A k-layer self-organizing structure for product management in stock-based networks

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

A k-layer self-organizing structure for product management in stock-based networks

A. Carneiro Viana[1],[3], N. Mitton[2], L. Schmidt[2], M. Vecchio[1]
[1] INRIA Saclay-Ile de France, [2] INRIA Lille-Nord Europe, Univ. Lille 1, CNRS, [3] TKN/TU-berlin,

***Abstract*—Traditional Distributed Hash Tables (DHT) abstraction distributes data items among peer nodes on a structured overlay network in storage-intensive applications. The question is whether DHT-based systems can provide reliable and scalable storage services also in stock-oriented applications, where logistics, traceability and fault-tolerance are the main requirements. In this paper a novel approach to self-organizing stock management networks, based on a two-level DHT mechanism is proposed and analyzed. Some preliminary results suggest that our method can reduce network traffic and achieve high data availability also in such storage and management networks.**

I. Introduction

Nowadays, trade industries are growing up and need more and more the support of technology for managing their goods, for logistic, inventories and traceability concerns. Such ordinary applications 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 and managed, a general communication pattern with entities in different geographic sites communicating, and a need for limited communication overhead and reliable data. The network should be standard compliant[1] able to be connected to an EPC[2] network (*e.g.* which respects the EPC standards) and support requests issued from an EPC ALE[3] engine in order to fit most of companies' applications.

The kind of networks we are dealing with is composed of heterogeneous entities. In order to identify products to be controlled, they are equipped with unique identifiers labels (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 when information needs to be retrieved from them. 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. Wireless readers may appear/disappear, generating topology changes. The network must be able to self-organize and self-configure.

In this paper, we propose SENSATIONa 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. One level is dedicated to each single object kind (*e.g.* sportswear, shoes, etc) or location according to the application needs and granularities (*e.g.* Paris, France or Europe). This unique structure allows 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 limits 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 or location 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 II describes the case studies we are focusing on in order to illustrate more easily the

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

[1]RFID relative standards are established by EPC Global.

[2]http://www.epcglobalinc.org/

[3]http://www.epcglobalinc.org/standards/ale

behavior of SENSATION. Section III sets background works useful for the understanding of SENSATION. Section IV describes the relative works useful in the construction of SENSATIONwhile Section V explains in details SENSATION. Section VI presents SENSATIONperformance results. At last Section VII concludes that work by prospecting some future works.

II. 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.

III. Preliminaries

The spontaneity of self-organizing networks makes the provision of a scalable and efficient location service in their context a non-trivial problem. Since a dynamic association between the identification and the location of a node is needed, a mechanism to manage this association has to be provided. Furthermore, this latter should minimize the control message overhead for routing and location discovery. An efficient solution is to perform an *indirect routing* [2][3][6].

A routing operation is referred as *indirect* when it is performed in two steps: *(i)* first the target node is located and then *(ii)* a directed communication with the target is performed. The indirect-routing allows the network to decouple the location information 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 and provides a general mapping between any information and a location by using $hash$ functions. A DHT uses a virtual addressing space $\mathcal{V}$, which is partitioned and assigned to different nodes in the network. As an example, consider an addressing space $\mathcal{V} = [0, 30[$ is shared among 3 nodes i, j, k such that node i gets the partition $q(i) = [20, 30[$, node j gets $q(j) = [10, 20[$ and node k gets $q(k) = [0, 10[$.

Each information is *hashed* into a key ($hash(v) = key_v \in \mathcal{V}$) in $\mathcal{V}$ and then stored in the node containing the hash key in its partition. 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.

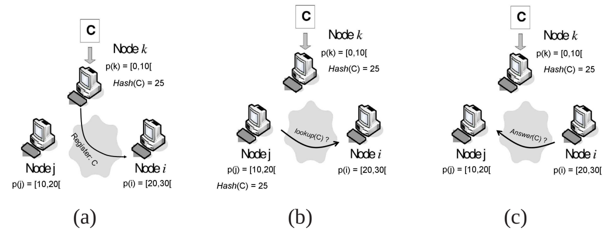


Fig. 1. (a) Node k stores content C, and registers information about C on node i which is its rendezvous node. (b) Lookup phase of node j to contact the rendezvous node of C. (c) Lookup answer with information about C.

When implementing any DHT mechanism some events related to the creation and management of the DHT-based structure have to be considered. At the following, we discuss how such events are performed.

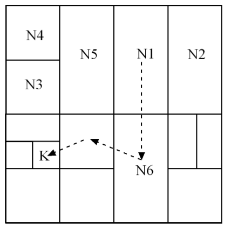
Node Arrival: When a node u enters the network, it has to join the DHT-based structure. To do so, u retrieves the way to contact a node already in the structure, which is then used as an *entry-point* to the DHT-based structure. Then, a partition in the logical address space is assigned to u. Routing information is then updated among nodes to reflect the presence of u in the structure. Finally, u retrieves all the $(key, value)$ pairs under its responsibility from the node that previously stored them.
Node Failure: When a node fails, its stored data is lost unless the DHT is using a replication mechanism 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: A node departure normally does not represent a critical event in DHT implementations. This is due to the fact that, this event often explores notification primitives to be used by nodes before leaving the infrastructure. This allows other nodes to copy application data from the leaving node, to recover the left logical address space, and to immediately update their routing information.

IV. Related works

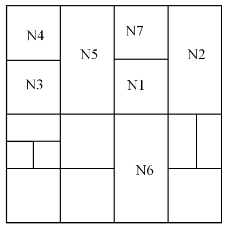
We briefly discuss here the works most related to SENSATION. For more details, we invite the reader to check the referred papers.

A. *Content Addressable Network (*CAN*)*

In [4] the authors introduced the notion of multi-dimensional structured overlays and showed its improved routing efficiency with respect to the classical one-dimensional counterpart. In particular, they proved that the average path length in a system with n nodes and d dimensions scales as $O(d(n^{\frac{1}{d}}))$.
Nodes arrival and partition assignment: In CAN, an identifier in the form $[x, y, z]$ (if $d = 3$) is assigned to each data item. 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 v, which is the current node responsible for the zone where n lays. Then, v splits its zone in half and assigns one half to n (cf. Fig. 2(b)). Finally, v transfers to n the keys lying in the zone it has become responsible for.
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 (e.g. neighbors $N1$ and $N6$ in Fig. 2(a)). If the node does not own the zone of the destination, it forwards the message to its neighbor with the coordinates closest to the destination (Fig. 2(a)). In a d-dimensional space equally partitioned into x zones, this procedure results in an average of $O((d/4)(x^{\frac{1}{d}}))$ routing steps. This expresses that increasing the number of dimensions significantly reduces the average route length.



(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 $N1$: $\{N7, N2, N6, N5\}$.

Fig. 2. CAN mechanisms illustration.

Node failure: The zones of failing 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 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.
SENSATION and CAN: SENSATION uses CAN as a framework to distribute the responsibilities area of servers in a geographical way. CAN allows SENSATION to designate the contact and entry nodes in every layer (so for each kind of products) in an easy and homogeneous way. CAN has been chosen rather than other DHT scheme because of its low complexity and geographic features that it provides to the data distribution.

B. *Tribe*

In [5], the authors proposed the Tribe protocol, specifically designed for large scale self-organizing networks. Tribe creates a topology that is a logical network representation and describes the relative location of nodes according to their neighborhood in the physical network.
Nodes arrival and 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 the highest 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. 3.
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

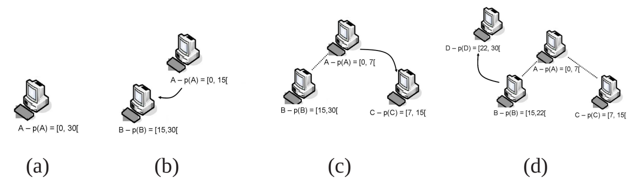


Fig. 3. Tribe virtual space sharing. (a) node A is alone and responsible for the whole virtual space $\mathcal{V} = [0, 30[$. (b) Node B pops up and gets half of A partition. (c) Node C appears, A gives again half of its partition. (d) Node D arrives, B shares its partition since it has a larger partition than node A.

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.
- 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.

Node failure and departure: Similar to CAN, the zones of failing or leaving nodes are taken over by alive nodes, *i.e.* the parents of the missing node in the tree. In the case of failures, the keys stored by the failed node are lost. Discontinuity between the control region of parents and missing nodes may happen, forcing parents temporarily manage multiple zones.

SENSATION and Tribe: Tribe is the DHT used by SENSATION in every single layer to organize data concerning a kind of objects. Tribe has been chosen as framework rather than any other DHT schemes because of its tree structure which is the most appropriate structure to perform the kind of operations provided by SENSATION (unicast, brodacast, anycast, k-cast).

C. SOLIST

In [1] the authors presented the SOLIST structured overlay, which provides an efficient $*$-cast suite for wireless sensor networks. SOLIST identifies a set of basic functionalities widely used in distributed applications and propose an efficient implementation of this suite in a multi-layer structured network. The basic idea is to assign a type to each sensor and to a layer. A layer is composed by all sensors of the same type. Fig. 4 shows a SOLIST multi-layer projection, composed by a common basic layer and a group of overlay layers.

Node arrival and routing: Each node in SOLIST belongs to the common basic layer and is assigned to a virtual coordinate in a relative Cartesian space. Messages in the basic layer are routed using a lightweight geographic routing protocol. After joining the basic layer, a node has to be included in an overlay t. For this (the same is performed to query about type t), a given sensor has to contact at least one node belonging to the corresponding layer t. To find out such a node, SOLIST proceeds in two steps. The first step consists in contacting an *entry-point* for the asked layer. The entry-point will not necessarily belong to layer t but it is aware of the closest sensor belonging to it. The second step consists in reaching the *contact node* belonging to the layer t. This two-steps contact phase allows reaching with limited energy, bandwidth, and time costs, the closest node belonging to the asked layer. In order to identify the entry-points and the contact nodes, SOLIST uses hash functions. The joint of nodes and routing in the overlay follow the CAN structure previously explained.

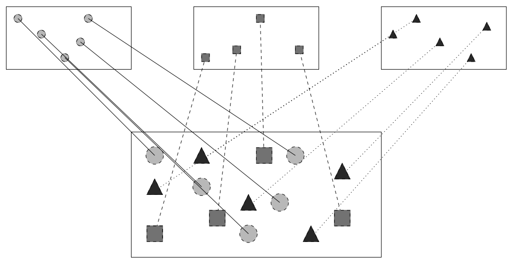


Fig. 4. Projection of 3 groups of nodes (▲, ■ and ●) into three layers.

Node failure and departure: A node failure or departure affects the way the overlay layers are managed, what is performed according CAN.

SENSATION and SOLIST: SOLIST is a general framework that has inspired the design of SENSATION. They although differ since SOLIST has been designed for wireless networks and thus is constraint by wireless links features and neighborhood definitions. SENSATION considers wired Internet networks and thus assumes that there exists an IP underlying routing allowing to easily reach every server. Yet, although applying the same basic principles as SOLIST, SENSATION re-defines every operation to fit other assumptions and to be compliant to business needs and requirements.

V. SENSATION

A. Solution overview

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

In SENSATION, a *type* is assigned to each data item in a SOLIST-like fashion [1]. A type may represent a product kind (*i.e.*, trousers) or a location (*i.e.*, Warehouse A), since it is addressed without distinction. Each type is associated to a layer. To address requests concerning a product kind or a geographical area, requests will be sent to the corresponding set of layers. As an example let us consider to have a sport articles inventory (a city inventory), composed by bikes and rackets (two different products, *i.e.* layers): if a request aims at retrieving such an inventory, then independent requests are sent to both racket and bike layers and the results are aggregated afterwards. Using SOLIST mechanisms, we can broadcast, k-cast and unicast efficiently into these layers. Once a layer is reached, a Tribe structure is linking nodes.

Each layer t is an overlay network composed by nodes and links. *Nodes represent databases and servers sharing data items of type t*. When a node n joins the *overlay network of a layer t*, a virtual space partition together with a set of neighboring nodes are assigned to n; the latter represents other nodes in t-layer which are connected to n. Note that since this is a logical network, links may not be direct in the physical underlying network. SENSATION uses Tribe [5] to assign the space partition to nodes and to create links. In addition to the overlays and as in SOLIST, nodes also belong to a common basis space. SENSATION defines this space and assigns logical coordinates in this space to nodes, which allow their location (cf. Section V-B). We use the CAN protocol in this space in order to benefits the d-dimensional virtual space and to map it with geographical coordinates.

The structure is then used for three purposes: *(i)* **Distributed storing:** every node which has fresh data of a given type t, firstly retrieves the closest node managing that type t. Through this node, the Tribe t-layer is acceded and the node responsible for storing the searched data is updated. *(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 be stored 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 network but directed to nodes concerned by the request, thus diminishing the latency and limiting the communication overhead. The first step consisting in reaching the proper layer allows the 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 to stop the request when enough nodes have been reached. We use Tribe instead of other tree base p2p system because of its lower complexity.

B. Data management structure

The SENSATION structure has to include all nodes of all sites of the 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, a *cell coordinate* (X_C, Y_C) is assigned to each cell (city), representing the relative position in the virtual coordinate system along the x and y dimensions. More in details, the cell located in the bottom left corner of the grid has $(0, 0)$ coordinate, the next ones have coordinates equal to $(1, 0)$ and $(0, 1)$, along the x and y dimensions respectively, and so on. Then, each cell is logically divided into T square areas, where T is maximum number of types supported by the application and should be a perfect square number (cf. Fig. 5). Each area represents a virtual location in the cell, with *type coordinate* (X_T, Y_T) within a cell; each location acts as an entry-point for a different type in the logical cell. Thus, type coordinates (X_T, Y_T) in cells i and j represent entry-points for the same layer T. 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 on an item of type t (insertion, lookup, *etc.*) a requesting node u has firstly to reach the t entry-point within the cell he belongs to. By contacting this entry-point node, u can reach the layer t; once in the layer, he can traverse the Tribe tree ending in the node which has actually stored the item. The critic point is how to share among nodes the virtual space represented by the entry-point set. We simply propose to share the entry-points set among the nodes in a cell, by using CAN. For this reason, at a steady state the nodes 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]. Like in CAN, we use a bi-dimensional congruential mapping function in order to determine the type-t entry-point in the (X_C, Y_C) cell ($ep_t^{(X_C,Y_C)}$), which is:

$$\begin{aligned} ep_t^{(X_C,Y_C)} &= \left(Xep_t^{(X_C,Y_C)}, Yep_t^{(X_C,Y_C)}\right) \\ &= \left(mod(t,d) + X_C \cdot d, \left\lfloor \frac{t}{d} \right\rfloor + Y_C \cdot d\right) \end{aligned} \quad (1)$$

where $d = \sqrt{T}$, $\lfloor \frac{t}{d} \rfloor$ is the integer division between t and d and $mod(t, d)$ is the remainder after this division. When $ep_t^{(X_C,Y_C)}$ is computed, the request can be routed (using CAN) towards the requested entry-point. Once reached the right layer, Tribe is used to reach the node storing the item.

Virtual coordinate assignment: to be inserted in the structure, nodes firstly needs coordinates, which are given by the user who arbitrarily assigns to each new node a unique geographic position in the grid. On the basis on the assigned site, a unique logic 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, consider a user plugs a new node 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, $ID = 01|10|11|11$ (*i.e.* 01 being the binary representation of X_C, 10 the binary

[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.

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

representation of Y_C, and 11 the binary representation of X_T and Y_T, on 2 bits each).

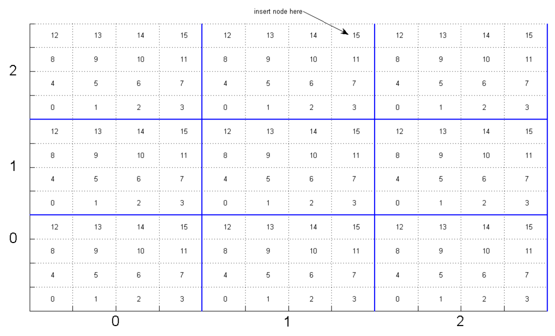
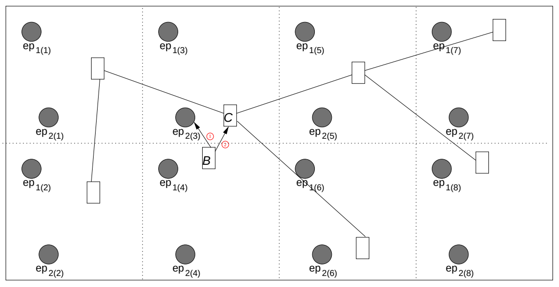


Fig. 5. Coordinate assignment to a node.

Adding a node: to join the structure, a node u has to *(i)* obtain a portion of the entry-point set in the common basis space (*i.e.* join CAN) and *(ii)* obtain a portion of the Tribe addressing space in all the layers corresponding to the types of objects it stores and has to register (*i.e.* join Tribe). Consider a node u wants to join SENSATION. The nearest CAN node v already in the structure has to give to u a portion of the entry-points it is currently managing. Let us consider, for instance, that u receives an entry-points set $P = 1, 2, 7, 8$. This means that u will represent the entry-point for any request concerning types $1, 2, 7, 8$ 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 t has to be stored in the infrastructure, the node which is currently managing the entry-point t is contacted by using CAN (note that this operation is purely local, in the sense that the responsible node of type t is local to the cell). Since this node is also registered in the Tribe t-layer, the request can be forwarded using Tribe. Notice that, when the right layer is reached, any level of replication of the data can be imposed, in order to increase the reliability. Just applying different hash functions to same item ID, in fact, would result in a different node managing the hashed ID (thus, the storing node). The only constrain in having a level of replication r is that the hash functions set r has to be known by all nodes. With this technique a tunable fault-tolerance level can be introduced. Moreover, this technique lets the application further diminish the overhead cost introduced by Tribe. This aspect will be explained later on.
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 and according to the query, a different scheme is used. In particular, for unicast queries, the request is routed using Tribe until the node managing the requested hashed ID is reached. This node is the responsible of directly providing the answer to the asking user. When a replication level r is imposed, we know that actually the item (if it was added) was stored r times in the layer. For this reason, it is less expensive (from a latency and communication overhead point of view) computing the hash function r times and reaching the Tribe node at the minimum distance from the contact node.

C. Example

To conclude this section we provide an example. Let us assume that a node B wishes to draw an inventory of trousers using SENSATION. B has first to contact the closest entry-point of the type "trousers". For this, by applying Eq. 1, B contacts ep_t using CAN and gets the location of the nearest node C managing trousers (the contact node) (Fig. 6). Note that ep_t knows node C since the latter has been registered at node ep_t using the same 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.



① Node B computes the coordinates of the nearest entry point ($ep_{2(3)}$) which returns the nearest node (C) identifier and coordinates belonging to the layer

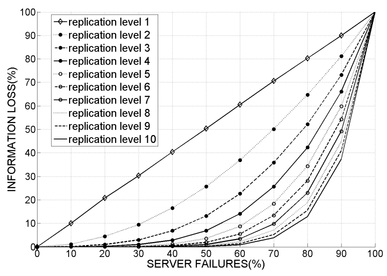
② Node B contacts node C, which is the nearest node in the ligh−t−layer

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

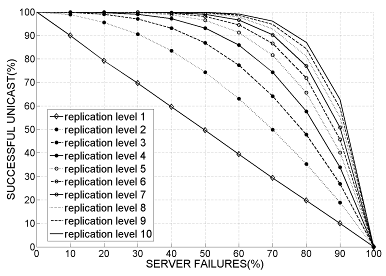
A different request may be to know whether a product exists. Node B simply contacts the closest entry-point for the asking type. If the entry-point returns a node ID then the existence is proved, otherwise the entry-point returns NOT FOUND. Finally, if a user wants to know whether there are at least k items of a given type, he contacts the closest entry-point which communicates to the nearest contact node (node C). Then, the user sends a $k = 10$-cast request to C; node C decreases k by the number of items of the given type it is currently managing and forwards the request to the next node in Tribe overlay; the operation is repeated until $k = 0$, or all nodes in the layer are contacted.

VI. SIMULATION

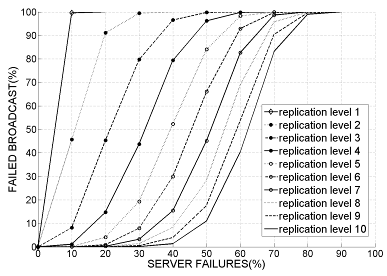
Exhaustive performance analysis is performed to analyze the routing cost of requests, the robustness at various replication levels, and the distribution of the information in SENSATION.
Scenario: a square universe is divided into 64 cells of equal size and 10 nodes are randomly placed in each cell (it follows that 640 nodes are inserted). Items are then added in the following way: 100 items of type i are inserted from randomly



(a) Percentage of lost information.



(b) Successful unicast queries.



(c) Percentage of failing broadcast queries.

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

extracted nodes, 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 node. This action (*i.e.* denoted as *unicast query*) in a classical database network should not have any cost, since the information is stored in the local node. On the other side, using SENSATION, the entry-point of the requested type has to be contacted (routing with CAN). Fig. 8 shows the overhead introduced by CAN for reaching the requested entry-points. In the shown histogram, it can be easily seen that more than 35% of the entry-points can be contacted with a routing cost of 1 hop. Note that this overhead is independent on the replication level introduced in Tribe layers.

Dealing with node failures: here, a percentage of failing nodes varying within 0 and 100% is considered. Note that, when a node is experimenting a failure, its registered items are unavailable (it is a database 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 loss regimes, the percentage of lost information (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 and even having 50% of failing nodes, only 25% of data are lost. This same amount of losses is even achieved when 70% of nodes are down with a replication level equal to 3. In addition, it is worth noting that this reflects the worst scenario since in these simulations, failures appear randomly while in realistic scenario, when a node fails, its closest nodes 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 nodes 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 nodes in which they were registered. Thus, 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.

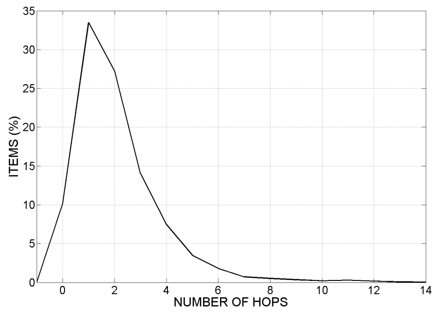


Fig. 8. Overhead in CAN.

VII. Conclusion

We have proposed SENSATION, a new self-organizing stock management structure, allowing data replication, distributed storing, and requests management. These features help in the distributed management of new storing warehouse management, which tends to increase in scale and to be more and more interconnected. Results show that SENSATION is effective in providing reliability and scalability for different replication levels and loss regimes. Future works would include a detailed study on 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] Y. Busnel, M. Bertier, and A.M Kermarrec. Solist: A lightweight multi-overlay structure for wireless sensor networks. RR 6404, INRIA, 2007.

[2] 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, 2001.

[3] 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 *Proc. of ACM Mobicom*, Boston, MA, USA, 2000.

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

[5] A. C. Viana, M. Dias de Armorim, S. Fdida, and J. Ferreira de Rezende. Indirect routing using distributed location information. In *Proc. of IEEE PERCOM*, Washington, DC, USA, 2003. IEEE Computer Society.

[6] Y. Xue, B. Li, and K. Nahrstedt. A scalable location management scheme in mobile ad-hoc networks. In *Proc. of IEEE LCN*, Tampa, FL, USA, 2001.