A Self-organizing and Self-configuration Algorithm for Resource Management in Service-oriented Systems

Javad Zarrin
Instituto de Telecomunicações
3810-193 Aveiro, Portugal
Email: javad@av.it.pt

Rui L. Aguiar
Universidade de Aveiro
3810-193 Aveiro, Portugal
Email: ruilaa@ua.pt

João Paulo Barraca
Universidade de Aveiro
3810-193 Aveiro, Portugal
Email: jpbarraca@ua.pt

***Abstract*—With the ever increasing deployment of service-oriented distributed systems in large-scale and heterogeneous computing environments, clustering and communication overlay topology design has become more and more important to address several challenging issues and conflicting requirements, such as efficient scheduling and distribution of services among computing resources, reducing communication cost between services, high performance service and resource discovery while considering both inter-service and inter-node properties and also increasing the load distribution and the load balance. In this paper, a four-stage hierarchical clustering algorithm is proposed which automates the process of the optimally composing communicating groups in a dynamic way while preserving the proximity of the nodes. The simulation results show the performance of the algorithm with respect to load balance, scalability and efficiency.**

***Keywords*—Self-configuration, Self-adaptation, Service-oriented Systems, Self-deployment, Resource Discovery, Resource Management**

I. Introduction

Large-scale service-oriented systems' lunch on the infrastructure contains thousands of processors (computing resources) distributed across multiple clusters. The services must collaborate with each other in order to achieve a common goal without relying on any centralized control. Such systems are subject to dynamicity and scalability. The distribution of services based on centralized or hierarchical architecture will raise the bottleneck and single point of failures.But employing a fully-distributed system has its drawbacks too. Furthermore, the service distribution and allocation among a large set of dynamic resources would become too complex to be managed by using manual or statical configuration. Services need to become self- adaptive to maintain the entire system performance and functionality. Along this line, it is necessary to employ an efficient self-deployment and self-configuration mechanism which dynamically creates communicating groups of resources by building logical overlays on top of network topology. In this paper we present a multi-step, load-balanced, self-organized, clustering algorithm for overlay establishment (particularly for resource discovery) in distributed environments which preserve the locality of the nodes within the groups while it deals with efficiency and scalability.

The rest of this paper is organized as follows: Section II investigates the current related works. In section III we present our clustering algorithm and we explain the multi steps of the proposed grouping mechanism in hierarchical layers. In section IV we discuss the simulation and experiment results and finally in section V we present our conclusion.

II. Related Works

Over the past years, several group-based distributed systems have incorporated grouping and clustering mechanisms into their design as a way to tolerate the system growth, enhance the system performance and isolate faults. These approaches can be classified particularly for resource discovery into three main classes: quantity based (none-content based), quality based (content based) and hybrid clustering.

In quantity based clustering, overlay construction gets involved in the methods of organizing nodes/resources in the groups without considering the content (for example in terms of attributes, features, properties and behaviors) of nodes/resources. For this purpose, the focus might be on some performance related issues such as load balance, maintenance, self-organization, stability (churn and fault tolerance) while the overlay is constructed along the way to finally satisfy all the requirements of a resource discovery protocol which aims to run on top of it. To give some examples of quantity based clustering we could mention the following systems: In JXTA[1], a large number of nodes are decomposed for a set of manageable groups by introducing the concept of PeerGroups[2]. Shark[3] alters the nodes in the groups based on the common interest of the users. By the same token, Considine[4] creates multiple cluster-based overlays for Chord while considering the metrics such as network proximity between peers within the group. In super-peer systems such as [5] and [6], the less powerful nodes are clustered around super-nodes. This reduces the network overhead significantly by converting a costly all-to-all communication scheme into a more efficient hierarchical pattern.

On the other hand, the quality based clustering approaches organize groups based on the content similarity of each individual nodes/resources considering different aspects. According to the different views and definitions of the concept of the content similarity, there are several well-known approaches for content clustering which shape the discovery mechanisms in different directions. Proximity-aware[14], semantic-aware[15] and QoS-aware[16] resource discovery are the sample applications of content-based clustering(see TableI).

TABLE I. EXAMPLES OF CLUSTERING APPROACHES FOR RESOURCE DISCOVERY

Overlay Architecture	Mechanism	Clustering	Resource Discovery Examples
Semantic-Aware	The nodes/resources with similar contents are organized in the same cluster	Content-based Clustering, Semantic Overlay Network (SON)[7]	ERGOT[8]
Proximity-Aware	The physically nearby nodes/resources are organized in the same cluster	Content-based Clustering	TriPod [9], PIRD [10]
QoS-Aware	The nodes/resources with similar quality of service are organized in the same cluster	Content-based Clustering	CycloidGrid [11]
Load-Aware	The nodes/resources are organized in a particular overlay (e.g, DHT ring) in order to equally distribute the communication loads among nodes to enhance the overall system performance.	None-Content based Clustering, P2P overlay network (e.g., DHT based P2P)	PIAS[12] and ASTAS[13]
Super-Peer, Tree	The nodes/resources are organized in a particular overlay (e.g, tree or hierarchy) in order to enhance the overall system performance. (e.g. Load balanced Tree)	None-Content based Clustering, None-DHT based P2P Overlay	The works in [5], [6]

In hybrid clustering, the communicative groups are constructed based on a combination of both qualitative and quantitative characteristics and behaviors of the nodes in the system. The PeerCluster[17] is an example of hybrid clustering which groups the peers with similar interests using a hypercube structure to increase the querying efficiency. Cluster-K+[18] is another approach which uses a static tree structure to alter the network peers in the groups based on the possibility of subscribing to multiple content types for each peer (i.e., resource types) while rapidly being informed of content updates.

The aforementioned group-based systems have been proposed for different applications and research topics. To the extent of our knowledge there are few research works related to dynamic overlay construction in service-oriented systems. The work in[19] proposes a service-oriented architecture which could optimize the execution of the service workflows by enabling dynamic service grouping without considering resource grouping.

Furthermore, the works in[20] and [21] address two different aspect of self-deployment, self-adaptation and self-configuration of the services in service oriented architectures. [20] proposes a dynamic self-adaptation pattern that defines the way in which a set of components that make up an architecture or a design pattern dynamically cooperate to change the configuration of a service-oriented system to a new configuration through constructing overlay for the components. In the other work,[21], the authors propose a self-adapted autonomic ontology-based architecture which characterizes the properties of services participating in the autonomic collaborative environment.

We must note that none of these works address the problem of service distribution and resource mapping in their overlay construction proposals with regards to resource management issues. However, in this work, our focus is on providing an efficient resource-oriented mechanism for overlay construction in the presence of service oriented architectures.

III. ALGORITHM

In order to create and maintain groups and also support the query processing for the resource discovery, we implement a three-layer overlay which includes Leaf-Node(LN) Layer, Aggregate-Node(AN) Layer and the Super-Node(SN) Layer. Our algorithm contains multiple steps to establish and create a dynamic hierarchical overlay on top of the network topology.(see Figure1) In this section we explain these steps with regard to the following assumptions:

Assumption 1: We assume that the system is fully unstructured which means that the nodes do not know any information about each other. i.e., the nodes do not have any information about the network topology. Rather they only know about their local set of connection gates.

Assumption 2: The underlying topology is dynamic, with frequent changes. Furthermore the nodes characteristics are changing over time.

Furthermore, we define a set of terms (i.e., input parameters and criterias) which are used to structure our algorithm which could be additionally used to evaluate the algorithm's efficiency. These terms are as follows:

1) Group Diameter (G_d): It is the maximum number of hops between an aggregate node of a group and other leaf-node members within the group. (i.e., it is the maximum distance between an aggregate-node and a leaf-node within a group).
2) Locality Factor (k): It is a defined criteria to indicate the proximity of the nodes within a group. $k = \frac{1}{G_d}$, where k is the locality factor and G_d is the group diameter.
3) Neighboring Indicator (N_i): It clarifies the definition of the neighboring nodes through specifying the maximum number of hops between the source node and the potential neighbors.
4) Grouping Variables: They refer to the overlay design parameters (i.e., the validated range for the group's size in different layers of hierarchy) such as Maximum/Minimum number of allowed nodes per group in AN/SN layer.

Continuing this section we discuss the different steps of our algorithm. On the first step the nodes discover the underlying topology by recognizing its direct set of connecting neighbors. Afterwards, on the second step, in a time frame, nodes start to negotiate and exchange messages with each other in order to efficiently create the first level of the communicating groups which consists of specifying a group and a role (either leaf-node or aggregate-node) within the group for every one of the nodes

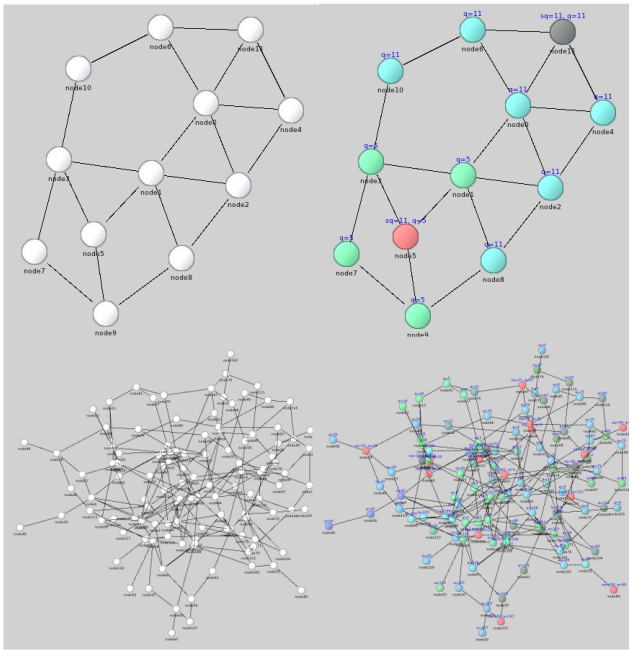

Fig. 1. Clustering overlay on top of the physical network topology

in the system. On the next step, an optimization algorithm is applied to enhance the results of the previous step by merging the groups that have their sizes below the required threshold. Finally, on the last step, the aggregate-nodes of each group must participate in a process to elect the super-nodes which leads to establishing the second level of the communicating groups.

Step 1: A node initializes the procedure by sending the ID-Request messages to all of its local connecting gates in order to obtain information about the possible neighbors. By default, the module-roles and the grouping variables (such as resource-id, qms-id and sqms-id) of all the nodes are unknown. The resource-id denotes the node's address (e.g., IP address) while the qms-id (aggregate-node's address) and sqms-id (super-node's address) specify the layer and the group that the node belongs to. Upon receiving ID-Request message at a destination, the receiving node records the sender information and sends its information to the requester through a message of type ID-Reply. Meanwhile, it checks its local information about the other neighbors and if there is still some missing information, it propagates the ID-Request messages to all of its connecting gates except the one that already knows about the other side of the connecting edge.

Step 2: Each node itself generates a random delay bounded in a specific time frame. Consequently, a WakeUP event is automatically triggered when the delay is over. Upon the occurrence of the event, the node checks its states and if the value of the module-role is still unknown it sends a message of type Join-Request to all its neighbors.(see Algorithm1) The information about the neighbors have already been collected in step 1. Depending on the states of the destination nodes, arrival of the Join-Request messages cause different reactions in the receiver nodes. When the module-role of the receiver node is unknown, it changes the current state of its module-role to LN (leaf-node) value while it also sets the value of its grouping variable for the first layer (i.e., qms-id) according to the address of the request sender. In addition to that it sends a message of type Join-Accept to the requester address. It means that the receiver node is ready to join a group which potentially can be created by the requester itself as the aggregate-node. In the case that the module-role state of a receiver node is LN, which means that this node already belongs to a group, it relays the request message to its assigned aggregate-node by forwarding the message to the address mentioned in the qms-id. Another possibility is that the receiver node of a Join-Request message is an aggregate-node (i.e., the node already belongs to a group and its module-role is AN). Each aggregate-node locally registers its LN members. Although, the number of LN members in a group is restricted to the maximum and minimum allowed size of the AN-Groups, it has to be clarified by the values of the overlay design input parameters. Thus, for each new joining request, the aggregate-node (as the receiver of a Join-Request message) must examine the possibility of adding the new member to the group and if it is feasible it must send an Update message to the original requester containing the information of the aggregate node saying that the requester is allowed to join the group otherwise the request message (i.e. the Join-Request message) is just ignored. However if the receiver node decides to ignore the request message, it caches the requester information as a potential remote aggregate-node for the purpose of super-node election on the later steps of the algorithm.(see Algorithm1, see also Figure2)

Algorithm 1 WakeUp,JRequest

```
Generate(WakeUp,RandomDelay)
On-WakeUp:
if local − node.state is unknown then
  for all remote − node IN neigbours − set(N_i) do
    sendMsg(JRequest, remote − node)
  end for
end if
On-JRequest:
switch (local − node.state)
case unknown:
  set local − node.state as LN
  set local − node.AN as msg.sender
  sendMsg(JAccept, msg.sender)
case LN:
forward(msg, local − node.AN)
default:
  if checkGP(GVars, k) > 0 then
    sendMsg(Update, msg.sender)
    AMembers.add(msg.sender)
  else
    Cache(msg.sender.info)
  end if
end switch
```

Generally, the most sensible reactions of the nodes to the Join-Request message are either Join-Accept or Update messages which specify the direction of the grouping from sender to the receiver or vice-versa. When a node receives a Join-Accept, it means that the node already is qualified by at least one node to establish a new group as an aggregate-node, however it might be possible that the node's state has been

changed by other parties during the time between issuing the Join-Request by the node and getting the Join-Accept from a particular node. Therefore the node behaves according to its current status to handle the incoming Join-Accept. So, if the current module-role is unknown, it changes its state to AN (aggregate-node), and a new group is created with at least two members (including the current node) and if its current state is LN, it simply forwards the Join-Accept message to its aggregate-node which is the representative of the group to make decision about the joining possibility of the new members. We must notice that this would happen only if the issuer of the Join-Accept is a node different from the local aggregate-node. In other cases, when the receiver of the Join-Accept is an aggregate-node, according to the group making policies, the AN receiver decides to add the Join-Accept issuer to its local group. It checks if the arrived message has been relayed from one of the current leaf-node members and if so, it sends a message of type Change to the original sender to update its grouping information for the new aggregate-node. Otherwise, the aggregate-node just records the information of the new member for the later query processing (e.g., resource discovery query). In other situations if the AN (as the receiver of a Join-Accept message) avoids admitting the potential new member (which is already reserved) to the group (based on the policies like group size or locality factor) it sends a message of type Failed-Group to the sender to make the sender node free by changing its status to unknown. In fact, when a node issues a Join-Accept, that node will be temporarily reserved for a potential grouping led by a particular aggregate-node. A Failed-Group message releases the reserved node.(see Algorithm2)

Algorithm 2 JAccept

```
On-JAccept:
switch (local − node.state)
case unknown:
    set local − node.state as AN
    set local − node.AN as local − node
    AMembers.add(msg.sender)
case LN:
if msg.sender is local − node.AN  then
    set local − node.state as AN
    set local − node.AN as local − node
else
    msg.flag.set(FJAccept)
    forward(msg, local − node.AN)
end if
default:
    if checkGP(GVars, k) > 0  then
        if msg.flag.get is FJAccept  then
            sendMsg(Change, msg.sender)
        end if
    else
        sendMsg(Change, msg.sender)
        Cache(msg.sender.info)
    end if
end switch
```

When a node receives an Update message, it means that there is an opportunity for the receiver node to join a group organized by the sender as the aggregate node. So, if the receiver's state is unknown, it sets its module-role as LN and

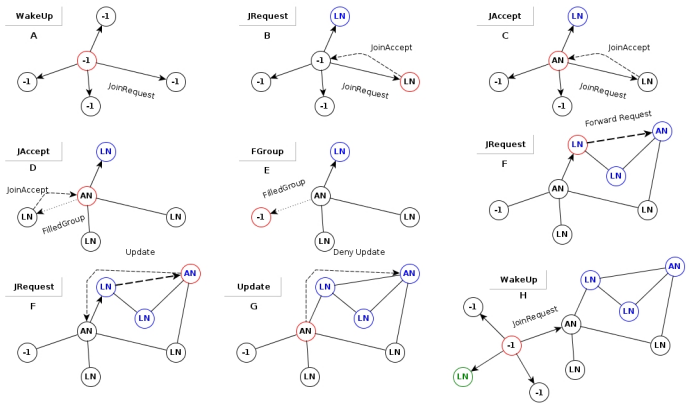

Fig. 2. Example of message processing for overlay making (maximum group size=3)

updates its grouping variable (qms-id) to the original sender's address. Otherwise, it sends a message of type DenyUpdate to the sender, notifying the remote AN that the current node is not available and it belongs to a different group. The DenyUpdate notification will erase the sender node from the list of AN-Group members of the destination node (remote AN). Furthermore, the update message receiver caches the information of the sender (remote AN) when its current state is AN. This information will be used during group optimization and super-node election process.

Step 3: The second step will be completed by triggering all the assigned WakeUp events in different individual nodes within various random intervals. Therefore, upon completing the second step, we expect that all the nodes in the system contribute to make a group and find out about their role within the group. But it might still be possible that some nodes are not within the group. In such a situation these nodes will establish their own group as an aggregate-node. They are clustered within a group that could be undersized. Besides, even the groups which are created during the second step may suffer from this problem. For example, the size of a group might be too short compared to the minimum allowed size of a group mentioned as the overlay design parameter. The third step of the overlay creation process tries to solve the aforementioned problem by implementing a grouping optimization algorithm through merging the small groups in order to create larger groups which could satisfy the required conditions of the overlay. The group size and the locality-factor are two important parameters for efficient clustering. The locality-factor that is proportional to the group's diameter is evaluated and determined during the second step while a new member is added to the group. So the grouping mechanism in the second step guaranties that the proximity of the nodes within a group will be preserved. On this step, the aggregate-nodes as the representatives of other group members inspect their own grouping conditions and if they notify that their groupings are not efficient particularly in terms of the group size they start to propagate Grouping-Info messages to a set of nodes which would be equal to the union of their members set, their proximity set (neighbors set) and their local set of possible aggregate nodes (cached information). We must note that the neighboring-indicator is an important parameter which specifies the neighbor's order. For example,

if the value of the neighboring-indicator parameter is 1, the proximity set (i.e., neighboring set) only includes the direct neighbors while for neighboring-indicator=2, the proximity set contains the combination of the direct neighbors and the second level neighbors.

Algorithm 3 Merging Optimization

```
if .state == AN and GValid(node, GVars, k) < 0 then
  iList= AMembers ∪ CacheANs ∪ Nset(N_i)
  for all remote − node IN iList do
    sendMsg(GInfo, remote − node)
  end for
end if
On-Group Checking:
if ANOps.size > 1 and ANOps.find(.AN) > 0 then
  finalSize = .group.size
  candidateANs.add(.AN)
  ANOps.erase(.AN)
  while finalSize <= Max and ANOps.size! = 0 do
    largestGroup = findLargestGp(ANOps)
    if finalSize + largestGroup.size <= Max then
      finalSize = finalSize + largestGroup.size
      candidateANs.add(largestGroup.AN)
    end if
    ANOps.erase(largestGroup.AN)
  end while
  if candidateANs.size > 1 then
    largestCGp = findLargestGp(candidateANs)
    candidateANs.erase(largestCGp.AN)
    for all remoteAN IN candidateANs do
      sendMsg(SMerging(largestCGp), remoteAN)
    end for
  end if
end if
```

Furthermore each aggregate node generates and assigns a Group-Checking event which will be triggered to recognize and determine the best possible group merging options. Grouping-Info messages contain information about the groups such as group size and qms-id (the aggregate-node address). The Grouping-Info receivers will collect these information for the later processing during Group-Checking event. When the Group-Checking event is triggered, the nodes which have received the Grouping-Info from at least two different aggregate-nodes will be distinguished as spot nodes. According to the collected information about the groups in vicinity, the spot nodes detect and analyze all the group-merging possibilities. It must also be taken into account that the final group after merging can't be over-sized and it should be smaller than the maximum allowed value for the overlay design while preserving the locality. On the other hand the merging cost (particularly in terms of communication) to switch the nodes from a small group to a larger group is much lower than switching from a larger group to a small group. Therefore the algorithm first chooses the largest validated group among the list and afterwards it selects a set of smaller groups (merging-list) to merge within the largest one. This operation will be iterated for the remaining groups in the list. Finally the spot nodes offer their merging proposals to the related aggregate-nodes through sending StartMerging messages to the aggregate-nodes of the groups within merging list. StartMerging provides a suggestion to the receiver to merge with a specific larger validated group while the locality preservation and the group making feasibility are guaranteed (See Algorithm3). Consequently the StartMerging receivers send their Merging-Request to the proposed destinations and later upon acceptance of the merging requests by target groups the requesters will notify their members to upgrade their grouping information according to the new aggregate-node.

Step 4: Similar to the second step, for each aggregate-node a WakeUP event is assigned which is automatically triggered in random intervals bounded within a specific time frame. Upon occurrence of the WakeUP event each aggregate-node sends a message of type SuperNodeElection and also its cached list of possible aggregate nodes to its neighbors. The neighbor nodes which are the LN members of this aggregate-node will also relay this request to their next level neighbor while considering to avoid forwarding replications. When the receivers of the SuperNodeElection are the LN members belonging to the groups which are differentiated from the requester group, the election requests will be forwarded to the corresponded aggregate-nodes in those groups.

IV. Evaluation

This section contains the evaluation results of the algorithm by running different experiments in order to evaluate the algorithm's performance with respect to load balance, efficiency and scalability. We use Omnet++ and OverSim simulator to simulate and evaluate our proposed algorithm. The most important contribution of this work is to provide a load balanced clustering approach which efficiently works in a purely unstructured distributed environment to implement service based distributed system such as resource discovery protocols.

TABLE II. Input parameter values used in the experiments (Scenario 1)

Parameter	Values
Physical network size	12 to 1000 nodes
Maximum size of a group in the first layer	40
Minimum size of a group in the first layer	10
Maximum size of a group in the second layer	20
Minimum size of a group in the second layer	2
Locality Factor k	k=.025
Number of iterations	100
Neighboring indicator N_i	1

The hierarchical overlay could be highly applicable to the deployment of such services if it can dynamically be adapted to the system requirements in a way that the overlay characteristics (such as the number of groups, replicas, super-nodes or aggregate nodes in each layer of the hierarchy) can be adjusted to suit the optimum number of values. For such overlay design, supporting the load balance is an important desirable aspect.

OverSim is a well-known validated P2P overlay network simulation framework which is based on Omnet++ simulator. However, in order to validate our simulation results, we have used different dynamic random topology graph for each iteration with discrete uniformly-distributed random edge weights (e.g. in term of latency [1,100]). We have also tested, validated and verified the simulation results for various small-sized systems such as n=12, n=24 and n=60.

TABLE III. INPUT PARAMETER VALUES USED IN THE EXPERIMENTS (SCENARIO 2)

Parameter	Values
Physical network size	12 to 1000 nodes
Maximum size of a group in the first layer	20
Minimum size of a group in the first layer	10
Maximum size of a group in the second layer	10
Minimum size of a group in the second layer	5
Locality Factor k	k=0.1,0.2,0.33,0.5
Number of iterations	100
Neighboring indicator N_i	1

To evaluate our work, in the first scenario, we focus on the load balance issue. According to the simulation parameters in the TableII we conduct a simulation scenario for the variable size of network. We run the simulation in multiple iterations to achieve results with better accuracy. The clustering is performed along the way to satisfy the required conditions. As the input parameters we bind the group size in each layer in range while we maintain the locality factor constant.

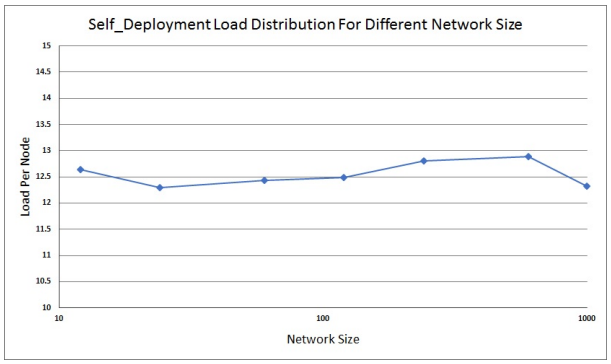

Fig. 3. Evaluation results for the Load distribution in different network sizes - Average number of processed messages per node for overlay making

The simulation results as illustrated in Figure4 demonstrates that the distribution of the communication loads for overlay setup are almost balanced between all the individual nodes in the system of size 600 and 1000 nodes and there are few nodes which receive and process the higher amount of messages. The clustering mechanism does not suffer from bottleneck and a single point of failure since the method does not rely on the specific pre-configured nodes. As we see in Figure3, the average number of transacted messages by each node during the clustering procedure is a value between 12 and 13 transactions for different network sizes. The processing nodes in the system can simultaneously and in parallel process their received messages. In other words, it means that the load distribution is independent from the network size, thus, the system provides the scalability while we increase the number of involved nodes.

In the second scenario (see TableIII), we evaluate the efficiency of the algorithm while considering the size of created groups and the locality of the nodes within each group. We have defined the locality as an indicator parameter that shows how much the nodes within a group are close to each other in general. It is assumed that all the nodes within the created logical groups through our proposed overly clustering mechanism are validated members since the locality factor for each new group's member

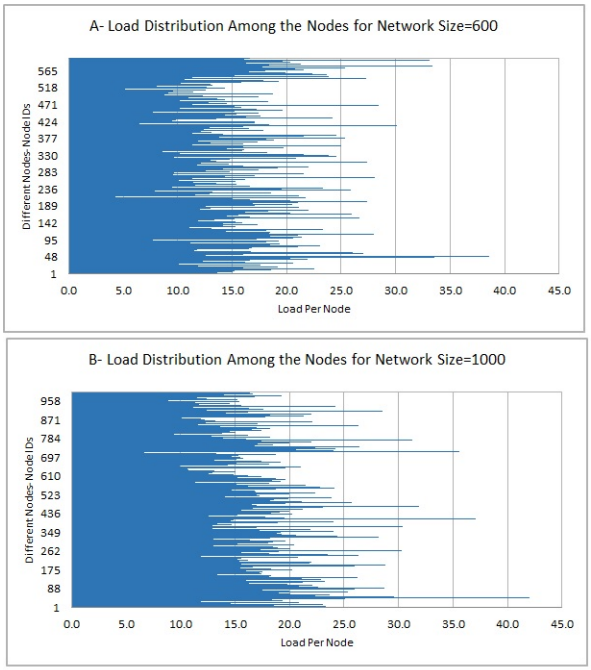

Fig. 4. Evaluation results for the Load distribution among the nodes for network size=600 nodes and network size=1000 nodes

in every layer will be checked upon arrival of the request in the corresponded aggregate-node or super-node of the target group. So, It is guaranteed that all the nodes within the created groups are locality-aware which are based on the required proximity conditions of the overlay. However the size of all the created groups might not satisfy the overlay design requirements. We define the Grouping Efficiency criteria according to the following formula:

$$\text{Grouping Efficiency} = \frac{\text{Number of the Qualified Groups}}{\text{Total Number of the Created Groups}}$$

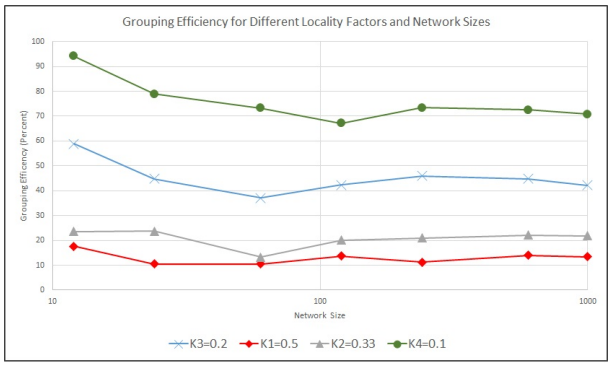

Fig. 5. Evaluation results for the clustering efficiency for different locality factors and network sizes

In the second scenario we keep constant the values for the maximum and minimum allowed size of each group while we change the value of the locality-factor as well as the network size. The results in Figure5 shows that the algorithm provides better efficiency for the lower values of the locality factor. If we increase the locality factor it will lead to the reduction of the

grouping efficiency, because, for the larger amount of locality factor, the group diameter would be smaller. This would reduce the freedom degree of the algorithm to create the groups within the validated range through adding or removing the potential nodes to the groups with regards to the underlying topology limitation, but the algorithm is still efficient for the proper values (moderate values) of the locality factors. Setting the locality factor for very small values would possibly result in inefficiency for lunching the services, lunched on the top of the overlay because of the expensive communication cost among the service components in the very large communicating groups. Figure5 also demonstrates the scalability of the system where the increment of the network size doesn't almost have a considerable impact on the grouping efficiency.

V. Conclusions

The recent huge increase in the popularity of large scale service-oriented distributed system has exposed the problem of service distribution in computing nodes/resources, which is especially due to the existence of the single point of failure in server-based systems and the scalability and efficiency challenges of the P2P systems. Logical overlays can improve the performance of service management and resource management through facilitating the efficient distribution and allocation of the service components in a large scale environment which is saturated by communicating nodes with thousands of processing resources. In this paper we present a hierarchical proximity-aware self-deployment and self-configuration algorithm to alter the underlying network topology in a way that the optimal clustering of the nodes in different layers of hierarchy can be performed dynamically. The simulation results show that our grouping mechanism supports the load balance while it composes the groups in the layers with respect to the required locality and clustering requirements. Furthermore, the experiment results prove the scalability and efficiency of the proposed algorithm for different system sizes.

VI. Acknowledgment

The authors acknowledge the support of Project FP7-ICT-2009.8.1, Grant Agreement No. 248465, Service-oriented Operating Systems (2010-2013) and of project Cloud Thinking(CENTRO-07-ST24-FEDER-002031).

References

[1] L. Gong, "Jxta: A network programming environment," IEEE Internet Computing, vol. 5, no. 3, pp. 88–95, May 2001. [Online]. Available: http://dx.doi.org/10.1109/4236.935182

[2] L. Gong et al., "Project jxta: A technology overview," Technical report, SUN Microsystems, April 2001. http://www. jxta. org/project/www/docs/TechOverview. pdf, Tech. Rep., 2001.

[3] S. Annapureddy, M. J. Freedman, and D. Mazières, "Shark: Scaling file servers via cooperative caching," in Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2, ser. NSDI'05. Berkeley, CA, USA: USENIX Association, 2005, pp. 129–142. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251203.1251213

[4] J. Considine, "Cluster-based optimizations for distributed hash tables," Boston University Computer Science Department, Tech. Rep., 2002.

[5] A. Montresor, "A robust protocol for building superpeer overlay topologies," in Peer-to-Peer Computing, 2004. Proceedings. Proceedings. Fourth International Conference on, 2004, pp. 202–209.

[6] B. Beverly Yang and H. Garcia-Molina, "Designing a super-peer network," in Data Engineering, 2003. Proceedings. 19th International Conference on, 2003, pp. 49–60.

[7] A. Crespo and H. Garcia-Molina, "Semantic overlay networks for p2p systems," in Proceedings of the Third international conference on Agents and Peer-to-Peer Computing, ser. AP2PC'04. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 1–13. [Online]. Available: http://dx.doi.org/10.1007/11574781_1

[8] G. Pirró, P. Trunfio, D. Talia, P. Missier, and C. Goble, "Ergot: A semantic-based system for service discovery in distributed infrastructures," in Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on, 2010, pp. 263–272.

[9] G. Pipan, "Use of the tripod overlay network for resource discovery," Future Generation Computer Systems, vol. 26, no. 8, pp. 1257–1270, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X1000018X

[10] H. Shen, Z. Li, T. Li, and Y. Zhu, "Pird: P2p-based intelligent resource discovery in internet-based distributed systems," in Proceedings of the 2008 The 28th International Conference on Distributed Computing Systems, ser. ICDCS '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 858–865. [Online]. Available: http://dx.doi.org/10.1109/ICDCS.2008.9

[11] T. Ghafarian, H. Deldari, B. Javadi, M. H. Yaghmaee, and R. Buyya, "Cycloidgrid: A proximity-aware p2p-based resource discovery architecture in volunteer computing systems," Future Gener. Comput. Syst., vol. 29, no. 6, pp. 1583–1595, Aug. 2013. [Online]. Available: http://dx.doi.org/10.1016/j.future.2012.08.010

[12] H. Ballani and P. Francis, "Towards a global ip anycast service," SIGCOMM Comput. Commun. Rev., vol. 35, no. 4, pp. 301–312, Aug. 2005. [Online]. Available: http://doi.acm.org/10.1145/1090191.1080127

[13] T. Stevens, M. De Leenheer, C. Develder, F. De Turck, B. Dhoedt, and P. Demeester, "Astas: Architecture for scalable and transparent anycast services," Communications and Networks, Journal of, vol. 9, no. 4, pp. 457–465, 2007.

[14] G. P. Jesi, A. Montresor, and O. Babaoglu, "Proximity-aware superpeer overlay topologies," in Proceedings of the Second IEEE International Conference on Self-Managed Networks, Systems, and Services, ser. SelfMan'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 43–57. [Online]. Available: http://dx.doi.org/10.1007/11767886_4

[15] Y. Sun, L. Sun, X. Huang, and Y. Lin, "Resource discovery in locality-aware group-based semantic overlay of peer-to-peer networks," in Proceedings of the 1st International Conference on Scalable Information Systems, ser. InfoScale '06. New York, NY, USA: ACM, 2006. [Online]. Available: http://doi.acm.org/10.1145/1146847.1146887

[16] C. Xiao and L. Nianzu, "Overlay construction within diffserv domains for qos-aware multicasting," in Proceedings of the 2010 Third International Symposium on Information Science and Engineering, ser. ISISE '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 271–274. [Online]. Available: http://dx.doi.org/10.1109/ISISE.2010.61

[17] X.-M. Huang, C.-Y. Chang, and M.-S. Chen, "Peercluster: A cluster-based peer-to-peer system," IEEE Trans. Parallel Distrib. Syst., vol. 17, no. 10, pp. 1110–1123, Oct. 2006. [Online]. Available: http://dx.doi.org/10.1109/TPDS.2006.142

[18] T. Obafemi-Ajayi, S. Kapoor, and O. Frieder, "Cluster-k+: Network topology for searching replicated data in p2p systems," Inf. Process. Manage., vol. 48, no. 5, pp. 841–854, Sep. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.ipm.2010.12.005

[19] T. Glatard, J. Montagnat, D. Emsellem, and D. Lingrand, "A service-oriented architecture enabling dynamic service grouping for optimizing distributed workflow execution," Future Gener. Comput. Syst., vol. 24, no. 7, pp. 720–730, Jul. 2008. [Online]. Available: http://dx.doi.org/10.1016/j.future.2008.02.011

[20] H. Gomaa and K. Hashimoto, "Dynamic self-adaptation for distributed service-oriented transactions," in Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2012 ICSE Workshop on, June 2012, pp. 11–20.

[21] G. Gharbi, M. B. Alaya, C. Diop, and E. Exposito, "Aoda: an autonomic and ontology-driven architecture for service-oriented and event-driven systems," International Journal of Collaborative Enterprise, vol. 3, no. 2, pp. 167–188, 2013.