Specification of a Framework, Fully Distributed, for the Management of All Types of Data and the Services Close to Users

Thierno Ahmadou Diallo(✉)

LI3, Assane SECK University of Ziguinchor, BP 523, Ziguinchor, Senegal
t.diallo@univ-zig.sn

Abstract. This article presents GRAPP&S (Grid APPlication & Services), a specification of a multi-scale architecture for the management (unified storage and indexing) of data and services near users. We manage all types of data and services through the use of specific node called proxy. GRAPP&S's architecture consists of three types of nodes, each with different roles. These nodes are grouped together in the form of communities (local networks) using multi-scale principles. The data is presented transparently to the user through proxies (an example of GRAPP&S nodes) specific to each type of data. In addition, the GRAPP&S architecture has been designed to allow the interconnection of different communities and the application of security and privacy policies, both within a community and between different communities. Our framework adopts a routing mechanism prefixed for research and access to data GRAPP&S. This access does not depend on a direct connection between the nodes, as in most P2P or other networks. In GRAPP&S, it is always possible to route the data transfer path used when looking at cases where a direct connection between the nodes is not possible.

Keywords: Hierarchical system · Proxies · Hierarchical addressing · Prefix routing

1 Introduction

Large-scale data management is a recurring problem in both science and business. Despite constant advances in memory and disk capacity, the use of a single storage device is no longer an option as competitive access, reliability, energy consumption and cost are obstacles to development systems. It is for this reason that researchers and developers have long turned to the development of distributed storage solutions, in order to work around these limitations.

In cases where the data can be represented under the In the form of files, NAS/SAN type solutions, P2P networks and also cloud storage represent technological choices capable of offering large-scale storage at a reasonable cost. These

N. Kumar et al. (Eds.): UBICNET 2021, LNICST 383, pp. 26–44, 2021.
https://doi.org/10.1007/978-3-030-79276-3_3

choices also apply to relational or NoSQL databases, but in this case access to the data always requires a (pseudo) centralized entity capable of aggregating and presenting the data (this does not exclude the parallel processing of data requests).

Through different strategies, these distributed solutions offer transparent solutions to increasing storage needs, while offering sufficient guarantees to ensure the consistency and sustainability of data. Today, the use of NAS/SAN or cloud file storage solutions has become as common as the use of USB drives or drives, once the potentially unlimited resources offered by P2P or cloud solutions present themselves several advantages in terms of cost, availability and use of physical resources. However, these solutions can also have drawbacks related to speed of access and data security; the solution to these drawbacks is still far from guaranteed and depends mainly on proprietary solutions offered by storage service providers. Another aspect to consider is the compatibility between the systems: if some APIs make the manipulation of files relatively simple, it is less obvious the integration of other data representations such as requests on a database, data streams or the performance of services. It is with the aim of providing a unified architecture for data and services that we present GRAPP&S (GRid Applications and Services), a multi-scale architecture for the aggregation of data and services. This framework has been designed to transparently integrate file type data as well as databases, streams (audio, video), Web services and distributed computing. Through a hierarchical structure based around the concept of "communities", GRAPP&S allows the integration of information sources with heterogeneous access protocols and various security rules.

Other contributions from GRAPP&S are the establishment of a separate specification of the communication middleware (P2P or otherwise), and the definition of a generic storage solution. In the first case, this is possible because the specification of GRAPP&S is based on properties such as management of community connectivity and management of routing between nodes. In the second case, the use of "proxy data" nodes makes it possible to unify access to various resources such as files, streams, services (FTP, mail) or data created on the fly by calculation spots or sensors.

Finally, data access is not dependent on a direct interconnection between nodes, as is the case with most P2P networks. In GRAPP&S, it is always possible to route the data transfer by the path used during the search, in case a direct connection between the nodes is not possible.

The remainder of this article is organized as follows: Sect. 2 briefly introduces the state of the art regarding multi-scale systems and P2P networks. Section 3 presents the main elements that make up the GRAPP&S architecture and how these elements interconnect, while Sect. 4 details the main operations related to managing and retrieving information in GRAPP&S. Section 5 details the current state of development of GRAPP&S and introduces the GAIA module for optimizing multi-scale storage [3,14]. Finally, Sect. 6 offers our conclusions.

2 Releated Work

2.1 Multi-scale Systems

In general, multi-scale systems are therefore structured around services deployed on several levels and which are completely in order to meet the more or less immediate needs of customers. Rottenberg et a [1] formalize this definition on the form: A multiscale system is a system distributed over several levels of different sizes in one or more dimensions (equipment, network, geography, etc.). In [3] presents an example of a multi-scale system in his work around "cloudlets". In this work, the authors examine the limitations of mobile devices and the inadequacy of the solutions currently in place for the outsourcing of mobile services (in particular the transfer of these services to cloud computing). The proposed solution is the implementation of stateless proximity servers connected to the Internet (cloudlets) to which mobile devices can connect directly via a Wi-Fi network. These devices are deployed as Wi-Fi hotspots in cafes, shops, etc. They have high computing power and allow mobile devices to outsource overly complex calculations to a nearby machine, which solves cloud computing latency issues. This paper [10] has proposed an application-aware cloudlet selection strategy for multi-cloudlet scenario. Different cloudlets are able to process different types of applications. To reduce latency and work overload in mobile clouds, the authors propose, strategy can balance the load of the system by distributing the processes to be offloaded in various cloudlets.

2.2 Peer-to-Peer (P2P) Networks

In [5] this work, propose a novel P2P-based MCS architecture, where the sensing data is saved and processed in user devices locally and shared among users in a P2P manner. This article fixes, the traditional server-client MCS architecture often suffers from the high operational cost on the centralized server (e.g., for storing and processing massive data), hence the poor scalability. Gassara and al. [4] propose a resource management middleware in a multi-scale context. The aim of this system is to offer flexible management of a set of resources distributed in a multi-scale manner. In this article, the term multiscale is used to describe the distribution of resources across networks of different sizes: PAN (Personal Area Network), LAN (Local Area Network), WAN(Wide Area Network). The proposed solution consists of grouping the resources into domains and domain federations in order to implement different management models. The authors propose a formal approach supporting the correct description of deployment architectures and their reconfigurations. According to defined models, correct deployment architectures are generated and one of them is selected to be deployed. This generation process is based on a multi-scale modeling approach adopting Bigraphs. In fact, the architecture of a scale is refined by adding the components of the next scale. Then, the obtained architecture is in turn refined and so on, until reaching the last scale. The transition between scales is performed through applying refinement rules. Based on correct by design,

the refinement process is executed on a correct scale architecture (respects the defined models) by applying correct rules. In [7] authors propose a cooperative caching scheme for structured data via clusters based on peer connectivity in mobile P2P networks. The proposed scheme reduces data replacement time in the event of changes in topology or cache data replacement using the concept of temp cache. This scheme shows its limits in the multiscale case. P2P networks, are robust systems because it combines the advantages of DHT and unstructured systems, but remains also limited by their drawbacks, such as dependence on resources of the same type(files only) and the exponential number of messages generated during a flood search. Faced with this work, we are motivated to propose the specification of a more generic hierarchical architecture, which allows the storage of all data formats while retaining the advantages in terms of network performance.

3 GRAPP&S

3.1 Model and Definitions

Model and Definitions. For the definition of the GRAPP&S architecture we consider a communication model represented by an undirected and connected graph $G = (V, E)$, where V designates the set of nodes of the system and E designates the set of communication links that exist between the nodes. The model used for our system is studied in [2]. Two nodes u and v are said to be adjacent or neighboring if and only if u, v is a communication link of G. $u_i, v_j \in E$ is a bidirectional channel connected to port i for u and to port j for v. So nodes u and v can send or receive messages to each other in asynchronous mode. A message m in transit is noted $m(id(u), m', id(v))$ or $id(u)$ is the identifier of the node sending the message, id (v) is the identifier of the receiving node, m' indicates the content of the message. Each node u in the system has a unique identifier id and has two primitives: **send(message)** and **receive(message)**. For the sake of clarity, we introduce some definitions.

Definition 1. A node is defined as being a capacity of calculation, of storage, with means and channels of communications.

Definition 2. Raw data is a stream of bytes which can be in different forms: an object database or relational, a file (text and hypertext, XML), a stream (video, audio, VoIP), P2P files, database queries or results from a calculation/service.

Communication and Overlay Networks. The communication model presented in Sect. 3.1 is generic enough not to influence the way whose messages are actually delivered, being limited only to the definition of bidirectional communication properties between two vertices. For this reason, the GRAPP&S architecture can be based on any one overlay communication network which guarantees a reliable two-way communication between two vertices, which allows to explore different communication paths for each directed edge (routed network). This

gives greater freedom of implementation and adaptation to the runtime environment, since send/receive operations can be implemented in different ways, depending on the capabilities of node communication. In this case, three main scenarios can be considered:

- Push, where the sender is able to send a message directly to the recipient, - Pull, where the receiver regularly searches for messages on standby (this model is frequently used in the case of networks behind a NAT/firewall), and - Proxy, where the neighbors must go through an intermediary node in order to exchange messages (for example, thanks to a publisher/subscriber middleware). In these three scenarios, it is always possible to establish a direct or partial neighborhood between the processes, which is compatible with the model by directed connected graphs and which therefore meets the needs of the GRAPP&S architecture.

3.2 Elements of the GRAPP&S Architecture

In order to present our architecture, we first introduce some notations. A community (Ci) is an autonomous entity, which groups together nodes which can communicate with each other and which share a defined property: same location, same administration authority (remote servers belonging to the same company, for example) or same application domain (business database, for example). A community contains a single Communicator - (c) process and at least one Resource Manager process- (RM) and a Data Manager - (DM) and these processes are organized hierarchically in a community. The interconnection between different C communities is done through point-to-point neighborhood links between Communicator processes.

Communicator (C). The Communicator node(c) plays an essentially related role the transport of information and the interconnection between different communities, such as when passing messages through firewalls. It is the entry point to the community, and it ensures its security from the outside, through the establishment of Service-Level Agreements (SLAs) with other communities. Likewise, the communicator coordinates the internal security of the community, and can modify its access policies through decisions made within the community [8]. A node c has a unique identifier (ID) from which we construct the identities of the other nodes in the community. This node does not store data and does not index.

Resource Manager (RM). The Resource Manager (RM) processes index and organize data and services in the community. They receive requests from users and ensure their preprocessing. RM nodes participate in finding data in the community. For fault tolerance and performance purposes, the information indexed by the RMs can be redundant and/or partially distributed (DHTs, for example). In order to make the coordination of RMs more efficient, we recommend the election of a designated RM (see Sect. 4.c) ger which allows the node DM to communicate with the RM node to which it is connected.

Data Management (DM). Data Manager (DM) processes interact with data sources, which can be in different media such as databases (object or relational), documents (text/XML/multimedia), streams (video, audio, VoIP), data from sensors or a cloud service. A DM node is a service that has the following components: (i) an interface (proxy) adapted to the different sources data (hard drive, WebDAV server, FTP, database, Dropbox type cloud storage, etc.) and connected to them by a connection protocol specific to the type of data, for example JDBC, ODBC, FTP, etc. (ii) a query manager that allows you to express local or global queries and ((iii) a communication manager which allows the node DM to communicate with the RM node to which it is connected.

3.3 Community Management

GRAPP&S can be deployed in several types of architecture depending on the placement of the nodes. In the placement model (i), the nodes can be grouped together in a single physical machine (see Fig. 1.a). This is a typical example of a machine from a private individual, who wishes to host an architectural community. The placement of nodes in this form can be justified by its simplicity to implement during its implementation phase, using the concepts of inheritance and polymorphism. The nodes are interconnected by sockets, RPC solutions so that they can communicate by message in both directions between two nodes.

GRAPP&S can be deployed in several types of architecture depending on the placement of the nodes. In the placement model (i), the nodes can be grouped together in a single physical machine (see Fig. 1a). This is a typical example of a machine from a private individual, who wishes to host an architectural community. The placement of nodes in this form can be justified by its simplicity to implement during its implementation phase, using the concepts of inheritance and polymorphism. The nodes are interconnected by sockets, RPC solutions so that they can communicate by message in both directions between two nodes.

In (ii) the nodes are organized in a server farm such as a cluster, which is characteristic of HPC networks (Fig. 1.b). Finally, (iii) nodes can be grouped together if they share the same location or administration property (see Fig. 1.c). This is an example of a network formed by the nodes of a company or a research laboratory. Each GRAPP&S node has a unique identifier (ID). IP or MAC addresses are not sufficiently precise identifiers because they do not uniquely identify the different nodes that may reside on the same machine (for example, one RM and several DMs).

Additionally, the use of IP or MAC addresses does not guarantee a unique identity, as private IP addresses can be reused just as much as MAC addresses. Indeed, some unscrupulous manufacturers reuse the MAC addresses which are assigned to them, and this causes many problems in the local networks, just like in the deployment of IPv6 networks. Thus, we propose the solution which consists for each node thus has a single ID_local string, in the form "$urn : communityname : uuid : bit - string$". The expression of the hierarchical addressing is done by the concatenation of the IDs in the form of a prefix, i.e., the ID of the node ci is equivalent to its ID_local, the ID of the ID_{RM_i}

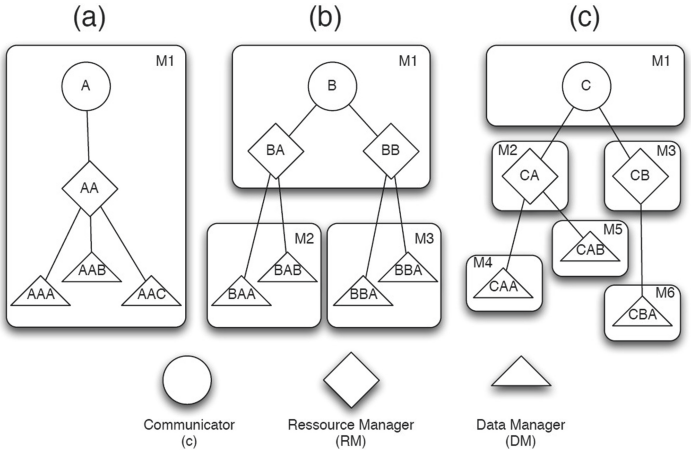

Fig. 1. Organization of nodes (a) in a machine, (b) in a cluster and (c) in a network

node is formed by ID_{c_i}-ID_{RM_i}, and the DM_i node ID has the form ID_{c_i}-ID_{RM_i}-ID_{DM_i}. An advantage of using a clean addressing model to GRAPP&S is that this makes it independent of the addressing model of the overlay network on which GRAPP&S is implemented. Thus, two GRAPP&S communities implemented on different middleware (TomP2P and Phex3, for example) will always be compatible, once the connection has been established between their communicators.

3.4 Node Management

The topology of the network changes frequently due to the mobility of nodes. We are working under the assumption or nodes that arrive in the network are initially a node of type DM. Depending on the conditions of the environment where this node is located, it may be assigned additional roles and "move up" in the hierarchy.

Connecting a Node. When a DM node arrives in the network it has two ways to find an RM node on which it can to log in.

- If the DMi node knows one or more RM nodes, it sends a Hello() broadcast message and collects all the identities of the RM nodes, which it keeps in an array ordered by the identifier. He can thus connect to the RM node which has the highest identifier. If the latter disconnects, then the DMi removes it from the table and connects to the next RM node;
- If on the other hand the DM node does not know any RM node, it must carry out a discovery on the local network (for example, thanks to a multicast) or contact a directory service which can indicate the identifier of a node RMi.

As the way to find the RMi node depends on the implementation, it is not specified in our architecture. Finally, if no attempt to connect to an RM node (and by extension a c node) is successful, the DM node has the option of forming its own community. It thus assumes the three roles c, RM and DM, until other nodes join it. At this time, an election can take place in order to redistribute the roles between the nodes.

Disconnecting a Node. Nodes may experience voluntary disconnections or unintentional (breakdowns). As in the case of disconnections voluntary is trivial, here we focus on unintentional disconnections. Between two hierarchical levels, failures can be detected either by periodic Pull type messages (also known as heartbeat), on demand by Push messages (ping-pong) [9] or even by relying on a specific mechanism. Middleware overlay. For nodes belonging to the same hierarchical level, monitoring can also be done through a mechanism for passing "service" tokens. This not only allows the lightening of the detection mechanism (it is enough to monitor its predecessor and its successor) as allows the rapid dissemination of information to all nodes. For the implementation of a generic failure detection mechanism, we recommend a two-step procedure. First, each node has a list of neighbors $\{N_1, ..., N_n\}$ composed of nodes in direct contact (for example, an RM_i is in contact with its c, its DMs and its neighbors RM_{i-1} and RM_{i+1}. To this list of neighbors is associated a list of timers of wait $\{ta_1, ...ta_n\}$. When no message from node Nk is received until the expiration of time tak, a suspicion of failure is raised and should be verified with a second node which is also in direct contact with the suspect node. Thus, if the suspicion concerns node c, a node RMi questions its direct neighbor RM_{i+1} with a token message initialized to false. If RM_i received a message from node c before the expiration of its tac timeout, RM_{i+1} changes the token value to true and returns the token message to its sender RMi. This means (indirectly) that node c is not disconnected and node RM_i can send a message to node c again. If on the other hand $RM_i + 1$ has not been contacted recently by c, it will forward a token with the value false which, thanks to the passage of the token, will alert all the RM nodes $\{RM_1, .., RM_n\}$ of the failure of c. Similarly, if a node c suspects an RM_k node, it can ask RM_{i+1} for confirmation. Obviously, this generic procedure can be adapted to different situations such as a node which contains one RM and several DMs. In this case, the detection mechanism can be lightened to better respond to the characteristics of the node. Following the confirmation of a failure, the affected nodes must (i) update their information (neighbor list, index tables, etc.) and possibly (ii) proceed to the election of a new RM (respectively c) which will take care of any orphan DM (or RM).

Election Algorithms. Given the dynamic and volatile nature of computer networks, it is important to choose an election algorithm that is as light and responsive as possible. The election of a node may be necessary in two situations:

either to replace a failed node and guarantee continuity of service (for example, during the failure of a node c), but also to simplify coordination between nodes of the same type, with for example the election of an RM which would act as a "supernode" for the indexing of data and services. It should be noted that prior knowledge of higher level nodes is not mandatory, as different techniques are used to obtain the identifiers of other nodes. The simplest method consists in using the GRAPP&S addressing mechanism directly: being independent of the communications middleware, this addressing system makes it easy to go up the GRAPP&S hierarchy and to contact other nodes (thanks to the routing of the r Water overlay). It is therefore sufficient to go up the levels of your own identifier or to contact other nodes whose identifiers have been collected (those from which you have recently received requests, for example). This technique also allows contact other c_j communicators and re-integrate into a network of communities after their ci communicator inadvertently disconnects. As a last resort, GRAPP&S can rely on any topology discovery mechanisms (broadcast/multicast) offered by the own overlay. Since the problem of reconnecting to the rest of the community can be dealt with more or less easily within the GRAPP&S own architecture, it is interesting to look into the election algorithms themselves.

In GRAPP&S, we recommend an election algorithm distributed model inspired by OSPF and IS-IS routing protocols [13]. Indeed, GRAPP&S nodes have a unique identifier that can be used systematically by these election algorithms. The choice between the IS-IS or OSPF algorithms is more related to implementation preferences and node heterogeneity. Indeed, the IS-IS election algorithm is deterministic, where the chosen one is always the node with the largest identifier (called DIS - Designated IS). This mechanism is simple to implement and requires practically no exchange of information because the nodes already have a list with the identifiers of their neighbors, there would only remain the cost associated with taking up the functions of an elected node has a different role from the one it previously occupied. The disadvantage of this technique is that a network with a high rate of volatility can cause repeated elections, either when the leader is disconnected or when a node is connected with a priority identifier.

In cases where volatility may impact the performance of the network, it is possible to use a non-deterministic mechanism such as OSPF. In this type of more conservative algorithm, the choice of a leader (DR - Designated Router) is only necessary if the current leader disappears. Thus, the entry of new nodes into the community has a less significant impact on the functioning of the network.

4 Operations in GRAPP&S

4.1 Storage and Indexing

Data storage in the GRAPP&S network involves Data Manager DM nodes, while Resource Manager RM nodes are used to index data and services. At the end,

each piece of data is uniquely identified by the DM node identifier, to which is added an extension containing information and the MIME type of the data. This makes it possible to cross the barrier of the simple "file name", and can therefore make coexist static data (files), dynamic data (queries on a database, results of a calculation) and temporary data (voice or video stream, state of a sensor, etc.).

Adding new data to the network is done as follows: When a DM_i node arrives in the network, it connects to an RM node and publishes the characteristics of its data to be indexed. Any changes to the data on a DM are propagated to the RM to which it is connected, which can then update this information and share it with other RMs.

This information propagation can take different forms depending on the policies used when implementing the RM network. An implementation that wants to keep it simple can simply keep a local index on each RM, which will be consulted during a search. On the contrary, an implementation wishing to minimize the exchanges during a data search will consider the use of a super-node within the RMs or a DHT mechanism. It is also possible to promote the replication of indexes and (see data), which requires coordination between RMs in order to keep copies consistent. In any case, the overload of a node's functionalities ("supernode") is not an obligation in our structure but simply a specificity that may be present in a given implementation.

4.2 Basic Routing

All of our primitives, whether for resource research or data transfer, are based on a common and standard routing scheme within our architecture. We can therefore use a hierarchical routing scheme, adapted to GRAPP&S. A community of GRAPP&S is a tree T, whose vertices correspond to its different components. If one considers T with a classification of the vertices of T according to a DFS [12], by construction one thus obtains: for each vertex X of identifier IdX the address of X is constituted by the binary string representing IdX concatenated by the binary strings of the parent vertices of X in T. The address of each vertex X is therefore constituted by a binary string P ATHX and an integer LpathX representing the length of the chain.

For any vertex X in the tree T, we consider the variable MasqueF the mask formed by a binary string of LpathX + 8 bits whose first LpathX bits are all at 0 and the last 8 bits are at 1. For example for a vertex X such that LpathX = 16, its mask will be MasqueX = 00000000. 00000000. 11111111. This construction allows to obtain the following properties: let Y be a vertex of the tree T, if Y is a descendant of X in T, then PATHX is a substring of PATHY and by applying a logical AND between MasqueX = 00000000.00000000.11111111 and PATHY we have the identifier of Y.

If Y is not a descendant of X in T, then we have to go through the father of X to get to Y. Let Ancestor (X, Y) be the function which for any pair of vertices returns TRUE or FALSE depending on whether X is the parent of Y in the tree T or not. The architecture of GRAPP&S is hierarchical and consists of three levels. This limits the size of the node addresses to a reasonable size, even in a large network.

```
m : message to send to the destination;
local_address : address of the local node;
destination : destination node;
add : Adresse d'un nœud;
Procedure Route(m[, destination])
    If (destination ≠ ∅) then
        If (canReach (destination)) then
            add ← getAdd (destination);
            Send(m, add, local_address);
        else
            add ← getNextHop (local_address, destination);
            Send(m, add, local_address);
        end If
    else
        add ← father;
        Send (m, add, local_address);
    end If
End
```

Algorithm 1: Basic routing method

The basic routing algorithm (see Algorithm 1), allows the transmission of messages between two vertices, knowing the identifier of the source node and that of the destination node, and of course, without having to calculate routing tables. The main functionalities of the GRAPP&S architecture are research and access to data and services. Research in GRAPP&S makes use of hierarchical routing which is based on the identifiers of the nodes (prefixed addressing) which define the paths through which the search requests pass. As we will see below, access to data and services does not depend on a direct connection between nodes. In GRAPP&S, it is always possible to reconcile information by routing in the event that a direct connection between the nodes is impossible.

```
m : message;
role : role of the node (DM, RM, c);
m.type : tmessage type (req — reply — get — data);
Procedure Receive(m)
    If (role=DM) then
        If (m.type=reply) then
            m' ← ("get", m.data, local_address)
            Route(m', m.source);
        end If
        If (m.type = get) then
            m' ← ("data", data, local_address)
            Route(m', m.source);
        end If
    end If
    If (role=RM) then
        If (m.type=req) then
            If (recherche_locale(m)=FAUX) then
                If (origin=child) then
                    Route(m, father);
                end If
            end If
        else
            Route(m, m.destination);
        end If
    end If
    If (role=c) then
        If (m.type=req) then
            Diffusion(m);
        else
            Route(m, m.destination);
        end If
    end If
End
```

Algorithm 2: Message processing according to the roles of the nodes

```
m : search message;
m.source : address of the DM which sent the message;
m' : reply message;
local_address : local node address;
Function Recherche_locale(m) : boolean
    listeDM ← RM.Verif_of_RM(m.data)
    If (listeDM ≠ ∅) then
        For ( each DMk of RM such as DM_k ∈ listeDM ) do
            m'← ("reply", m.data, DM_k)
            Route(m', m.source);
        end For
        return TRUE
    else
        return FALSE
    end If
End
```

Algorithm 3: Local search

```
Index_of_RM[0...n] : index du nœud RM;
X ←{} : list of DM;
TypeMime : MIME type of information ;
TypeMimeRec : Search criteria;
Function Verif_of_RM(TypeMimeRec) : X ←{}
    For (i = 0 to n) do
        If (Index_of_RM[i].TypeMime=TypeMimeRec) then
            X ← DM_i;
        end If
    end For
    return X;
End
```

Algorithm 4: Verifying the index RM

```
m : message to be broadcast;
origin : address of the RM which transmitted the message;
local_address : : address of the local node;

Procedure Diffusion(m, origin)
    For ( each child RMk of ci such as RM_k ≠ origin) do
        Route(m, RM_k)
    end For
End
```

Algorithm 5: Message broadcast to RMs

4.3 Data Search

Among the main functionalities of GRAPP&S is search and access to data. This section describes our algorithm for finding data in a community of GRAPP&S. The different nodes which intervene in this research have some basic functions, some exposed previously, on which we will rely to define our algorithms: Send allows a source node to send a message to a destination node. However, this primitive requires a direct connection between nodes, which is why it is associated with the Route primitive; Route allows a source node to transmit a message to any node, either by sending it directly through Send, or by relaying the information to a node with the routing mechanism prefixed.

In order to get an address in the prefixed routing mechanism, we use a getNextHop() method which performs the comparison of the addresses and applies the subnet masks in order to get the address of the next node towards the destination. Receive which is used to process the messages received, according to their types and according to the role played by the node (see Algorithm 3). $Local_research$ that allows a DM to respond to a resource search. When searching, messages can be of two types:

- type search message: ("*Req*", *content*, *source*)
- type response message: ("*Reply*", *content*, *source*)

When a client searches for data on GRAPP&S, it comes into contact with a DM_i proxy, which sends a Y request containing the characteristics of the data and the type req. So the Y query takes the form <"req", content, source>. As the message does not have a specific recipient, research in a GRAPP&S community is done in stages, respecting the hierarchical organization of the network. The search procedure is as follows:

1. $DM_i \in C_i$ sends the request Y to its node $RM_i \in C_i$ using Route () (see Algorithm 1).
2. RM_i receives the request Y according to its type (see Algorithm 3) and checks using the $Local\_Search$ (m) method (see Algorithm 1) whether in its index there is a neighbor DM which contains the data sought
3. if yes, then the RM_i node returns to the DM_i node a list of nodes DM which contain the information sought, in the form of a Y″ message with the type reply. The form of the Y″ request is ("*Reply*", *content*, *source*). The illustration is given in Fig. 6 where the node $DM_i \in C_i$ with identifier AAA sends the Y request of type req to its node $RM_i \in C_i$ using the Route() function. The AA identifier RM_i node replies by sending a reply type message containing the identifier of the DM_y node which contains the data sought.
4. if not, the node RM_i forwards the request to its node $c_i \in C_i$ for a broadcast to the other nodes $RM_i \in C_i$ such that the $RMk \neq RMi$ (Algorithm 5). Each of these RM_k nodes execute the $Local_Search(m)$ method (see Algorithm 1) to check if in its local index there is a neighbor DM which contains the data sought,

5. When an $RM_i \in C_i$ node finds a match, a reply stamped reply from this message with the form <"reply", content, source> will be returned to the sender DM_i node by taking the reverse path to go using Route () (see Algorithm 1).
6. If none of the $RM_k \in C_i$ nodes finds a match, and if the C_i community is connected to other communities, then the ci forwards the request according to the access policies defined and implemented, or negotiated, in place in each community.

At this search step, we do a flood search, flooding on the community tree. This search by flood is illustrated in Fig. 2 where the communicator node Ci of identification A broadcasts in the community Ci to all $RM_k \in C_i$ the message Y of type Req. The identifier RM_k node $BB \in C_j$ finds a match and responds to the AAA identifier DM_i node by taking the reverse path with a reply type message containing the identifier of the DM_y node that owns the data. However, it can be noted that the flooding is limited to the structure of the GRAPP&S communities, and not to the structuring of the underlying peer-to-peer network. Here are now the different algorithms of the functions presented previously. First, the specific management of messages according to the different types of nodes. Then the Local Search for a resource algorithm. Now the Resource Manager index test and verification algorithm. Finally, the algorithm ensuring the dissemination of messages to the Resource Manager.

4.4 Data Transfer in GRAPP&S

If the search is successful, the client (DM_x) obtains the identifier of the DM_y node that holds the data. During the data transfer stage, messages can take two different forms

- content request message $(get, content, source)$
- content transfer message $(data, content, source)$

When a node receives a message of type *(data, content, source)*, if it is not the target of this message, then it retransmits the message. If the latter is intended for him then he decides either to display the content of the message or to store it thanks to a Save() primitive on a support such as: on disk, on a cloud service such as Dropbox or Seafile, on a messaging service such as Gmail, etc. In this case, he has two possibilities to contact DM_y and recover the data: (i) by direct connection; (ii) by a routed connection if direct connection is not possible. In both cases, we will call the Route() primitive. If direct connection is possible, sending will be done by *Send()*, otherwise it will be done by the hierarchical routing mechanism. *Remark 1.* As the resource was found by a search and its location was returned to the site that initiated the request, there is therefore a bidirectional path using the links of the hierarchical architecture of GRAPP&S, which can be used between the requester and the resource.

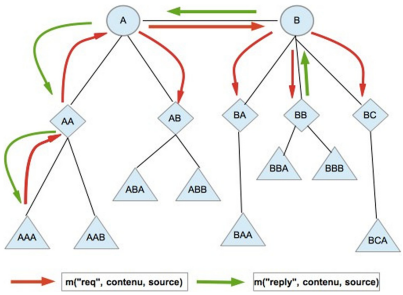

Fig. 2. Research in the GRAPP&S network

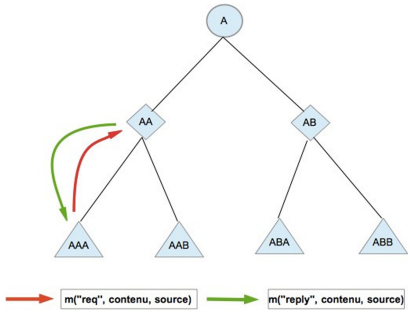

Fig. 3. Local search in GRAPP&S

Direct Connection Possible. If a direct connection is possible between the requesting node and the owner of the resource, the DM_x node directly sends the request $(get, content, source)$ to the DMy node by a simple Send() method (see Algorithm 2). The latter DM_y responds with the transfer message $((data, content, source)$ by following the reverse path forward (see Algorithm 2). This is the example of nodes that are in the same subnet. An illustration is given in Fig. 3 where the AAA identifier DM_x node sends directly to the node AAB identifier DM_y, a request to access the data. The latter responds by following the reverse path. Then the DM_x node can view the data or store it on a medium such as a disk, on a cloud service, on Gmail etc., thanks to the save() function. *Remark 2.* Two ABC and DEF identifier nodes may well be part of the same network, and be part, as their identifiers show, of two GRAPP&S communities. In this case, a direct connection will then be possible. For direct connection not possible.

Direct Connection is Not Possible. If direct connection is not possible, then the message is sent by hierarchical routing in GRAPP&S. The hierarchical routing mechanism is applied, and therefore the resource will follow the path taken by the search request and its associated response. In this case, the operations carried out are as follows

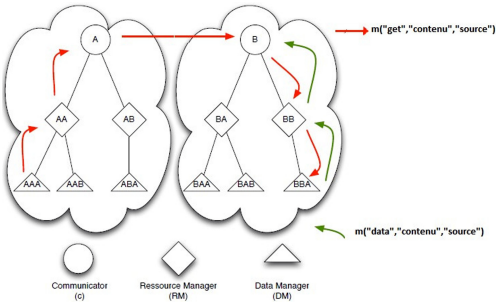

Fig. 4. Data repatriation via a routed connection

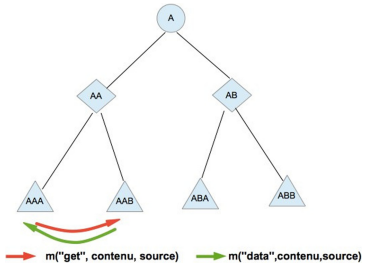

Fig. 5. Repatriation in the case of a local network in GRAPP&S

- the DM_x client sends a get type request to its RM, which will retransmit the get message using its Route() primitive to its parent, the communicator node C_i.
- the C_i node, according to the prefixed routing (see Algorithm 2), sends the get request to the RM node which indexed the data.
- The latter, by prefixed routing (see Algorithm 2), forwards the get request to the DM_y node responsible for the data.
- the DM_y node, which owns the data, can transmit it with the Route () primitive (see Algorithm 2), thanks to a data type message by following the reverse path on the way. Then the DM_x node can visualize the data or store it on a support arbitrary, thanks to the save() function.

Figure 4 Illustrates this routed connection, where the AAA Identifier DM_x node sends the get request message using its Route() primitive that goes up the tree to download the data managed by the BBA Identifier DM_y node. The latter the DM_y node in the same way, when it receives a get type message, the DM_y node which holds the data can transmit it with the Route() primitive, thanks to a data type message (Fig. 5).

This hierarchical search mechanism prevents the flooding of network links. The hierarchy allows to define the paths by which the requests transit and as the logical connectivity of our architecture is by definition of (n − 1), it suffices

to apply a PIF type algorithm to aggregate the requests and reduce the number of messages from a search.

As access to GRAPP&S data does not depend on a direct connection between nodes, it is always possible by routing to retrieve the data by the path used during the search, in case a direct connection between the nodes does not is not possible.

5 Future Work

5.1 Development of a Prototype on Pastry

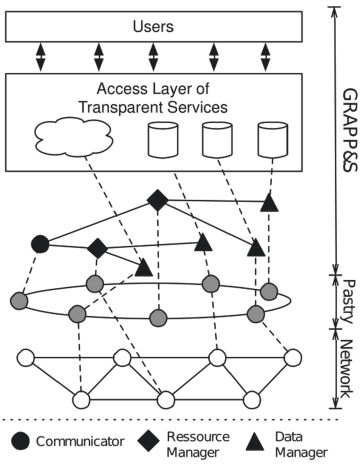

Fig. 6. Illustration of the distribution of nodes GRAPP&S on a Pastry overlay

A prototype of the GRAPP&S architecture is under development. This prototype uses the TomP2P overlay network to interconnect the various nodes of the GRAPP&S architecture, as shown in Fig. 6. A prototype of the GRAPP&S architecture is under development. This prototype uses the TomP2P overlay network to interconnect the various nodes of the GRAPP&S architecture, as shown in Fig. 6. The use of TomP2P is purely practical because the Pastry [11] overlay takes care of any low level interconnection (opening of sockets, detection of failures, etc.), thus allowing developers to concentrate on the coordination of GRAPP&S nodes (election, indexing of data and services, processing of requests, research, etc.). The use of a design pattern like Facade allows a modular development where the overlay can be easily replaced. Indeed, the performance constraints related to node management and data transfer may be too high for TomP2P, if the community is expected to have a high number of nodes or if the network has a high level of volatility. In addition to demonstrating the application of the GRAPP&S architecture and the implementation of scalability tests,

this prototype will also allow the analysis of the impact of the location of the nodes in relation to the management of the hierarchy. Indeed, the operation of GRAPP&S is based on a hierarchical structure for the routing, the search and the interconnection of remote nodes. If the placement of nodes has an impact on the performance of the architecture, it should also be taken into account when selecting RMs and c.

5.2 Resource Redistribution with GAIA

While GRAPP&S initially acts as a mediator for the location of resources stored in the different DMs, it also has the potential for development thanks to the redistribution of resources through the GAIA scheduler. Like the body of the same name for the work of Isaac Asimov, GAIA aims to integrate and possibly reorganize resources between the different MDs, according to criteria such as frequency of use, proximity e vis-à-vis customers, the need for replication, the cost of storage and, of course, the feasibility of such a distribution. Indeed, the GAIA project concerns the development of a scheduler that will be able to manage the storage of certain types of data in order to increase the efficiency of access to the most used elements, while reducing the storage costs of less critical data. Thanks to GAIA, GRAPP&S will be able to better manage the integration of "slow" storage services such as Amazon Glacier, which present a reduced cost but also more restrictive access policies than traditional storage mechanisms.

5.3 Future Application of Our Architecture

Our architecture will be used to connect the health institutes (public for example), interested in the sharing of resources through a pooling of the health data of each institute in the form of "community". Indeed the revolution in information technologies, and the spread of the Internet of Things (IoT) and smart city systems, have fostered widespread use of smart systems. As a complex, no stop service, healthcare requires efficient and reliable follow-up on daily operations, service and resources.

Faced with the challenges of cloud-only sharing solutions where data storage is remote and access speed slow, Cloud and edge computing are essential for smart and efficient healthcare systems in smart cities. Our framework will become a decentralized edge computing solution that will allow rapid access to resources due to the proximity of data and consumers.

6 Conclusions

In this article we present our work around GRAPP&S (GRid Applications and Services), a multi-scale architecture for data aggregation and services. We present the main features of GRAPP&S, such as the specification of its components, connection and disconnection mechanisms, indexing operations, research and access to data, as well as the principles recommended for the coordination of

knots. Based on the principles of multi-scale systems GRAPP&S was designed as a hierarchical network based around the concept of "communities", which allows the integration of information sources with heterogeneous access protocols and various security rules. In addition, the use of Data Managers, specialized proxies for the adaptation and processing of different types of data, it is possible to transparently integrate both file type data as well as databases, flows (audio, video), web services and distributed computation. GRAPP&S is a work in progress, including a prototype in development course will be used for passing tests scale and as a platform for advanced services such as security grids [7] and storage optimization with the GAIA scheduler, introduced in this article.

References

1. Rottenberg, S., Leriche, S., Taconet, C., Lecocq, C., Desprats, T.: MuSCa: a multiscale characterization framework for complex distributed systems (2014)
2. Chalopin, J., Godard, E., Métivier, Y., Ossamy, R.: Mobile agent algorithms versus message passing algorithms. In: Shvartsman, M.M.A.A. (ed.) OPODIS 2006. LNCS, vol. 4305, pp. 187–201. Springer, Heidelberg (2006). https://doi.org/10.1007/11945529_14
3. Satyanarayanan, M.: Mobile computing: the next decade. SIGMOBILE Mob. Comput. Commun. Rev. **15**, 2–10 (2011)
4. Gassara, A., Rodriguez, I.B.: Describing correct deployment architectures based on a bigraphical multi-scale modeling approach. Comput. Electr. Eng. **63**, 277–288 (2017)
5. Jiang, C., Gao, L., Duan, L., Huang, J.: Scalable mobile crowdsensing via peer-to-peer data sharing. IEEE Trans. Mob. Comput. **17**(4), 898–912 (2018)
6. Lee, D., Park, N., Kim, G., et al.: De-identification of metering data for smart grid personal security in intelligent CCTV-based P2P cloud computing environment. Peer-to-Peer Netw. Appl. **11**, 1299–1308 (2018)
7. Bok, K., Kim, J., Yoo, J.: Cooperative caching for efficient data search in mobile P2P networks. Wirel. Pers. Commun. **97**, 4087–4109 (2017)
8. Huang, X., Qin, Z., Liu, H.: A survey on power grid cyber security: from component-wise vulnerability assessment to system-wide impact analysis. IEEE Access **6**, 69023–69035 (2018)
9. Jeanneau, D., Rodrigues, L., Arantes, L., et al.: An autonomic hierarchical reliable broadcast protocol for asynchronous distributed systems with failure detection. J. Braz. Comput. Soc. **23**, 15 (2017)
10. Roy, D.G., De, D., Mukherjee, A., et al.: Application-aware cloudlet selection for computation offloading in multi-cloudlet environment. J. Supercomput. **73**, 1672–1690 (2017)
11. Rowstron, A., Druschel, P.: Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems. In: IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), pp. 329–350, November 2001
12. Fraigniaud, P., Gavoille, C.: Routing in trees. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 757–772. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-48224-5_62
13. Bhatia, M., Manral, V., Ohara, Y.: IS-IS and OSPF Difference Discussion. IETF Internet Draft, January 2006
14. Satyanarayanan, M., Bahl, P., Caceres, R., Davies, N.: The case for VM-based cloudlets in mobile computing. IEEE Pervasive Comput. **8**, 14–23 (2009)