Support for resource aggregation in collaborative P2P systems

Damian Arellanes, Sonia Mendoza
Centro de Investigación y de Estudios Avanzados del
Instituto Politécnico Nacional - CINVESTAV-IPN
Departamento de Computación
Email: {darellanes,smendoza}@cs.cinvestav.mx

Dominique Decouchant
Universidad Autónoma Metropolitana - UAM Cuajimalpa
División de Ciencias de la Comunicación y Diseño
Departamento de Tecnologías de la Información
Email: decouchant@correo.cua.uam.mx

***Abstract*—In the last years, usage alternatives of P2P systems have been proposed, in which nodes provide resources and use resources of others in a collaborative way, in order to accomplish high scale tasks. Such systems are called "Collaborative P2P Systems" and have an important role in resource aggregation: advertisement, discovery, selection, matching, and binding phases. However, any toolkit does not allow developers to incorporate resource aggregation support into such systems using a specific programming language. In addition, existent solutions do not achieve cohesion among the key phases of resource aggregation (selection, matching, and binding). The main contribution of this paper is the design and implementation of a novel support that provides cohesion among such key phases and allows developers to aggregate multi-attribute, single-attribute, dynamic, static, and heterogeneous resources into collaborative P2P systems. The proposed Java-based support follows an unstructured topology based on super-peers, which advertise, select, match, and bind resources on behalf of their peers, reducing the network traffic. To validate our proposal, we developed a collaborative P2P application that generates the Mandelbrot set by means of the collaboration of several heterogeneous resources.**

I. Introduction

P2P systems are distributed systems without central control. Thus, their nodes are autonomous and act as client and server [1]. The nodes of a P2P system have a common purpose, although they may have different capabilities (e.g., memory, processing, and bandwidth). Therefore, a P2P system can be defined as a "collective intelligence", in which each component node carries out specific functions, according to its capabilities, in order to achieve a common goal. In recent years, usage alternatives of P2P systems have been proposed, e.g., each node offers its own resources and uses the resources of the other nodes, so that a form of resource collaboration is established among the nodes. The term "resource collaboration" refers to a process composed of seven phases: advertisement, discovery, selection, matching, binding, use, and release [2]. Excepting from the last two phases, which depend on the specific purpose, most of these phases can be specified in a generic way.

In order to accomplish a large-scale task, collaborative P2P systems aggregate diverse resources such as processor cycles, storage, bandwidth, sensors and actuators, middleware, scientific algorithms, software, and (Web and Cloud) services. In a similar way, these systems share data that consumes, generates, modifies, and manages a variety of contents [2]. Collaborative P2P systems are applicable to a wide range of contexts, such as Grid, Cloud, opportunistic computing, sensor networks, mobile social networks, and emergency management [3].

Current P2P protocols have focused on gathering groups of distributed resources, in order to achieve tasks, which are difficult to be accomplished using the resources of a single node. These protocols are based on resource aggregation, which is composed of the following phases: advertisement, discovery, selection, matching, and binding (i.e., a subset of the resource collaboration process). In particular, the last three ones are the key phases, so they have to maintain a cohesion degree in order for the effectiveness of resource aggregation to be high. The advertisement and discovery phases can be carried out in an independent manner, since the phases of resource aggregation do not follow a sequential order (excepting for the key phases). Therefore, some P2P protocols (e.g., Mercury [7]) do not implement the advertisement or discovery phases.

Although resource aggregation can be executed in P2P systems, it is not exclusive to them, e.g., there are centralized platforms for resource aggregation, such as GENI [4]. Therefore, resource aggregation can be carried out using centralized or distributed approaches, but the former one presents an important failure point: the central node. On the other hand, most of the P2P protocols for resource aggregation just implement a subset of the mentioned phases. In addition, it is not trivial to discover, aggregate and use dynamic, distributed, and heterogeneous resources: thus, efficient solutions to address this problem must be developed [2]. Collaborative P2P systems have an important role in resource aggregation, but it does not exist any toolkit that allows developers to incorporate resource aggregation support in such systems using a specific programming language.

In this paper, we propose a novel Java-based toolkit that facilitates the aggregation of heterogeneous, distributed, dynamic, and multi-attribute resources by using an unstructured P2P architecture based on super-peers. Our proposal is fully decentralized, presents high resilience, provides cohesion among the key phases of resource aggregation, and facilitates the development of collaborative P2P systems for highly dynamic environments. Our toolkit exploits the storage, processing, communication, and displaying capabilities of all nodes, in such a way nodes with better capabilities manage the system. To validate the toolkit functionality, we developed a collaborative P2P system that generates the Mandelbrot set by allowing several heterogeneous and distributed resources to collaborate.

The rest of this paper is organized as follows. After analyzing related work in section 2, we present the architecture and design of the proposed support in section 3. Then, in section 4, we detail the implementation and validation of the functionality provided by such a support. Finally, in section 5, we conclude this work and propose some future extensions.

II. Related work

Nowadays, few systems provide a support for resource aggregation but any of them neither offer cohesion among the key phases of this process nor can be easily integrated into an application using a specific programming language.

The GENI (Global Environment for Network Innovations) platform [4] allows users to carry out experiments related to the design and evaluation of protocols, the integration of social networks, the management of contents, and the development of network services. GENI aggregates resources (e.g., processing, storage, and sensors) from multiple management domains. A central repository sends a list of potentially useful resources to the user. The GENI API allows resource providers to advertise their capabilities in aggregates by means of XML documents, while experimenters are allowed to discover, select, use, and release resources [4]. However, GENI has several limitations in the development of collaborative P2P systems:

1) The user can ask for a set of arbitrary resources, relying on their instincts or in the *least effort principle* (they do not consider application requirements) [3].
2) The matching phase is not considered, so not all the combinations of selected resources are able to work together [3]. Therefore, GENI does not provide cohesion among the key phases of resource aggregation.
3) GENI acquires distributed users and resources, so a central repository is insufficient and susceptible to failures [3].
4) GENI does not offer means to create personalized zero-attribute, single-attribute, and multi-attribute queries.
5) GENI does not support the development of applications that need to aggregate resources using a specific programming language.

SWORD [5] is a resource aggregation service for distributed systems that allows users to describe and aggregate resources using a topology of interconnected groups. SWORD collects reports of static and dynamic resources available in provider nodes and allows users to select resources by specifying groups of required resources, restrictions per node (e.g., available memory and storage space), restrictions among nodes (e.g., latency), and restrictions among groups (e.g., bandwidth) as well as penalties caused by unsatisfied requirements.

SWORD offers both centralized and distributed versions. The centralized version is integrated in the PlanetLab infrastructure as a discovery tool. The distributed version is specified theoretically and follows a structured P2P architecture based on a ring divided in regions (each one corresponds to a range of attribute values). SWORD just provides support for the following phases: advertisement, selection, and matching of static, dynamic, multi-attribute, and single-attribute resources. Report nodes are resource providers, which advertise their attributes using XML documents. However, SWORD also has limitations in the development of collaborative P2P systems:

1) Popular queries in real workloads ask for dynamic resources with a wide range of attribute values [6], but SWORD is only applicable in semi-dynamic environments (e.g., Grid computing) where resources do not change frequently [2].
2) SWORD does not provide the binding phase required by collaborative P2P systems to ensure resources availability. Therefore, it does not offer cohesion among the key phases of resource aggregation.
3) In the distributed version of SWORD, provider nodes send attribute updates every two seconds, so it is not useful when attributes change very frequently.
4) In the distributed version of SWORD, attributes in real workloads [6] force it to use several segments of its ring, while others are rarely used [2].
5) There is not a support that facilitates the integration of SWORD into collaborative P2P systems, using specific programming languages.

On the other hand, there exist structured P2P protocols (e.g., Mercury [7], LORM [8], and MAAN [9]) and unstructured ones (e.g., flooding, goosiping, and random walks) [2]. These protocols facilitate the development of resource aggregation applications to some extent. Even, it is possible to combine these protocols, in order to create a new one. Most of the structured protocols rely on Distributed Hash Tables and their performance is good in latency-sensible systems (e.g., CASA [10] [11]). Such protocols can work on semi-dynamic environments. However, unstructured P2P protocols accomplish a bigger number of resource aggregation phases than the structured ones, and they work on highly dynamic environments (e.g., mobile social networks). Unlike structured protocols, the unstructured ones cannot be deployed on high scale systems (e.g., Cloud computing) [2].

III. Architecture and design

The proposed support follows an unstructured architecture based on super-peers, which allows to:

1) Monitor the Resource Specifications[1] of their peers.
2) Advertise, discover, select, match, and bind resources on behalf of their peers.
3) Increase the speed of queries, while reducing the inherent latency in content searches.
4) Reduce the cost of the resource advertisement, discovery, and selection phases.
5) Track relations among the resources managed by their peers, in order to facilitate the matching phase.
6) Resolve real queries with moderate costs [3].
7) Reduce network traffic, since they are the only involved entities in communication processes.

In particular, we use the Myconet protocol [12] to create and maintain the unstructured topology based on super-peers. This protocol allows to create auto-configurable and fault-tolerance P2P systems. Myconet groups nodes, giving priority to the peers with more computing capabilities, which can result in better performance and rise in the convergence rate (i.e., the time required to obtain an optimal configuration). Myconet constructs and mantains a strongly interconnected

[1]Resource Specification is a document in which a node advertise its resources along with their attributes, and usage restrictions.

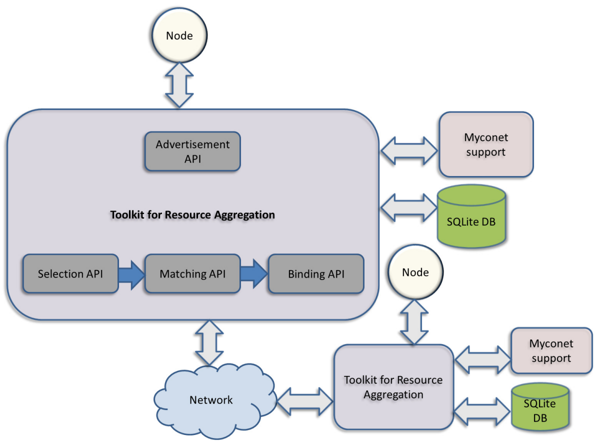

Figure 1. Resource Aggregation Toolkit

and decentralized super-peer overlay; unlike other unstructured approaches, dynamically adjusts the connections between super-peers, increasing efficiency and resilience to the loss of peers. Myconet outperforms SG-1, ERASP and many other super-peer protocols [12]. Biologically-inspired models have been lately the subject of active research, as they hold promise to enable desirable properties (e.g., resilience and self-organization) for large-scale and distributed systems.

The proposed toolkit, implemented in Java, provides four APIs for carrying out resource aggregation: advertisement, selection, matching, and binding. Fig. 1 illustrates these APIs and their function in a P2P environment. Each node has a SQLite database, which stores and manages information (e.g., statistic list and resource specifications).

In the following subsections, we detail the components of our approach based on super-peers (see subsection III-A) as well as each one of the resource aggregation phases provided by our support (see subsections III-B to III-E).

A. Topology based on super-peers

The proposed topology has two types of nodes: super-peer and normal peer. A super-peer carries out requests on behalf of the peers it is in charge of, by selecting, matching, and binding resources. A normal peer asks its super-peer about needed resources and receives answers from it. Super-peers are arbitrarily connected by means of directed edges, so it is not necessary that all of them are inter-connected one another.

Let S denote the set of super-peers, E be the set of directed edges, and C denote the set of collaborative P2P supports. The architecture of the collaborative P2P support $c \in C$ is defined as a graph $G = (S_c, E_c)$, where S_c and E_c are respectively the sets of super-peers and directed edges of the support c, such that $S_c \subset S$ and $E_c \subset E$. On the other hand, let SP_i and SP_j be two super-peers, such that $SP_i, SP_j \in S_c$, and e denote a directed edge, such that $e \in E_c$. SP_i and SP_j are neighbors in the P2P network of c if and only if $e = (SP_i, SP_j)$. Super-peers maintain a routing table to their corresponding normal peers and other super-peers, while normal peers have a routing table to the super-peer that is in charge of them.

In Myconet, super-peers and normal peers are characterized by an integer value that denotes their capacity degree to satisfy the needs of other peers. Such an abstraction may vary depending on the application purpose. In our proposal, the capacity of super-peers and normal peers is defined in terms of storage, processing, and bandwidth, since the matching phase may need high processing capacity, while the selection phase may require good bandwidth among super-peers, which in turn need to have a good storage capacity to manage the operations of their normal peers.

Myconet uses a set of local rules that determine the behavior of the states of a super-peer defined in a hierarchical way as: extending, branching, and immobile. Thus, the upper state (immobile) corresponds to a super-peer with good capacities, availability, and stability in comparison with the rest of the nodes [12]. Two parameters are necessary: the number of connections from a super-peer to a set of normal peers, which is proportional to the capacity of such a super-peer, and the number of links from a super-peer to other super-peers, which is a personalized parameter. Dissemination of information about living peers and their respective states is carried out by the Newscast protocol using the goosiping technique.

B. Advertisement phase

A normal peer advertise its resources to its super-peer via a XML document that contains resource specifications. To carry out this phase, a normal peer sends to the super-peer an advertisement agent (a resource specification) with a cost of $O(1)$. Resource specifications describe resources characterized by multiple heterogeneous, static, and dynamic attributes. For each resource, a set of attributes and their values (e.g., so="linux" and memory=8GB) as well as their usage restrictions (e.g., availability=[4:00AM-13:00PM]) are specified. Since dynamic attributes change frequently under real workloads [6], normal peers constantly inform their super-peer of updates. Thus, there exist two types of advertisement agents: the initial one and the updating one.

Let SP_f and SP_g be two super-peers, such that $SP_f \in S_c$, $SP_g \in S$ and $SP_g \notin S_c$. If SP_f abandons the P2P network of the support c (voluntarily or due to a failure) or reduces in rank (since it does not fulfill the Myconet rules), the pertinent rules are applied according to its current state (extending, branching, or immobile), in order for a new super-peer (SP_g) to replace it. In the former case (i.e., when SP_f abandons the P2P network of c), the support c auto-organizes and the normal peers of SP_f advertise the new super-peer SP_g their resource specifications. In the latter case (i.e., when SP_f reduces in rank), SP_f transfers its statistic list and their normal peers to the new super-peer SP_g. In both cases, SP_g will become a member of S_c, while SP_f will stop belonging to S_c.

C. Selection phase

Resource queries are also specified in XML documents and ask for groups of nodes. Queries can be zero-attribute, single-attribute, or multi-attribute. For each group, a query specifies the group name, the number of required nodes, the attributes required per node, and the restrictions among nodes. Restrictions among groups are also specified in the query.

The range of values for the specified attributes can be integers (e.g., a range of needed cores) or float (e.g., a range of required latency). It is also possible to specify attributes of string type (e.g., operating system). For integer or float attributes, the query defines a range of allowed values (min_val and max_val), a range of ideal values (min_ideal and max_ideal), and a penalty value[2] (P). By contrast, for each attribute of string type, only the required value and the penalty P need to be specified.

A normal peer or a super-peer can need a resource query, but in any case the super-peers are the query originators, so they send query agents transporting a XML document that specifies a query. Query agents visit super-peers using a technique that depends on the specific purpose of the application. In fact, an application can either need good performance or compromise performance in order to always obtain a set of results in the selection phase. Thus, the proposed support allows us to specify one of these two options: to imperatively find resources ($requires : find\_resources$) or to improve performance even if resources are not always found ($requires : performance$). In the former option, all the super-peers are flooded with query agents while, in the latter option, we use a technique called "intelligent walks", which is described below.

Every query agent is sent in the P2P network of c with a limited TTL (Time To Live). Each super-peer maintains a statistic list of solved queries, in which each entry includes the identifier of the solver super-peer and the query *per se*. Let $SP_{initiator}$ denote the originator of a query, such that $SP_{initiator} \in S_c$. Before sending a query agent, $SP_{initiator}$ uses a similarity metric to determine whether, in its list, there exists a similar query to the requested one. $SP_{initiator}$ sends an agent to each one of the super-peers from the list, whose solved queries have a high similarity (greater than 75%); if any query from the list is not similar (lesser or equal to 75%), $SP_{initiator}$ sends a query agent to n super-peers (neighbors of $SP_{initiator}$) randomly selected (where n is a configurable parameter of the proposed support).

Let VN_c be the set of super-peers visited by a query agent in the P2P network of the support c, such that $VN_c \subseteq S_c$, SP_v denote a super-peer visited by a query agent, such that $SPv \in VN_c$ and $SP_v \neq SP_{initiator}$, NB_v be the set of super-peers that are neighbors of SP_v, such that $NB_v \subseteq S_c$, and SP_w denote a neighbor of SP_v, such that $SP_w \in NB_v$. Each visited super-peer that receives the agent forwards it to one and only one of their super-peers neighbors randomly chosen. SP_v decreases the TTL value before forwarding the query agent, in order to reduce bandwidth (preventing from sending an agent, which TTL has expired). If SP_v can solve the query, it sends a response and updating message to $SP_{initiator}$, in order for the latter to update its statistic list. This process terminates when the TTL expires, when SP_v does not have anymore neighbors, or when a termination condition specified in the query agent has been fulfilled (e.g., stop the process if the first result has been found). Before sending an agent to SP_w, such that $SP_w \neq SP_{initiator}$ and $SP_w \notin VN_c$, SP_v verifies that SP_w is still alive and in case it is not, SP_v selects another super-peer from its neighbors. In this way, the proposed support guarantees that query agents will not be lost on the way.

D. Matching phase

The computing load of this phase is distributed between $SP_{initiator}$ and each $SP_v \in VN_c$. The goal of this phase is to determine which are the best sets of nodes that satisfy the requirements of a query. Let NP_v denote the set of normal peers of the super-peer SP_v, Q_c denote the set of queries in the support c, G_q denote the set of attribute groups specified in a query $q \in Q_c$, CG_v be the set of candidate groups determined by SP_v for the query q, such that $\mathbf{card}(CG_v) \leq \mathbf{card}(G_q)$, and A_g denote the set of attributes specified in a group g, such that $g \in G_q$. Thus, each $SP_v \in VN_c$ returns its CG_v to $SP_{initiator}$, if and only if $CG_v \neq \emptyset$. To determinate whether a node $x \in \{SP_v\} \cup NP_v$ is a candidate node for the group g, its penalty is calculated by each attribute $a \in A_g$.

Equation (1) defines the penalty P_x^a of the node x for an attribute a of any type (integer, double or string):

$$P_x^a = P_g^a \delta_g^a \tag{1}$$

where P_g^a is the penalty specified for the attribute a in the group g, and δ_g^a is the deviation of a in g.

Equation (2) defines particular cases for the value of δ_g^a (deviation measured in native unities of an attribute, e.g., GB for free space in hard disk, or miliseconds for latency) in an attribute a of the integer or double type specified in the group g, while the equation (3) defines particular cases for the value of δ_g^a in a of the string type also specified in g:

$$\delta_g^a = \begin{cases} 0 & \text{if } val_x^a = val_g^a \\ \infty & \text{otherwise} \end{cases} \tag{3}$$

where val_x^a is the value of the attribute a of the node x and val_g^a is the value of a specified in the group g.

The total penalty of the node x for the group g denoted as P_x^g is defined by equation (4):

$$P_x^g = \sum_{i=1}^{k} P_x^i \tag{4}$$

where k is the number of attributes specified in g and P_x^i is the penalty of the node x for the attribute i of any type.

The node x does not satisfy the requirements specified in g, if and only if P_x^g is infinite. In this case, the node is omitted; otherwise, it is added to the pertinent group in CG_v.

Once $SP_{initiator}$ has received CG_v from each $SP_v \in VN_c$, it executes a dynamic programming-based algorithm to generate all the possible combinations of candidate nodes for each group $g \in G_q$. A list L_g of node combinations that can work together (node candidates), and another list M_g of node combinations that cannot work together are created for the group g. Then, the following operations are performed:

Let C_g be the set of node combinations[3] for the group g, and NC_g denote a node combination for g, such that $NC_g \in C_g$.

1) For each NC_g:
   From M_g, it is determined whether it is possible for the nodes in NC_g to work together. If not, NC_g is aggregated to M_g; otherwise the following operations are carried out:

[2] P is the penalty when the range of ideal values is not satisfied.

[3] Each element in C_g is a set of nodes and each one of these is a node combination.

$$\delta_g^a = \begin{cases} min\_ideal - min\_val & \text{if } min\_val \leq val_x^a < min\_ideal \\ max\_val - max\_ideal & \text{if } max\_ideal < val_x^a \leq max\_val \\ 0 & \text{if } min\_ideal \leq val_x^a \leq max\_ideal \\ \infty & \text{otherwise} \end{cases} \quad (2)$$

where min_val and max_val are respectively the min and max values of the specified range in the group g for the attribute a, min_ideal and max_ideal are respectively the ideal min and max specified values in g for a, and val_x^a is the value of a of the node x.

- A partial penalty PP of NC_g is calculated when it does not satisfy the number of nodes specified for the group g, as follows: $PP = n - \mathbf{card}(NC_g)$, where n is the number of nodes specified for the group g, and $\mathbf{card}(NC_g)$ is the number of elements in NC_g.
- Let R_g be the set of restrictions between nodes specified for the group g, such that $\mathbf{card}(R_g) = m$. The total penalty TP of NC_g denote the penalty when it does not satisfy the restrictions among nodes and the penalty per restriction $r \in R_g$ (denoted as P_r^i) is calculated in a similar way as equation (1), therefore: $TP = (\sum_{i=1}^{m} P_r^i) + PP$.
- NC_g is aggregated to L_g.

2) For each group $g \in G_q$, the nodes contained in L_g are ordered according to their corresponding total penalty.
3) Let T_q be the set of restrictions between groups specified in the query q, such that $\mathbf{card}(T_q) = p$. Similarly to step 1, but instead of forming groups of candidate nodes, $SP_{initiator}$ creates a set of groups of candidate node groups denoted as CG_{final}, which satisfy restrictions among groups. The penalty per restriction $t \in T_q$ (denoted as P_t^i) is calculated in a similar way as equation (1). Each element in CG_{final} has a penalty, denoted as P_{groups}, when it does not satisfy the restrictions among groups, whose equation is similar to equation (4): $P_{groups} = \sum_{i=1}^{p} P_t^i$
4) $SP_{initiator}$ selects the element of CG_{final} presenting the lowest value of penalty P_{groups} denoted as B_q (i.e., the best candidate groups for the query $q \in Q_c$), such that $B_q \in CG_{final}$.

E. *Binding phase*

Once the best candidate groups (B_q) have been selected, it is necessary to be sure that the found resources are available for usage. Due to churn or failures, the resources cannot be available when it is required to use them [2]. In order to carry out this phase, $SP_{initiator}$ tests the state of each node in each group $b \in B_q$; to do this task, it sends a ping message to each one of such nodes. If one of them is available, $SP_{initiator}$ sends it a binding agent that carries a XML document for a resource request. This document contains the needed resources. The available node determines whether it is possible to use such resources or not; if so, it returns an acceptation message to $SP_{initiator}$; otherwise, the available node is eliminated from b. When this phase terminates, $SP_{initiator}$ sends B_q to the normal peer that has requested resources (if needed).

IV. Implementation and evaluation

In order to validate the functionality of the toolkit, we implemented a collaborative P2P application, whose goal is to generate the Mandelbrot set. This application uses the proposed toolkit to aggregate resources and is composed of a master node, some processing nodes and a display node. The master node is in charge of determining the display node and distributing the tasks among the processing nodes. These are responsible for computing a fragment of the Mandelbrot set and communicating their results to the display node, which in turn gathers and shows such results on the screen (see fig. 2).

Listing 1. Query with two groups: processing and display

```
<query>
    <option>requires: find_resources</option>
    <group>
        <name>Processing</name>
        <num_nodes>8</num_nodes>
        <cpu_speed>1024.0,2500.0,4096.0,4096.0,0.2</cpu_speed>
        <free_mem>512.0,1024.0,4096.0,8192.0,0.05</free_mem>
        <busy_cpu>0,0,30,75,0.05</busy_cpu>
        <cores>1,2,4,4,0.03</cores>
        <free_hdisk>50.0,60.5,92.5,100.0,0.05</free_hdisk>
        <os_name>"Linux",0.0</os_name>
    </group>
    <group>
        <name>Display</name>
        <num_nodes>1</num_nodes>
        <screen_width>800,1024,1366,2560,0.02</screen_width>
        <screen_height>600,1024,2048,1700,0.02</screen_height>
        <bit_depth>8,16,48,48,0.0</bit_depth>
        <refresh_rate>40,60,120,120,0.05</refresh_rate>
    </group>
    <rest_betw_groups>
        <group_names>Processing, Display</group_names>
        <latency>0.0,0.0,50.0,100.0,0.2</latency>
    </rest_betw_groups>
</query>
```

In resource queries, the $requires : find\_resources$ option is specified (a flooding-based technique that employs query agents) since it is required that query agents always find results; in addition, the validation application is a low scale collaborative P2P system (12 nodes), so the network bandwidth consumption is insignificant. The resource aggregation phases are carried out in the following way:

1) Advertisement: normal peers explicitly advertise their resources by means of resource specifications. These nodes constantly send resource updates to their super-peers.
2) Selection: the super-peer in charge of the master node selects a group of multi-attribute resources and specifies a resource query requesting two types of node groups (see Listing 1): a processing group and a display group. In the former group, 8 nodes with 6 attributes (CPU speed, free memory, busy CPU, number of processor cores, free hard disk, and operating system) are specified; restrictions among nodes are not specified since inter-node comunication is not established. In the latter group, just one node with 4 attributes (screen width, screen height, bit depth, and refresh rate) is specified. Like in the former group,

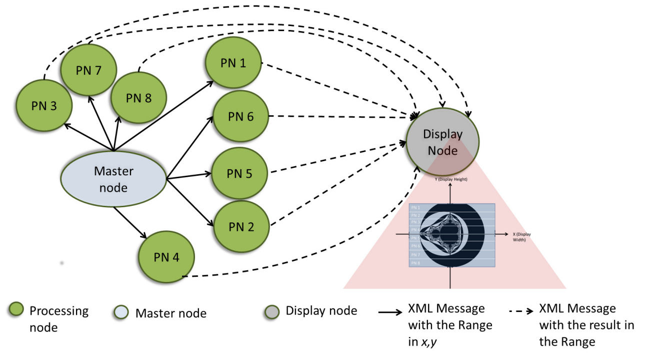

Figure 2. Message Flow within the Collaborative P2P System.

"latency" is specified as a restriction among groups, since the nodes of the former group have to communicate with the node of the latter group.

3) Matching: the super-peer originator of the query executes the matching algorithm using the results of the selection phase as input. This phase give us the best node for the display task and the best nodes for the processing task.
4) Binding: the super-peer originator of the query determines whether the resulting nodes (fulfilling the requirements and restrictions established by the master node) are available for usage.

As mentioned in the Introduction, the use and release phases are specific to each application. In particular, in our validation application the master node divides the plane x, y in rows and distributes tasks based on the results provided by the super-peer originator of the query. The processing nodes use the "escape time" algorithm to process the assigned range; they directly communicate with the display node. Each time a consumer node finishes using the resources of a provider node, release the resources of this later one.

V. Conclusions and future work

The proposed support is constructed on an unstructured P2P architecture based on super-peers, which allows us to aggregate distributed resources (characterized by static, dynamic, and heterogeneous attributes) in collaborative P2P systems. Using a topology of super-peers offers more resilience, so that it can auto-configure dynamically. The proposed support also provides cohesion among the key phases of resource aggregation and takes into account the application requirements, so it offers Quality of Service (QoS).

Even if unstructured architectures work well on highly dynamic environments, they are not suitable for high scale systems. For this reason, we expect in the near future extent our proposal, in order to also facilitate the development of collaborative P2P systems following structured architectures. Moreover, it is possible to define logical conditions and (&), or (|), and not (!) for each attribute specified in a query (e.g., so="linux" & "windows"). This facility is an additional contribution, since any related work does not allow users to carry out queries containing logical conditions. On the other hand, we can design a resource aggregation middleware from the proposed support that facilitates even more the development of collaborative P2P systems. Resource aggregation in collaborative P2P systems is a relatively new subject, so some problems remain open. We believe that such systems will be very important in near future owing to their wide-scope application and to the evolution of computing devices. Therefore, resource aggregation in mobile computing can become a research subject, whose solutions could provide several benefits, e.g., exploiting the resources of the devices belonging to all the humans, in order to carry out scientific supercomputing.

Our proposal facilitates the development of collaborative P2P systems by means of Java. We have omitted the discovery phase, since resources are explicitly advertised with a cost $O(1)$. Resource queries can be solved as long as the flooding technique is used. However, when bandwidth needs to be saved, it is possible to use the intelligent walks technique, although it does not guarantee the obtaining of results in the selection phase.

References

[1] [Tanenbaum and Steen, 2007] A. S. Tanenbaum and M. V. Steen, *Distributed Systems Principles and Paradigms*, 2nd Edition, Prentice Hall, USA, 2002.

[2] [Bandara and Jayasumana, 2013] H. M. N. Dilum Bandara and A. P. Jayasumana, *Collaborative Applications over Peer-to-Peer Systems - Challenges and Solutions*, Peer-to-Peer Networking and Applications, Vol. 6, No. 3 , pp. 257-276, 2013.

[3] [Bandara, 2012] H. M. N. Dilum Bandara, Enhacing collaborative Peer-to-Peer systems using resource aggregation and caching: a multi-attribute resource and query aware approach, Doctoral Thesis, Colorado State University, Fort Collins, Colorado, Department of Electrical and Computer Engineering, Fall 2012.

[4] [Berman et al., 2014] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhurif, R. Ricci and I. Seskar, *GENI: A federated testbed for innovative network experiments*, In Computer Networks, Vol. 61, pp.5-23, March 14, 2014.

[5] [Oppenheimer et. al, 2008] D. Oppenheimer, J. Albrecht, D. Patterson and A. Vahdat, *Design and implementation tradeoffs for wide-area resource discovery*, ACM Transactions on Internet Technology (TOIT), vol. 8, no. 4, September, 2008.

[6] [Bandara and Jayasumana, 2012] H. M. N. Dilum Bandara and A. P. Jayasumana, *Evaluation of P2P resource discovery architectures using real-life multi-attribute resource and query characteristics*, In Consumer Communications and Networking Conference (CCNC '12), pp. 634-639, Las Vegas, Nevada, USA, June 14-17, 2012.

[7] [Bharambe, Agrawal and Seshan, 2004] A. R. Bharambe, M. Agrawal, and S. Seshan, *Mercury: Supporting scalable multi-attribute range queries*, In Proc. ACM Special Interest Group on Data Communication (SIGCOMM' 04), August/September, 2004.

[8] [Shen, Apon and Xu, 2007] H. Shen, A. Apon, and C. Xu, *LORM: Supporting low-overhead P2P-based range-query and multi-attribute resource management in grids*, In Proc. 13th Int. Conf. on Parallel and Distributed Systems, December, 2007.

[9] [Cai, Frank, Chen and Szekely, 2004] M. Cai, M. Frank, J. Chen and P. Szekely, *MAAN: A multi-attribute addressable network for grid information services*, Journal of Grid Computing, January, 2004.

[10] [Bandara and Jayasumana, 2012] H. M. N. Dilum Bandara and A. P. Jayasumana, *Adaptive Sensing of Atmosphere: State of the art and research challenges*, In Globecom Workshops (GC Wkshps), pp. 1378-1383, Anaheim, CA, USA, December 3-7, 2012.

[11] [Lee et al, 2012] P. Lee, A. P. Jayasumana, H. M. N. D. Bandara, S. Lim and V. Chandrasekar, *A peer-to-peer collaboration framework for multi-sensor data fusion*, J. of Network and Computer Applications, vol. 35, no. 3, pp.1052–1066, May, 2012.

[12] [Snyder, Greenstadt and Valetto, 2009] P.Snyder, R. Greenstadt and G. Valetto, *Myconet: A Fungi-Inspired Model for Superpeer-Based Peer-to-Peer Overlay Topologies*, In Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO '09), pp. 40-50, September 14-18, 2009.