sci. (2010), doi:10.1016/j.jcss.2010.01.010

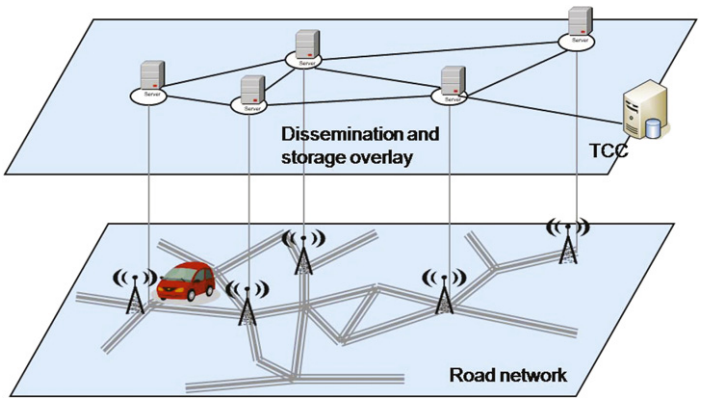

Fig. 2. Intelligent transportation scenario.

replica of it should be placed; 3) containers with the provided name are created on those peers. Additionally, replication aspects are installed.

- *globalRead(LocationAwareKey key, Selector... sel)*: 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.
- *globalWrite(LocationAwareKey key, Entry... entries)*
- *globalTake (LocationAwareKey key, Selector... sel)*
- *globalDestroy(LocationAwareKey key)*
- *globalShift(LocationAwareKey key, Selector... sel)*

For each of these four content modifying 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.

4. Distributed platform for intelligent transportation applications

We evaluate the benefits of the proposed architecture in an intelligent transportation (ITS) scenario, where fast moving vehicles communicate with a fixed, geographically distributed infrastructure, as illustrated in Fig. 2.

The communication between vehicles and infrastructure occurs via the so-called road side units (RSUs), which are installed along the roads. RSUs nodes are interconnected in a mesh wired broadband network and communicate also with vehicles situated in their communication range using wireless short range communication protocols [29]. The exchange of information between vehicles and RSUs is bidirectional: in one type of ITS applications vehicles provide sensor data such as humidity, road temperature, speed, brake status, etc.; other ITS applications distribute relevant traffic events from the road operator towards certain RSUs that further broadcast them to vehicles in the area. These events contain information on incidents, road works, speed limits, etc. In both cases the content is geo-located and its relevance in space and time is limited to a certain region, moving direction and time period.

There has been substantial research in Europe, US and Japan in the field of cooperative systems for intelligent transportation, mainly devoted to wireless car2car communication and routing methods. It has been shown that, by adding infrastructure nodes (infostations [30], beacons, relay nodes, etc.) the delivery ratio [31] is improved and the path length is reduced down to one hop.

Little attention has been paid to the role of infrastructure nodes in intelligent dissemination of content. The closest related project to our approach, COOPERS, is part of e-Safety EU initiative and focusses on infrastructure to vehicle communications. Traffic, weather and safety messages are distributed from a traffic control center to the road side units. With the distributed architecture presented in Section 3 we can do better in the sense of building a self-organizing, scalable and resilient system.

4.1. Service architecture

A whole class of distributed applications that disseminate spatial–temporal content can be realized using our architecture with application components, space containers and a powerful notification framework. In this architecture the content is not sent directly to applications, but to input containers that in the simplest case behave like queues. One or several service applications are notified to retrieve the content, process it, then drop the results into other containers. For mobile nodes, the capability to discover all active services running on one node is important particularly when a service is deployed only on certain nodes. Therefore, the system provides a special container with the role of a service registry. The running services are published in the registry and from there they are announced continuously over a wireless control channel. In Fig. 3 we

Please cite this article in press as: S. Bessler et al., Using tuple-spaces to manage the storage and dissemination of spatial–temporal content, J. Comput. System Sci. (2010), doi:10.1016/j.jcss.2010.01.010

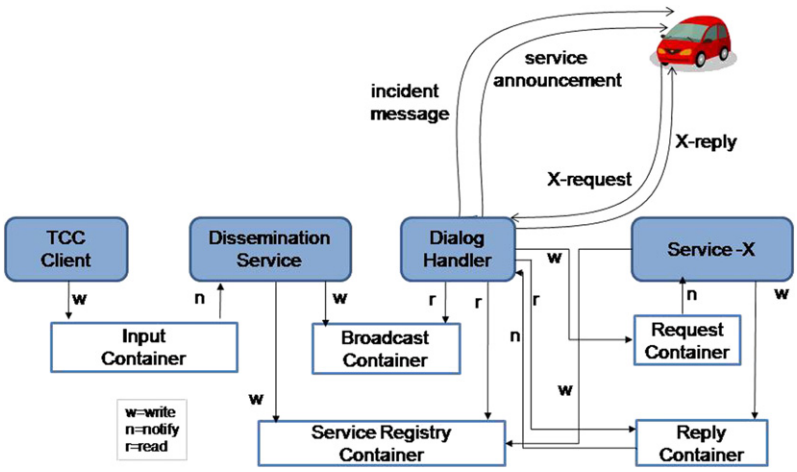

Fig. 3. Space-based service architecture.

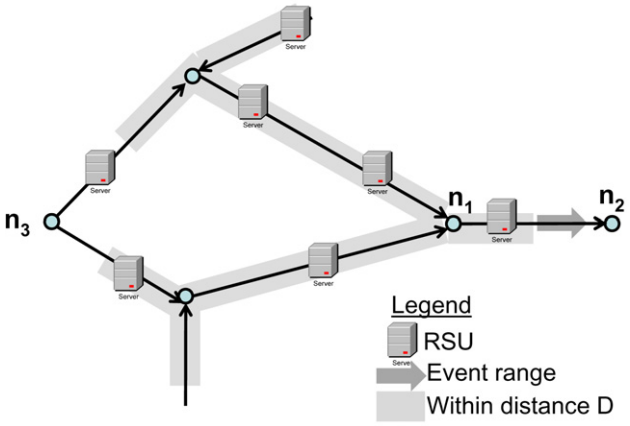

Fig. 4. RSUs at distance D upstream of (n_1).

illustrate this architecture with two services, namely the message dissemination service we will discuss in the next section and an interactive service that is invoked by requests from mobile nodes (vehicles).

With the geographical hashing scheme described in Section 3.2, any content created by some node or injected in the system can easily be routed to a geographically specified region. In case we want to alert the drivers before they arrive to the event location, the geographical routing scheme above is not sufficient: the overlay peers have to be put in relationship with the road network and with the navigation on it. This is achieved by a road network model that contains nodes, links, turning restrictions and the RSU peers with their positions. The network structure is available in each peer node and is used for navigation, shortest paths calculations, and for the advanced routing applications mentioned above.

4.2. Dissemination algorithm

The goal of an intelligent dissemination of traffic and safety messages is to deliver them only at those road side unit nodes for which the event is relevant, for instance only for the vehicles located upstream of an accident. Each event type may have a different dissemination rule, i.e. an accident or incident event message should be disseminated at least 2 km ahead of the accident location, and within 1 minute time, whereas a “speed limit” message from the traffic control center should reach the affected vehicles within 10 seconds.

This service must be available in each RSU node and must calculate for each incoming content message the set of RSUs on which the content shall be replicated. (Note, that replication in this case is completely different from the low-level mechanism discussed to achieve resilience.) Once arrived at the destination RSUs, the message is broadcast to the vehicles passing by.

The algorithm uses navigation along the road network graph $N(V, A)$, where N is the set of nodes and A is the set of directed links (road segments). The content message is mostly associated with “start” and “end” geographical locations given by WGS84 coordinates. Referring to Fig. 4, we have first to find the link (n_1, n_2) on which the event “starts” at a certain location. Such an association requires in general case to apply a map matching procedure. Having identified the link

Please cite this article in press as: S. Bessler et al., Using tuple-spaces to manage the storage and dissemination of spatial–temporal content, J. Comput. System Sci. (2010), doi:10.1016/j.jcss.2010.01.010

(n_1, n_2) and given the dissemination distance D, we proceed as follows (see Fig. 4): the procedure is basically a labeling of the network nodes from n_1 via its incoming arcs, except the arc (n_2, n_1). The labeling occurs if the distance of both end nodes of an arc is smaller than D. In this case all the RSUs on that arc are included in the resulting dissemination set. The border case is, when for a certain arc the distance to one end node is less than D, but to the other end node, it is larger than D. In this case we have to check the RSU list on that arc one-by-one and include those closer than D. The algorithm ends when no edge can be labeled any more. In order to avoid cycles, we make sure that no edge contains the node n_2.

Here it has to be mentioned that if we try to solve the problem using shortest paths to node n_1, the solution will contain a tree. In Fig. 3 two paths exist from a node n_3 towards the destination node n_1. If we apply shortest paths, only one path is selected and labeled. This solution is however wrong, because cars follow individual decisions and would use both paths, therefore ALL the RSUs on these paths have to be included in the resulting dissemination set.

5. Performance analysis

The system has been implemented and deployed on eight nodes in our lab. Our partner, the Austrian highway operator ASFINAG, developed in the COOPERS project an infrastructure-to-vehicle communication solution in which geo-located traffic and safety messages (coded in a markup format called TPEG-ML) are distributed to road side units situated in Tyrol. As mentioned before, the COOPERS system disseminates the messages from a central traffic control center (TCC) directly to the roadside units.

As the nodes (peers) in our test bed have identifiers in the modified hash space corresponding to coordinates on the highways A12 and A13 in Tyrol, the TPEG messages created by the Asfinag Traffic Control Center can be injected and processed on our system as well.

An evaluation of such a system can be performed along the following criteria:

- classical performance indicators such as processing delay and message throughput,
- self-organization feature or management overhead when deploying additional nodes,
- ability to balance the data traffic load,
- data integrity during temporary node failures,
- scalability.

The load sharing aspects are currently studied and mechanisms are in place to distribute service requests on less loaded peers. The self-organization feature is already provided by the P2P underlying layer, but has to be enhanced at the space and application layer. Thus, when a new node joins the network, the road network data structure is updated and replicated to all the nodes.

In the following we look in more detail at the delay and throughput measurements. The messages generated in real time by the traffic control center have been stored in xml files. In order to generate processing load, these messages have been injected at once in the system via several peers. After being parsed, and transformed into an internal tuple format, the messages were time-stamped. After the dissemination and their dispatching to the final destination the messages were time-stamped again, so that the time spent in the system could be measured.

In Fig. 5 1000 messages have been entered into the system creating over 2500 traces in the destination containers (because the dissemination process replicates most of them). Looking at the delay as a function of the release time into the system, we observe at the beginning the effect of queue building. The “patterns” in the figure are cause by the aggregation of delays on different RSUs: some RSUs are slower or have a larger processing load, others are much faster.

The delay histogram in Fig. 6 displays a long tail, which is a result of the aggregation of fast and slow nodes, having different processing loads. A future task in the project will be to explore ways to better share the load between nodes, in particular for those applications that are not constrained to run on a certain node.

As part of the experiments, we measured the total message throughput by counting all the messages that arrive in the destination containers in all nodes, in the same second. (See Fig. 7.)

In the first experiments the message throughput in the system has been limited, against the expectations, by the “route to the nearest node” function. This function dispatches a message entering the system to the node closest with respect to the message coordinates. The function uses heavily the DHT substrate and was meant to shorten the average paths to the RSUs on which the message is to be disseminated. It turned out that this function requires 180 ms on average per message and limits in this way the throughput drastically.

Based on this observation we can discuss the scalability of the proposed architecture in comparison to a plain DHT approach. For the latter, the theoretical lookup complexity is $O(\log N)$ where N is the number of nodes. Since DHT lookup operations are expensive, the more messages are requested at once from a container, the more efficient is the container based architecture. The reason for this is the fact that, once the container is found via DHT, its messages can be accessed directly via one TCP call, see [32] for more details.

The performance absolute numbers are of course modest. This is due primarily to the bottleneck function mentioned above, but also to the java implementation of all components and the low performance of the Pentium 4 machines used.

Please cite this article in press as: S. Bessler et al., Using tuple-spaces to manage the storage and dissemination of spatial–temporal content, J. Comput. System Sci. (2010), doi:10.1016/j.jcss.2010.01.010

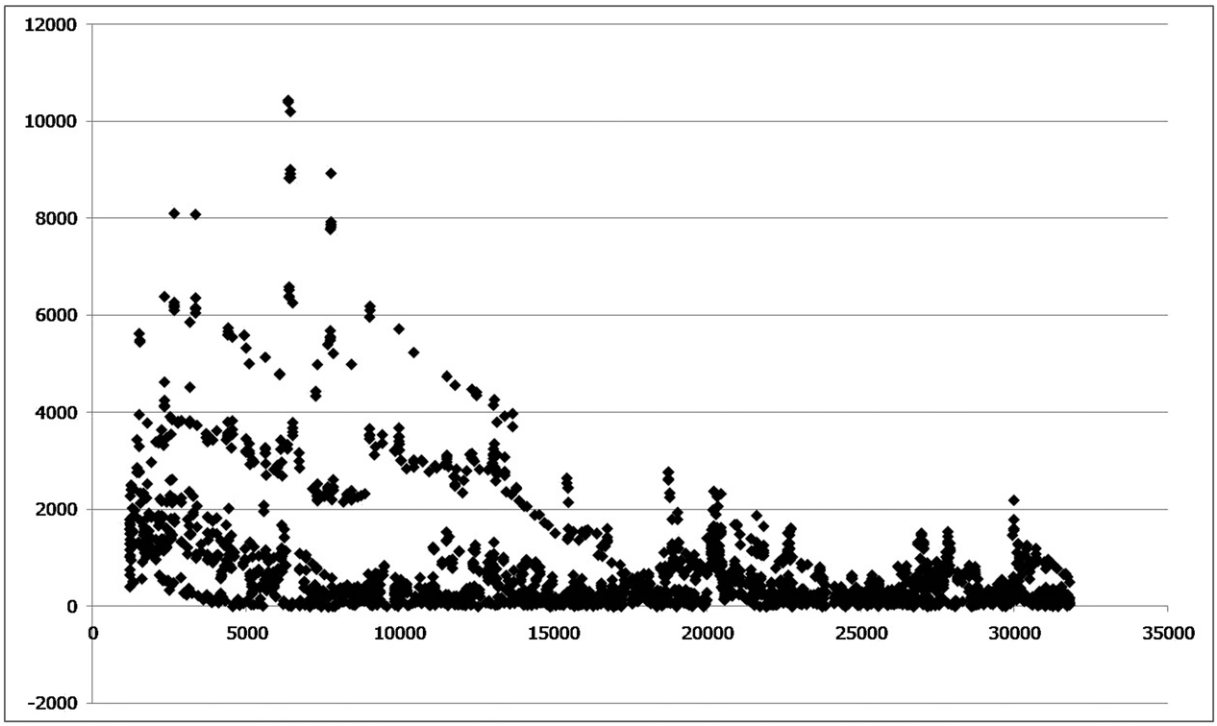

Fig. 5. Time spent in the system versus message entrance time.

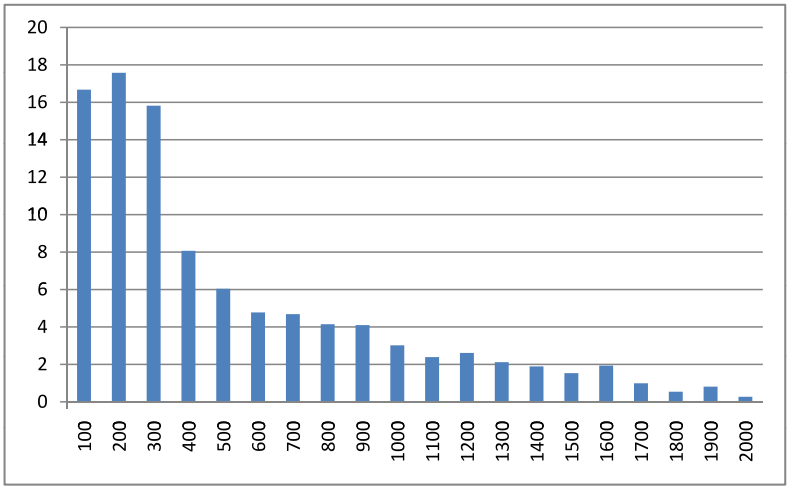

Fig. 6. Histogram of delay [ms].

6. Concluding remarks

In this work we present a content distribution system that combines the self-organization features of distributed hash tables with the powerful processing capabilities of tuple spaces. We consider in particular applications in which the content is volatile and geo-tagged.

Most DHTs have a few known drawbacks: similarity and locality of data (e.g. Meier vs. Meyer) cannot be easily modeled. Neither can lookups be performed, without specifying exactly the searched name. We present a system approach that largely compensates for these disadvantages. Using shared space containers, we are able to handle similarity and locality in storage and search processes, and apply more complex queries to Linda tuples. In general, the manipulation of data in a container is efficient, since it does not change any DHT entries that would create DHT signaling traffic.

The main contributions of this work include:

Please cite this article in press as: S. Bessler et al., Using tuple-spaces to manage the storage and dissemination of spatial–temporal content, J. Comput. System Sci. (2010), doi:10.1016/j.jcss.2010.01.010

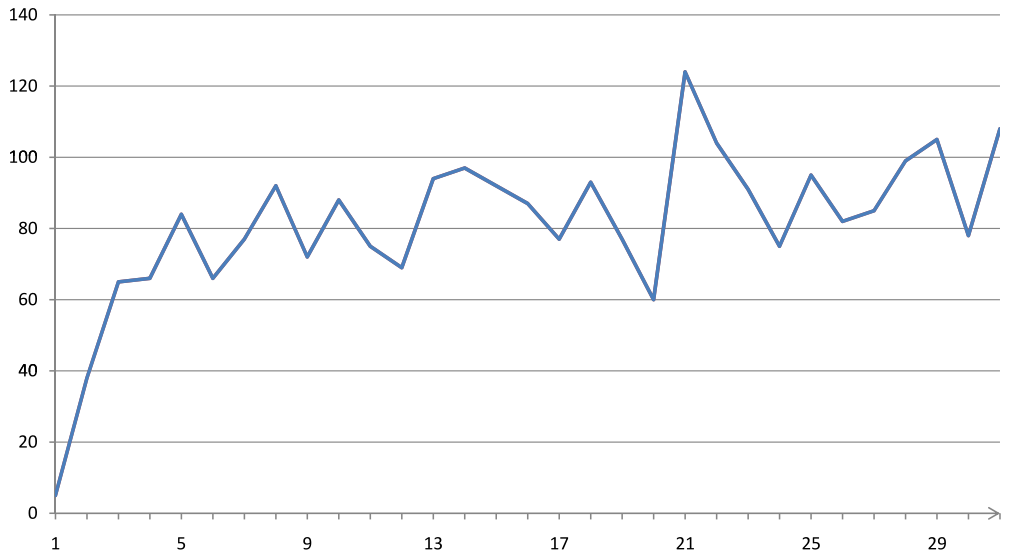

Fig. 7. Total message throughput [msg/s].

- a data centric routing mechanism that allows to store and process a message at the geographically closest node. This operation is however expensive in terms of processing delay;
- a simple solution for the integrated replication of both DHT entries and containers, a solution which however exploits the Pastry DHT replication mechanism;
- a service architecture and an example dissemination service that illustrates the use of such a system in intelligent transportation scenarios.

Further research will be conducted towards better efficiency through load sharing mechanisms between peers and towards advanced application routing algorithms to support the mobility and the temporary connectivity of content sources (the vehicles in our scenario).

Acknowledgments

This work was performed in the project Realsafe and has been supported by the Austrian Government and by the City of Vienna within the competence center program COMET. We thank Peter Meckel and Marco Jandrisits of Asfinag AG for their support.

References

[1] Online document (last visited, 2009-06-03), URL: http://www.4ward-project.eu/.
[2] Online document (last visited, 2009-06-03), URL: http://www.nets-find.net/Funded/Postcards.php.
[3] V. Jacobson, A new way to look at networking (last visited, 2009-06-03), URL: http://video.google.com/videoplay?docid=-6972678839686672840.
[4] P. Druschel, 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, IEEE Computer Society, Washington, DC, USA, 2001, p. 75.
[5] A. Rowstron, 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, ACM, New York, NY, USA, 2001, pp. 188–201, doi: http://doi.acm.org/10.1145/502034.502053.
[6] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells, B. Zhao, OceanStore: An architecture for global-scale persistent storage, SIGARCH Comput. Archit. News 28 (5) (2000) 190–201, doi: http://doi.acm.org/10.1145/378995.379239.
[7] F. Dabek, M.F. Kaashoek, D. Karger, R. Morris, I. Stoica, Wide-area cooperative storage with CFS, SIGOPS Oper. Syst. Rev. 35 (5) (2001) 202–215, doi: http://doi.acm.org/10.1145/502059.502054.
[8] M. Balazinska, H. Balakrishnan, D. Karger, INS/Twine: A scalable peer-to-peer architecture for intentional resource discovery, in: Proc. of the First International Conference on Pervasive Computing, Zurich, Switzerland, in: Lecture Notes in Comput. Sci., vol. 2414, 2002, pp. 195–210.
[9] K. Aberer, P-Grid: A self-organizing access structure for P2P information systems, in: CoopIS'01: Proceedings of the 9th International Conference on Cooperative Information Systems, Springer-Verlag, London, UK, 2001, pp. 179–194.
[10] Z. Li, 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, IEEE Computer Society, Washington, DC, USA, 2005, pp. 104–112, doi: http://dx.doi.org/10.1109/HOT-P2P.2005.7.
[11] P. Ciancarini, Distributed programming with logic tuple spaces, New Gen. Comput. 12 (3) (1994) 251–284.
[12] E. Kühn, J. Riemer, R. Mordinyi, 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.
[13] Online document (last visited, 2009-06-03), URL: http://www.xvsm.org.

Please cite this article in press as: S. Bessler et al., Using tuple-spaces to manage the storage and dissemination of spatial–temporal content, J. Comput. System Sci. (2010), doi:10.1016/j.jcss.2010.01.010

[14] D. Gelernter, Generative communication in Linda, ACM Trans. Program. Lang. Syst. 7 (1) (1985) 80–112, doi: http://doi.acm.org/10.1145/2363.2433.
[15] N. Carriero, D. Gelernter, Linda in context, Commun. ACM 32 (4) (1989) 444–458, doi: http://doi.acm.org/10.1145/63334.63337.
[16] G. Cabri, L. Leonardi, F. Zambonelli, Mars: A programmable coordination architecture for mobile agents, Internet Computing, IEEE Internet. Comput. 4 (4) (2000) 26–35, doi:10.1109/4236.865084.
[17] G.P. Picco, D. Balzarotti, P. Costa, LighTS: A lightweight, customizable tuple space supporting context-aware applications, in: SAC'05: Proceedings of the 2005 ACM Symposium on Applied Computing, ACM, New York, NY, USA, 2005, pp. 413–419, doi: http://doi.acm.org/10.1145/1066677.1066775.
[18] E. Freeman, K. Arnold, S. Hupfer, JavaSpaces Principles, Patterns, and Practice, Addison–Wesley Longman Ltd., Essex, UK, 1999.
[19] T.J. Lehman, A. Cozzi, Y. Xiong, J. Gottschalk, V. Vasudevan, S. Landis, P. Davis, B. Khavar, P. Bowman, Hitting the distributed computing sweet spot with TSpaces, Comput. Networks 35 (4) (2001) 457–472, doi: http://dx.doi.org/10.1016/S1389-1286(00)00178-X.
[20] GigaSpaces enterprise application grid version 4.1 documentation, Tech. Rep., Gigaspaces Technologies, Ltd., 2005.
[21] Blitz javaspaces (last visited, 2009-06-03), URL: http://www.dancres.org/blitz/.
[22] D. Gelernter, N. Carriero, Coordination languages and their significance, Commun. ACM 35 (2) (1992) 97–107, doi: http://doi.acm.org/10.1145/129630.129635.
[23] E. Kühn, E. Mordinyi, L. Keszthelyi, C. Schreiber, Introducing the concept of customizable structured spaces for agent coordination in the production automation domain, in: 8th International Conference on Autonomous Agents and Multiagent Systems, ACM, Budapest, Hungary, 2009.
[24] E. Kühn, R. Mordinyi, C. Schreiber, An extensible space-based coordination approach for modeling complex patterns in large systems, in: 3rd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation, Special Track on Formal Methods for Analysing and Verifying Very Large Systems.
[25] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier, J. Irwin, Aspect-oriented programming, in: ECOOP'97 Object-Oriented Programming, Springer-Verlag, 1997, pp. 220–242.
[26] E. Kühn, F. Schmied, Xl-aof: Lightweight aspects for space-based computing, in: AOMD'05: Proceedings of the 1st Workshop on Aspect Oriented Middleware Development, ACM, New York, NY, USA, 2005, doi: http://doi.acm.org/10.1145/1101560.1101562.
[27] A.I.T. Rowstron, 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, Springer-Verlag, London, UK, 2001, pp. 329–350.
[28] T. Hossfeld, S. Oechsner, K. Tutschku, F.-U. Andersen, L. Caviglione, Supporting vertical handover by using a pastry peer-to-peer overlay network, in: PERCOMW'06: Proceedings of the 4th Annual IEEE International Conference on Pervasive Computing and Communications Workshops, IEEE Computer Society, Washington, DC, USA, 2006, p. 163, doi: http://dx.doi.org/10.1109/PERCOMW.2006.130.
[29] Q. Xu, T. Mak, J. Ko, R. Sengupta, Vehicle-to-vehicle safety messaging in DSRC, in: VANET'04: Proceedings of the 1st ACM International Workshop on Vehicular Ad Hoc Networks, ACM, New York, NY, USA, 2004, pp. 19–28, doi: http://doi.acm.org/10.1145/1023875.1023879.
[30] D. Goodman, J. Borras, N. Mandayam, R. Yates, Infostations: A new system model for data and messaging services, in: Proceedings of IEEE Vehicular Technology Conference, Rome, Italy, 1997, pp. 969–973.
[31] G. Marfia, G. Pau, E.D. Sena, E. Giordano, M. Gerla, Evaluating vehicle network strategies for downtown Portland: Opportunistic infrastructure and the importance of realistic mobility models, in: First International Workshop on Mobile Opportunistic Networking ACM/SIGMOBILE MobiOpp 2007, in Conjunction with MobiSys 2007, Puerto Rico, USA, 2007.
[32] E. Kühn, R. Mordinyi, H. Goiss, S. Bessler, S. Tomic, A p2p network of space containers for efficient management of spatial–temporal data in intelligent transportation scenarios, in: ISPDC'09: Proceedings of the 8th International Symposium on Parallel and Distributed Computing, IEEE, Los Alamitos, CA, USA, 2009, pp. 218–225, doi:10.1109/ISPDC.2009.27.

Please cite this article in press as: S. Bessler et al., Using tuple-spaces to manage the storage and dissemination of spatial–temporal content, J. Comput. System Sci. (2010), doi:10.1016/j.jcss.2010.01.010