SElf-orgaNizing Structures for mAnagemenT In stock Oriented Networks

Aline Carneiro Viana, Nathalie Mitton, Loïc Schmidt, Massimo Vecchio

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

SElf-orgaNizing Structures for mAnagemenT In stock Oriented Networks

Aline Carneiro Viana — Nathalie Mitton — Loïc Schmidt — Massimo Vecchio

N° 7192

Février 2010

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--7192--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
LILLE - NORD EUROPE

SElf-orgaNizing Structures for mAnagemenT In stock Oriented Networks

Aline Carneiro Viana*, Nathalie Mitton†, Loïc Schmidt† , Massimo Vecchio*

Thème : Systèmes et services distribués
Équipes-Projets ASAP et POPS

Rapport de recherche n° 7192 — Février 2010 — 17 pages

Abstract: This paper introduces SENSATION, a novel self-organizing stock management structure. SENSATION is based on a double DHT mechanism and is inspired from existing works such as SOLIST and Tribe. It proposes an efficient unique structure which can be used for different purposes such as data replication, distributed storing and request managements in stock management. These features help in the scalability and reliability of new storing warehouse management which tends to increase in scale and to be more and more interconnected. Results show that SENSATION provides interesting and promising results in terms of reliability and scalability.

Key-words: DHT, RFID, Smart memory placement, EPCGlobal

This work has been partially supported by the FP7 European project ASPIRE (http://fp7-aspire.eu) and the regional project ICOM.

* INRIA Saclay - Île de France
† INRIA Lille - Nord-Europe, Univ. Lille 1, CNRS

Centre de recherche INRIA Lille – Nord Europe
Parc Scientifique de la Haute Borne
40, avenue Halley, 59650 Villeneuve d'Ascq
Téléphone : +33 3 59 57 78 00 — Télécopie : +33 3 59 57 78 50

Infrastructure de réseau auto-configurable pour la gestion des stocks

Résumé : Ce papier introduit SENSATION, une structure auto-configurable pour la gestion de stock. SENSATION utilise un mécanisme de double DHT et s'inspire de travaux existant tels que SOLIST et Tribe. Dans ce rapport, nous proposons une structure unique et efficace pouvant être utilisée pour différents objectifs tels que la réplication des données, l'enregistrement distribué et les requètes pour la gestion des stocks. Ces caractéristiques améliorent le passage à l'échelle et la fiabilité de la gestion d'entrepots dont le nombre tend à grandir et à être de plus en plus interconnectés. Les resultats de SENSATION sont prometteurs en terme de fiabilité et de passage à l'échelle.

Mots-clés : DHT, RFID, Placement mémoire efficace, EPCGlobal

Contents

1 Introduction **4**

2 Case study **5**

3 Preliminaries **6**
3.1 Indirect routing 6
3.2 DHT Mechanisms 6

4 Related works **7**
4.1 SOLIST 7
4.2 Content Addressable Network (*CAN*) 8
4.3 Tribe 9

5 SENSATION **10**
5.1 Solution overview 10
5.2 Mapping of the virtual coordinate system 11
5.3 Example 13

6 Simulation **14**

7 Conclusion **16**

1 Introduction

Nowadays, trade industries are growing up and need more and more the support of technology for managing their goods, for both logistic and traceability concerns. Ordinary applications such as inventories need the support of new technologies to fit the scalability of companies. More and more servers and databases are deployed to store information and connected in networks to become accessible from every place. These databases are now supplied through the reading of bar-codes or RFID tags hold by objects. Shelves equipped by readers even consist in database themselves and can be connected to database network. Even mobile RFID readers can be directly connected to the network and transfers useful data.

Although the networking architecture might be general, it is desirable that the communication infrastructure takes application requirements into account. Here, we narrow our focus to applications of stock management with a large amount of products to be controlled in different geographic sites communicating, and a need for limited communication overhead and reliable data storing. The network should be standard compliant[1] to be connected to an EPC[2] network (*e.g.* which respects the EPC standards).

In order to identify products to be controlled, they are equipped with identifiers (RFID, bar code, *etc.*). To perform the identification, traceability and inventory actions, the network is composed of database servers, routers, and two kinds of readers: either fixed (as a smart shelf) or mobile. Mobile readers can read and store data without being connected to the network, and so transfer data when connected. In this case, routers or fixed readers act as gateways towards databases.

The first motivation is the scalability and reliability of such a network. Indeed, the number of entities can increase heavily and sets the need of large scale communications without overloading the network. Data have to be duplicated for ensuring reliability but in a smart way in order to avoid memory overhead and facilitate at the same time the query routing. Secondly, the different characteristics of stock management requests, coupled with the need of scalability, require a specific network infrastructure management. Different granularities for the different diffusion primitives are needed, such as broadcast (*"Count number of this specific product"*), k-cast (*"Is there a quantity x of this product"*) or anycast (*"Give me the price of this product"*). At last, due to the mobility of some readers, the structure of the network has to organize itself and adapt at every modification in a transparent and local way.

In this paper, we propose SENSATION a reliable k-level structure for distributed redundant memory placement. This structure allows to smartly store data and fast request to be processed by limiting bandwidth overhead. This unique structure allow performing different operations: *(i)* **distributed storing** to allow scalability by reducing memory overhead, *(ii)* **data replication** to ensure data reliability and *(iii)* **efficient request management** for answering requests in a smart scalable way which limit traffic and flooding overhead. This is achieved through the use of two distributed hash tables. The first one allows to reach the correct level, *i.e.* the level corresponding to the specific object

[1] RFID relative standards are established by EPC Global.
[2] http://www.epcglobalinc.org/

targeted by the request while the second one allows to reach the proper correspondent in this layer. For instance, if the request is "What is the price of a blue trouser ?", the first DHT directs the request towards the layer responsible for trousers while the second one then directs the request towards a node aware of the price of blue trousers. Results show that for a low replication level (3), only 25% of data are lost when more than 70% of servers fail. The remaining of this paper is organized as follows. Section 2 describes the case studies we are focusing on. It is worth noting that SENSATION may find several applications in stock management but we describe a specific case study in order to illustrate more easyly the behavior of SENSATION. Section 3 sets background works useful for the understanding of SENSATION. Section 4 describes the relative works useful in the construction of SENSATION while Section 5 explains in details how we use them in SENSATION. Section 6 presents SENSATION performance results. At last Section 7 concludes that work by prospecting some future works.

2 Case study

Our case study is represented by a wide area stock management application. It can be, for instance, a distributor with several warehouses spread all over one or more countries; in each warehouse the servers and the databases are supposed to be connected, thus forming a network. Such an organization may need several tools at different levels.

In order to have a glimpse on these different tools, let us firstly consider an user located in one warehouse: he may need to draw an inventory *(i)* of the whole set of all warehouses, or *(ii)* of a particular warehouse located in a specific area, or evenly *(iii)* of products laying in a small area of a specific warehouse. Moreover, the inventory may target either *(i)* a kind of product (*e.g. "inventory of every trouser"*), or *(ii)* products in a specific place (*e.g. "inventory of everything located in Paris"*), or *(iii)* even both (*e.g. "inventory of every trouser located in Paris"*). On its side, the application has to handle different granularity requests. Finally, requests can be of different natures, possibly requiring different mechanisms for their diffusion: consider, for instance, *(i)* broadcast queries (*"how many items of the specific product do I have?"*), *(ii)* k-cast queries (*"do I have at least k items of this product?"*), and *(iii)* anycast queries (*"give me the price of this product"*).

With a plain database network infrastructure, in all the abovementioned situations, every reader and/or database has to be queried (i.e. the network is flooded). Upon answers, some filters may be applied and/or data are aggregated. This normally introduces a non–negligible latency and causes a network overload, which actually is not necessary. Indeed, *(i)* for broadcast queries, only databases storing information about the articles being enumerated should be contacted; *(ii)* a k-cast query should be delivered as soon as k products have been found; finally, *(iii)* an unicast query may be answered after having contacted only one server. Essentially, these are the features we are to give SENSATION, in order to ensure scalability and high quality of service.

A second, equally important, concern in this case study is enabling a tunable fault–tolerance level. In order to add some reliability to data, it is necessary to include a degree of redundancy (i.e. provide one or more alternate servers

in which to store the same data). The main issues here are to decide where, how and how many times to replicate data. Indeed, in most cases, data are simply replicated in one or more backup servers, which are eventually used in the case that the primary server is experimenting a failure. On the other hand, SENSATION replicates efficiently data and takes advantage of this replication by routing requests towards the closest server storing the information thus decreasing the routing overhead.

3 Preliminaries

3.1 Indirect routing

Providing a scalable and efficient location service in the context of self-organizing systems is a non–trivial problem, due to the spontaneity of networks. This requires a dynamic association between identification and location of a node, and the specification of a mechanism to manage this association. Furthermore, there is the need for minimizing the control message overhead for routing or location discovery. An efficient solution is to perform an *indirect routing* [3, 4, 7].

A routing operation is referred as *indirect* when it is performed in two steps: *(i)* first locate the target and then *(ii)* communicate with the target. This allows the network to decouple the location of a node from the location itself. With this approach, the information can be totally distributed, which is important for achieving scalability in large scale networks.

Distributed Hash Tables (DHT) represent the basis of indirect routing. Basically, they provide a general mapping between any information and a location establishing then a location-independent routing layer. They use a virtual addressing space $\mathcal{V}$. Partitions of this virtual space are assigned to nodes in the network. As a basic example, consider Fig. 1 where the addressing space $\mathcal{V} = [0, 30[$ is shared among 3 nodes i, j, k. In the figure, node i is assigned partition $q(i) = [20, 30[$, node j gets partition $q(j) = [10, 20[$ and node k is responsible for $q(k) = [0, 10[$. The idea is to use a *hash* function to first distribute node location information among rendezvous points. This same *hash* function is known by every node. Each information is *hashed* into a key ($hash(v) = key_v \in \mathcal{V}$) of this virtual addressing space $\mathcal{V}$ and then stored in the node(s) responsible for the partition of the virtual space. Yet, in Fig. 1(a), node k has a content C to register. By applying $hash(C) = 25$ it gets the key of C, which is within the partition of node i. So, node k registers C in node i. Usually, node k also stores the location of C (i.e. it registers its own address). Later, when a node j is looking for content C, it applies the same *hash* function, obtaining the same result (i.e. $hash(C) = 25$) thus knowing that it has to contact node i (cf. Fig. 1(b)). Upon the request, node i answers node j with the information previously registered (Fig. 1(c)). If this information is the location of C, then node j gets the address of node k and can then directly communicate with k.

3.2 DHT Mechanisms

Node Arrival: When a node u enters the network, it has to join the DHT overlay. To do so, u retrieves the way to contact a DHT node v that is used as an entry point to the DHT. Then, u is assigned a partition in the logical

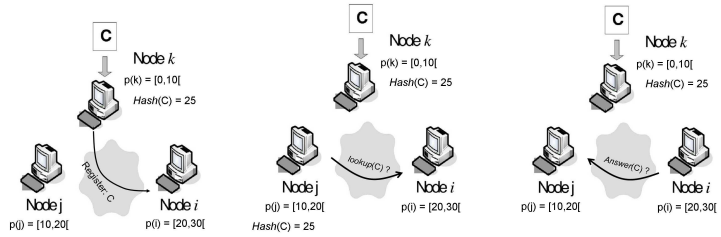


Figure 1: (a) Node k stores content C, and registers information about C on node i which is its rendezvous node. (b) Lookup phase of node j to contact the rendezvous node of C. (c) Lookup answer with information about C.

address space[3]. Routing information in the system is then updated to reflect the presence of u. Finally, u retrieves all $(key, value)$ pairs under its responsibility from the node that stored them previously.

Node Failure: When a node fails, the application data that it used to store is lost unless the DHT uses replication to keep multiple copies on different nodes. Some DHTs follow the simpler soft–state approach which does not guarantee persistence of data. Data items are pruned from the DHT unless the application refreshes them periodically. Therefore, a node failure leads to a temporary loss of application data until the data is refreshed.

Node Departure: DHT implementations often require departing nodes to notify the system before leaving. This allows other nodes to copy application data from the leaving node and to immediately update their routing information leading to improved routing efficiency.

4 Related works

Because of page restrictions, we only focus on literature works useful to SENSATION.

4.1 SOLIST

SOLIST [2] presents a structured overlay network and provides an efficient $*$-cast suite for wireless sensor networks. SOLIST identifies a set of basics functionalities widely used in distributed applications and propose an efficient implementation of this suite in a multi-layer structured network. The aim here is to give a type to each sensor and to group them depending on this type either static (heterogeneous sensors) or dynamic (battery state, sensor data, *etc.*). A layer is composed by all sensors of the same type. Fig. 2 shows a SOLIST multi-layer projection. To be included in the overlay of the proper type t it belongs to, or to perform a query about type t, a given sensor has to contact at least one node from the corresponding layer. To find out such a close node, SOLIST proceeds in two steps. The first step consists in contacting an *entry point* for

[3]Depending on the DHT implementation, u may choose arbitrarily partitions on its own or it determines one based on the current state of the system.

this layer. The entry point will not necessary belong to the correct layer but is aware of the closest sensor that belongs to that layer. The second step thus consists in contacting that *contact node*. Proceeding in two steps allows always reaching the closest node thus limiting energy, bandwidth and time spendings. To identify *entry points* and *contact nodes*, SOLIST uses hash functions.

4.2 Content Addressable Network (*CAN*)

CAN [5] introduces the notion of multi-dimensional identifier spaces by which routing efficiency is greatly improved compared to linear neighbor traversal in a single dimension. The average path length in a system with n nodes and d dimensions scales as $O(d(n^{\frac{1}{d}}))$. **Partition assignment:** In CAN, each data item is assigned an identifier of the form $< x, y, z >$ for $d = 3$. Each node is said to own a zone. CAN ensures that the entire space is divided into non-overlapping zones.Because a key represents a point P in the identifier space, $(key, value)$ pairs are stored on the node owning the zone which covers P. A new node n joining a CAN system sends a Join message to node d which is the current node responsible for the zone where n lays. Then, d splits its zone in half and assigns one half to n (cf. Fig. 3(b)). Finally, d transfers the keys to n for which it has become responsible. **Routing over the CAN overlay:** For routing purposes, a CAN node stores information only about its immediate neighbors. Two nodes in a d-dimensional space are considered neighbors if their coordinates overlap in one dimension and are adjacent to each other in $d-1$ dimension. For instance, in Fig. 3(a), nodes $N1$ and $N6$ are neighbors unlike nodes $N5$ and $N6$ which are not neighbors. If the local node does not own the zone of the destination, it forwards the message to its neighbor with the coordinates closest to the destination (Fig. 3(a)). In a d-dimensional space equally partitioned into n zones, this procedure results in an average of $O((d/4)(n^{\frac{1}{d}}))$ routing steps. This expresses that increasing the number of dimensions significantly reduces the average route length.
Node Failure: The zones of failing or leaving nodes must be taken over by alive nodes to maintain a valid partitioning of the identifier space. A node detects the failure of a neighbor when it ceases to send update messages. Through an efficient message advertising, the neighboring node with the smallest zone

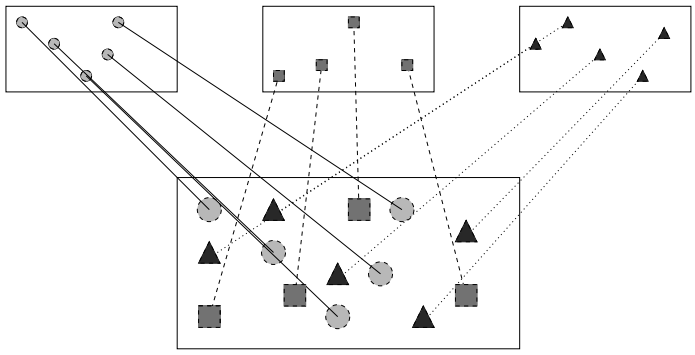
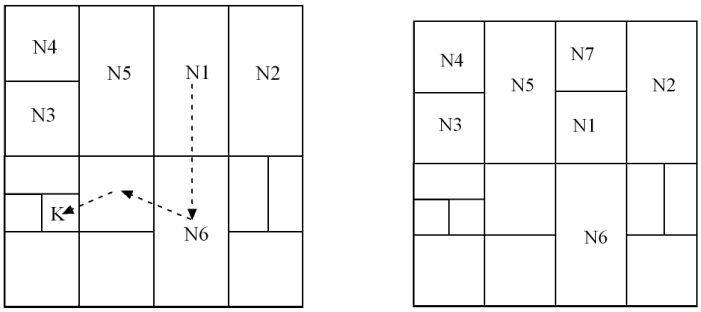


Figure 2: Projection of 3 groups of nodes (▲, ■ and •) into three layers.



(a) Route from node $N1$ to a key K with coordinates (x, y) in a two-dimensional CAN topology. Neighbor set of $N1$: $N2, N6, N5$.

(b) New node $N7$ arrives in $N1$'s zone. $N1$ shares its with $N7$. Updated neighbor set of $N1$: $N7, N2, N6, N5$.

Figure 3: CAN mechanisms illustration.

volume merges the deserted zone with its own zone if possible. Alternatively, it temporarily manages both zones.

Node Departure: When a node l deliberately leaves a CAN system, it notifies a neighbor n whose zone can be merged with l's zone. If no such neighbor exists, l chooses the neighbor with the smallest zone volume. It then copies the contents of its hash table to the selected node so this data remains available.

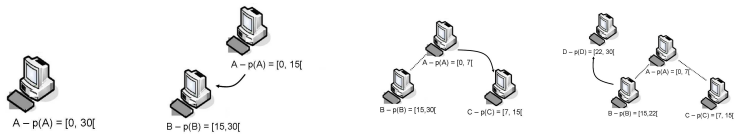
4.3 Tribe

Tribe [6] has been designed for large scale self-organizing networks. Without any central control entity or positioning mechanism, Tribe creates a topology that is a logical network representation, and describes the relative location of the nodes according to their neighborhood in the physical network.

Partition assignment: When a node arrives in the network, it receives a control region which will serve for two purposes: node identification and routing. A new node u in the network receives the control region from its neighbors which control region is the largest. This latter one gives half part of its region to the new node v and becomes its parent node. Tribe thus builds a tree as illustrated by links between nodes on Fig. 4.

Routing in the Tribe structure: The routing operation needs to be performed to reach a node responsible for a given key returned by the hash function, *i.e.* either when a node needs to register an information or to find it. Let $p(u) = [p_u^{\ominus}, p_u^{\oplus}[$ be the control region of node u. When node u entered the network, it has been assigned the virtual space $p_{init}(u) = [p_u^{\ominus}, p_{u_{init}}^{\oplus}[$ where $p_u^{\oplus} \leq p_{u_{init}}^{\oplus}$ since from then, u may have shared its initial space with its children. When node u needs to reach the node responsible for a key $key \in \mathcal{V}$, it proceeds as follows.

- If $key \in p(u)$, u can answer the request.
- If $key \notin p_{init}(u)$, then forwards to its parent.



(a) (b) (c) (d)

Figure 4: Tribe virtual space sharing. (a) node A is alone and responsible for the whole virtual space $\mathcal{V} = [0, 30[$. (b) Node B pops up (b) and gets half of A partition. (c) Node C appears, A gives again half of its partition. (d) Node D arrives, B shares its partition since it has a larger partition than node A.

- If $p_u^{\oplus} < key \leq p_{u_{init}}^{\oplus}$, then forwards to its child v such that $key \in p_{init}(v)$.

Every node reiterates this process till reaching the node responsible for the key.

5 SENSATION

5.1 Solution overview

SENSATION aims at proposing a self-organizing structure to manage data dynamism in a stock oriented network deployed over several geographic sites. It aims at answering stock management requests which may concern either a product, or a family of products or a geographical site.

To do so, in SENSATION, data are assigned a *type* like in SOLIST [2]. This type may be a product kind (*e.g.* type = trousers) or a location (*e.g.* type = Warehouse A) all addressed with no distinctions. Each type is associated to a layer. To address requests concerning a family of products or a larger geographical area, requests will be sent to a collection of layers. Indeed, if a request aims at drawing an inventory of sport articles and that sport articles family includes bikes and rackets, independent requests are sent to the racket layer and to the bike layer and aggregated afterwards. Similarly, if a request concerns every warehouse in Lille and that there are 2 warehouses in Lille, independent requests are sent to warehouse A layer and warehouse B layer.

Each layer t is composed of an overlay network composed of nodes and links. Nodes are databases and servers detaining information relative to type t, *e.g* every server or database storing data about trousers. When joining the overlay network of a layer t, a database A will be assigned a virtual space partition together with "neighbors", *e.g.* other database from the layer to which A is connected to. Note that since this is a logical network, this link may not be direct in the physical underlying network. SENSATION uses Tribe [6] to assign the space partition to databases and to create links. Although Tribe may be directly applied within a layer, the setting up of the SOLIST-like architecture needs some adaptation. In SENSATION, this first basis space needs to be defined and databases need to detain coordinates which allow locating them in this space. We use CAN [1] in this purpose.
The structure is then used for three purposes: *(i)* **Distributed storing:** every database which has fresh data of a given type t, firstly retrieves the corresponding responsible node for the type t. Through the responsible node it can enter

the Tribe t-layer, in order to reach the database responsible for storing its data, by using Tribe routing; *(ii)* **Replication:** any level of replication can be imposed to SENSATION. Indeed, once the Tribe layer is reached, just applying different hash functions to the item ID to store would change the server responsible for the storing. *(iii)* **Efficient request management:** thanks to the use of several layers, a request will not be flooded in the whole network but only on nodes concerned by the request. Resources and latency are saved. The first step which consists in reaching the proper layer allows this node selection. Then, the use of Tribe within an overlay facilitates the request management. Indeed, the tree-like structure of Tribe is useful to aggregate counting requests in each level of the tree and stop flooding in enough nodes have been reached.

5.2 Mapping of the virtual coordinate system

The virtual structure has to include all nodes of all sites of the stock oriented network. Considering a national scale, sites represent the cities where a store is located. Thus, the lookup structure will be divided in NC *cells*, being each cell corresponding to a site.

In the following, each cell (or city) is assigned a coordinate (X_C, Y_C) representing its relative position in the virtual system, along the x and y dimensions. More in details, the cell located in the bottom left corner of the grid has a coordinate equal to $(0, 0)$, the next ones have coordinates equal to $(1, 0)$ and $(0, 1)$ along the x and y dimensions respectively, and so on. Finally, each cell is logically divided into T square areas, where T is maximum number of categories (which as mentioned could be types of goods or sites) supported by the application, and should be a perfect square number. Each square represents a virtual location in the cell, with relative coordinate (X_T, Y_T) within a cell; each location acts as an entry point for a different type in the logical cell (square of coordinates (X_T, Y_T) in cell i and square of coordinates (X_T, Y_T) in cell j are entry points for the same layer(s)). Fig. 5 clarifies this concept; here the geographical area is divided into 3×3 virtual cells and each cell is divided into 16 sites: each site in the cell represents a different entry point for the 16 managed types: this means that the application can manage up to 16 types, being each type represented by an integer value within 0 and 15. For the sake of simplicity in the following we will consider virtual grids divided into the same number of cells along the x and y dimensions. This assumption does not limit the applicability of the approach, since one can build a grid with some unused cells.

By construction, we have exactly T entry-points within each cell, one for each type managed by the application.[4] This means that whenever an action has to be performed (insertion, lookup, ...) on an item of type p, a requesting user u has firstly to reach the p entry point within the cell he belongs to. By contacting the server database which is responsible of the requested type p (which actually is the server responsible of p entry point), u can reach the layer p (in the following we will say that "u is in the layer p"), so he can reach the server which has actually stored the item, by using Tribe. The critic point is how to share the virtual space represented by the entry-point set. We simply

[4]Note that there may be less than T entry points. If a position is empty, the closest server handles the role of the entry point for this empty position.

propose to share the entry-points set among the servers in a cell, by using CAN. For this reason, at a steady state the servers in a cell share the entry-points so as each of them is responsible of an non-overlapping partition of the DHT, being the DHT the entry-point set[5]. For this, we use a bi–dimensional congruential mapping function in order to determine the entry point of type p in the cell (X_C, Y_C) (in the following it will be denoted as $ep_p^{(X_C,Y_C)}$), which is:

$$\begin{aligned} ep_p^{(X_C,Y_C)} &= \left(Xep_p^{(X_C,Y_C)}, Yep_p^{(X_C,Y_C)}\right) \\ &= \left(mod(p,d) + X_C \cdot d, \left\lfloor \frac{p}{d} \right\rfloor + Y_C \cdot d\right) \end{aligned} \tag{1}$$

where $d = \sqrt{T}$, $\left\lfloor \frac{p}{d} \right\rfloor$ is the integer division between p and d and $mod(p,d)$ is the remainder after this division. Finally, once $ep_p^{(X_C,Y_C)}$ is computed, the request can be routed (using CAN) towards the requested entry point p. Once reached the right layer, Tribe is used for finding the node storing the item.

Virtual coordinate assignment: To be inserted in the structure, databases needs coordinates which are assigned by the user who arbitrarily assigns to each new node (database) a unique geographic position in the grid. On the basis on the assigned site, a unique ID in the virtual system is generated as a bit sequence containing information about the virtual cell coordinates (X_C, Y_C) and the relative position within the cell (X_T, Y_T):

$$ID = |bin(X_C, nbC)|bin(Y_C, nbC)|bin(X_T, nbT)|bin(Y_T, nbT)|$$

where $bin(x,n)$ is the binary integer representation of x over n bits, $nbC = \frac{\lceil \log_2(NC) \rceil}{2}$, $nbT = \frac{\lceil \log_2(T) \rceil}{2}$ and $|$ represents a string concatenation. To better understand this translation from a virtual site into a unique address, let us consider an example in which a user plugs a new database to the network in the geographical coordinate $(7, 11)$ of a grid of $NC = 9$ cells. If $T = 16$, the resulting virtual ID is 8-bits long: 2 bits being used for codifying each field. From Fig. 5 it is clear that coordinate $(7, 11)$ belongs to cell $(1, 2)$ and to square $3, 3$ in that cell. Thus, $ID = 01|10|11|11$ (01, being the binary representation of X_C, 10 the binary representation of Y_C and 11 the binary representation of X_T and Y_T, on 2 bits each).

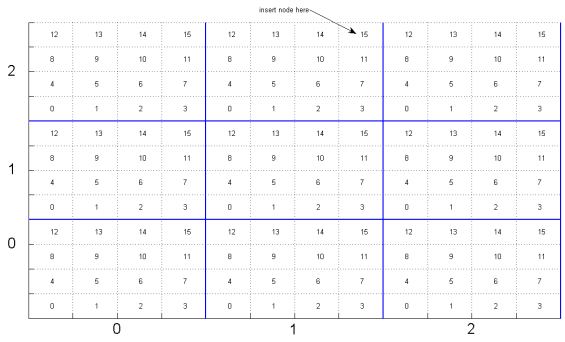


Figure 5: Adding a node. Cell coordinates are reported in the bottom and left sides of the x and y dimensions, resp.

[5]This is one of the main differences between SOLIST and SENSATION.

Adding a server: When a server u wants to join the infrastructure it has to *(i)* obtain a portion of the entry-point set (i.e. joining CAN) and *(ii)* obtain a portion of the Tribe addressing space in all the layers obtained by CAN and corresponding to the objects it has to register (i.e. join Tribe). To better clarify this, let us consider a server u wanting to join the infrastructure. The nearest CAN node v already in the structure has to give u a portion of the entry-points it currently manages. Let consider, for instance that u receives an entry-points set $P = 1, 2, 7, 8$. This means that u will be the entry point for any request of access layers $1, 2, 7, 8$ within the cell, so that u has also to join the same layers in Tribe. In other words, it has to obtain a portion of the Tribe addressing space in each of the mentioned layers.
Adding items: When an item of type i has to be stored in the infrastructure, the server which is currently managing the entry-point i is contacted by using CAN (note that this operation is purely local, in the sense that the responsible node of type i is local to the cell). Since this server is also registered in the Tribe i-layer, the request can be forwarded using Tribe. Notice that, when the right Tribe layer is reached, any level of replication of the data can be imposed, in order to increase the fault–tolerance. Just applying different hash functions to same item ID, in fact, would result in a different server managing the hashed ID (and so the storing server). The only constrain to impose to have a level of replication of h is that the hash functions set h has to be known to all servers. With this technique a tunable fault tolerance level can be introduced. Moreover, this technique let the application further diminish the overhead cost introduced by Tribe, when querying. This aspect will be clearer in the next paragraph.
Lookup phase: When a query has to be answered, firstly the responsible node of the requested type is contacted, by using CAN. This phase is common to the adding phase. Once in the right layer, in dependence on the type of query a different scheme is used. In particular, for unicast, the request is routed using Tribe until the server managing the requested hashed ID is reached. This server is the responsible of directly providing the answer to the asking user. When a replication level h is imposed, we know that actually the item (if it was added) was stored h times in the layer. For this reason, it is less expensive (from a latency and communication overhead point of view) computing the hash function h times and reach the tribe node at the minimum distance from the contact node. This nice feature of SENSATION is analyzed in the simulation experiments.

5.3 Example

Let us assume that a database B wishes to draw an inventory of trousers. To respond to it, by running SENSATION, B first contacts the closest entry point of the type 'trousers'. To identify this latter one, B applies Eq. 1 and CAN (which returns the relative location of the proper entry point corresponding to trousers layer in every cell). It determines the closest one ep_t. Then, B contacts ep_t and gets the location of the nearest node C managing trousers, called here *contact node* (Fig. 6). Note that ep_t knows node C since this latter one has registered at node ep_t by also using Eq. 1. The counting request is then sent to C. Since C has previously joined the layer corresponding to trousers, it can forward the message to all nodes managing trousers that have previously registered in this layer. The counting request will be sent in broadcast mode in the overlay and the answer will contain the number of trousers in the entire network. Another request is to know if a product is existent (here, trousers). As previously, node B contacts the closest entry point for the type 'trousers'. If the entry point returns a node id, the existence is proved. Otherwise, the entry point returns NOT FOUND and there are no trousers.

If a user wants to know if there are at least 10 trousers to ensure a command, he still contacts the closest entry point, which in turn sends the nearest contact node C

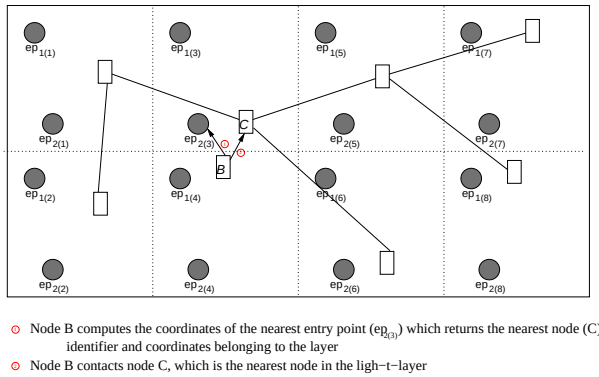


Figure 6: Node B retrieving the nearest node of LIGH-*trousers*-LAYER.

identity. User sends a $k = 10$-cast request to C, which forwards the request in the overlay, in decreasing k with the number of trousers it manages, to the next node of the same layer according to the broadcast algorithm. This next node does the same thing (decrease k and forward the request with the new k) until $k = 0$. When the user receives an answer, he knows if there are enough trousers without flooding the whole network.

A query may also concern a geographical location. In such a case, the same mechanism is applied on the type 'Lille' if the query wishes to set an inventory of Lille warehouse, for instance.

6 Simulation

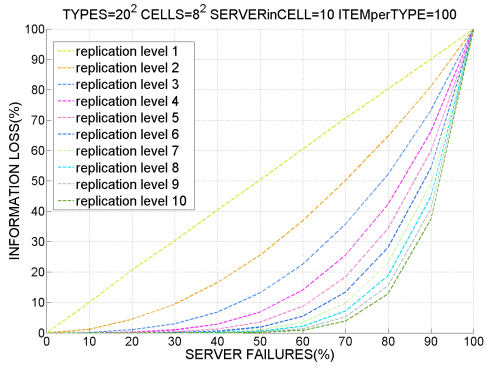
An exhaustive performance analysis is performed to analyze the routing cost of requests, the robustness at various replication levels and the distribution of the information in SENSATION.

Testbed: In each cell 10 servers are uniformly randomly placed (so a total of 640 servers) is a square universe divided into 64 cells of equal size over the Matlab[6] software. After having inserted the servers, items are added in the following way: 100 items of type i are inserted from randomly extracted servers, where $i \in [1, 400]$. Each experiment was performed 30 times and results were averaged.

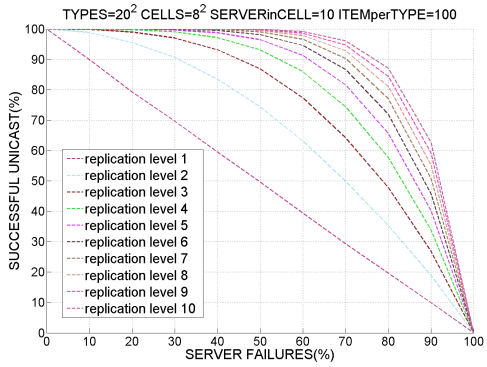
Overhead: After the registration phase, each registered item is queried by a server. This action (which is actually what we have denoted as *unicast query*) in a classical database network should not have any cost, since the information is stored in the local server. On the other side, using SENSATION, the entry point of the requested type has to be contacted (routing with CAN); once the responsible node is reached, Tribe is used to locate the server which is actually storing the requested item. Fig. 8 shows the overhead introduced by CAN for reaching the requested entry points. For the sake of readability a histogram is shown: it can be easily seen that more than 35% of the entry points can be contacted with a routing cost of 2 hops, which is a very low overhead. Note that this overhead is independent on the replication level introduced in Tribe layers.

Dealing with node failures: To evaluate SENSATION robustness, a percentage of failing servers varying within 0 and 100% is considered. Note that, when a server is experimenting a failure, the items registered there are unavailable (it is a database

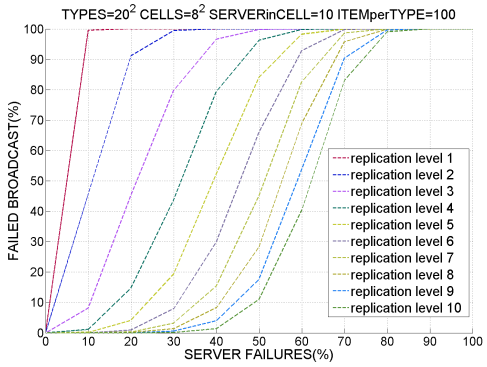
[6] http://www.mathworks.fr/



(a) Percentage of lost information.



(b) successful unicast queries.



(c) Percentage of failing broadcast queries.

Figure 7: Performance of SENSATION varying the replication level in Tribe and the percentage of failing servers.

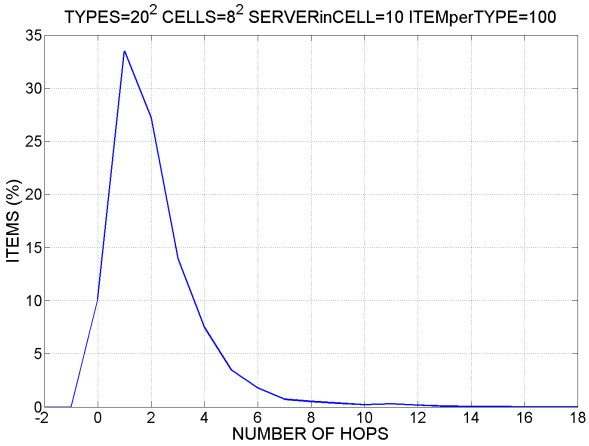


Figure 8: Overhead in CAN.

failure), while the routing in the layers is still possible. It means that during the failure the information is not available, but CAN and Tribe can react to the failure and auto-organize again. Fig. 7(a) shows, for different replication levels and different loss regimes the percentage of information lost (this is equivalent of showing the failure rate of unicast queries). It can be noticed that, with a level of replication equal to 1 the loss of information is linear. Indeed, the information is uniformly distributed as the number of failures. Moreover, as expected, increasing the level of replication in the Tribe layers increases the reliability of SENSATION. With a low replication level equal to 2, when 50% of servers are down, only 25% of data are lost. This same amount of losses is even achieved when 70% of servers are down with a replication level equal to 3. In addition, this is worth noting that this reflects the worst scenario since in these simulations, failures appear randomly while in realistic scenario, when a server fails, its closest servers are more likely to fail rather than other ones in the network. Since data are spread randomly, in realistic scenario, data losses will indeed be less numerous.

Request performance: Fig. 7(b) plots the percentage of successful unicast queries, at different replication levels and loss regimes. Results show that, unicast queries are pretty well managed by SENSATION with a sufficiently low order of replication and even at relatively high loss regime. This has a simple explication: let us assume a replication level of k. Information about an item is lost if and only if the k servers in which the item was stored are simultaneously experimenting a failure; otherwise, after the re-organization the item is registered again. Lost are only temporary. Unfortunately, this behavior is not to be expected in broadcast queries. Indeed, a broadcast query, in fact, can be correctly served (i.e. produce the right answer) if and only if all the items of the requested type are present in any of the k servers in which they were registered. As a consequence, just the disappearing of an item of the given type (the item on which the unicast query is failing) produces a failure of the broadcast query. Nevertheless, as shown by Fig. 7(c), SENSATION well resists to these failures.

7 Conclusion

We have proposed SENSATION, a novel self-organizing stock management structure. SENSATION is based on existing works proposed for wireless sensor networks and

adapts them to another context. It proposes an efficient unique structure which can be used for different purposes such as data replication, distributed storing and request managements. These features help in the scalability and reliability of new storing warehouse management which tends to increase in scale and to be more and more interconnected. Results show that SENSATION provides interesting and promising results in terms of reliability and scalability. Next steps would be to study in more details the cost of replication towards the gain in reliability. In addition, some real implementations and tests should be performed to complete the full analysis.

References

[1] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Looking up data in P2P systems. *Communications of the ACM*, 46(2):43–48, February 2003.

[2] Y. Busnel, M. Bertier, and A.M Kermarrec. Solist: A lightweight multi-overlay structure for wireless sensor networks. Research Report RR-6404, INRIA, 2007.

[3] J. P. Hubaux, Th. Gross, J. Y. Le Boudec, and M. Vetterli. Towards self-organized mobile ad hoc networks: the terminodes project. *IEEE Communications Magazine*, 39(1):118–124, January 2001.

[4] J. Li, J. Jannotti, D. S. J. De Couto, D. R. Karger, and R. Morris. A scalable location service for geographic ad hoc routing. In *Proceedings of ACM Mobicom*, Boston, MA, August 2000.

[5] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of ACM Sigcomm*, pages 161–172, August 2001.

[6] A. C. Viana, M. Dias de Armorim, S. Fdida, and J. Ferreira de Rezende. Indirect routing using distributed location information. In *In Proceedings of the First IEEE International Conference on Pervasive Computing and Communications (PERCOM '03)*, page 224, Washington, DC, USA, 2003. IEEE Computer Society.

[7] Y. Xue, B. Li, and K. Nahrstedt. A scalable location management scheme in mobile ad-hoc networks. In *Proceedings of IEEE Conference on Local Computer Networks (LCN)*.

Centre de recherche INRIA Lille – Nord Europe
Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399