2011 Fourth IEEE International Conference on Utility and Cloud Computing

The Cloud@Home Resource Management System

Salvatore Distefano, Maria Fazio and Antonio Puliafito
Department of Mathematics, University of Messina
98166, Messina, Italy
Email: sdistefano,mfazio,apuliafito@unime.it

***Abstract*—Cloud is strongly emerging in the ICT world as the distributed computing paradigm able to raise the level of abstraction of physical resources toward a "user-centric" perspective. In this challenging scenario, we propose a new form of computing mixing Cloud and volunteer paradigms' aims and goals into Cloud@Home. In particular, this work focuses on the Cloud@Home resource management issues. In order to deal with the dynamics of the overall infrastructure, where resources asynchronously join and leave the system, we propose a solution based on autonomic and self-adapting algorithms, structuring all the available resources into a hierarchical cluster. In this paper, we first specify the architecture of the Cloud@Home resource management system and then we detail the algorithms and the mechanisms that allow to elastically manage the hierarchical cluster-based infrastructure.**

***Keywords*-Cloud computing; Volunteer computing; resource management; hierarchical clustering; adaptation and reconfiguration.**

I. Introduction

Cloud is actually an effective solution in commercial and business context, providing a wide range of flexible, customizable, dynamic, resilient and cost effective infrastructural solutions. However, many open problems in Cloud infrastructures inhibit their use, such as information security (confidentiality and integrity), trustiness, interoperability, reliability and availability. To address some of them, we propose an alternative approach based on a more "democratic" form of Cloud computing, in which the computing resources of a single user, company, and/or community accessing the Cloud can be shared with the others contributing to the elaboration of complex problems.

In order to implement such idea, a source of inspiration could be the *Volunteer computing* paradigm. Volunteer computing uses computers volunteered by their owners to provide distributed scientific computing [1], [2]. Starting from the actual technological trend that moves towards Cloud and on demand service provisioning, we can figure out a volunteer Cloud infrastructure built on resources voluntarily shared (for free or by charge) by their owners or administrators, following a volunteer computing approach, and provided to users through a Cloud interface, *as a service*. We named such paradigm *Cloud@Home*.

One of the most important task to perform in Cloud@Home is the resource management. The complexity of such task is amplified by the high dynamics of the Cloud@Home environment, where resources can join and leave the infrastructure asynchronously, at their owners/administrators needs and wills. A feasible solution should take into account the infrastructure dynamics as well as the distributed nature of the paradigm. Since the dynamics of the system, regulated by the unpredictable and unexpected node joins and leaves, is not known a-priori, we propose a reactive solution that has to detect and to reconfigure the system after a specific join/leave event. It is based on autonomic self-adapting techniques that allow to locally aggregate and manage the Cloud@Home nodes and resources. Specifically we follow a hierarchical clustering approach, since it allows to clearly identify the cluster domains and to separate the concerns at different levels.

The aim of this paper is to present the design and implementation of a Cloud middleware specifically focused on the Cloud@Home resource management. On the one hand, it aggregates heterogeneous resources following an organization model based on hierarchical clustering. On the other hand, it offers the aggregated resources in a service-oriented perspective, as typical of the Utility-Cloud computing approaches. The proposed solution is based on the *Cloud@Home Algorithm for Resource Management* (CHARM), which includes and specifies the mechanisms for autonomously aggregating and managing the Cloud@Home contributor nodes into a hierarchical cluster. In this paper we specifically deal with the architecture of the Cloud@Home resource management system, identifying the main blocks and modules that have to implement such system. We also detail the mechanisms and the algorithms for autonomously managing the contributor nodes in order to achieve a self-adapting cluster hierarchy configuration.

To this purpose, the remainder of the paper is organized as follows: section II provides an overview of the Cloud@Home logical organization. Then, we focus on the Cloud@Home resource management system in section III, identifying the main resource management functionalities and necessary modules. Therefore, in section IV, we point out the specification of the static organization of resources in the hierarchical cluster, outlining tasks at different layers. We investigate the dynamic issues of the Cloud@Home resource management system in section V, where we present the specific algorithms for automatically creating and managing the

978-0-7695-4592-9/11 $26.00 © 2011 IEEE
DOI 10.1109/UCC.2011.26

hierarchical cluster. Final remarks of the proposed solution is discussed in section VI.

II. Cloud@Home Overview

The core idea behind Cloud@Home is to reuse *"domestic"* computing resources to build voluntary contributors' Clouds that can interoperate each other and, moreover, with other, also commercial, Cloud infrastructures. The overall infrastructure must deal with the high dynamics of its nodes/resources, allowing to move and reallocate data, tasks and jobs. It is therefore necessary to implement a lightweight middleware, specifically designed to deal with frequent, unpredictable and unexpected changes due to node join and leave. The solution should be therefore adaptive to such events through an elastic infrastructure that quickly and easily react to them, transparently to the final users.

Another important goal of Cloud@Home is the *security*, an open issue also in current Cloud infrastructures [3]. The virtualization in Clouds implements the isolation of the services, but does not provide any protection from local access. With regards security, the specific goal of Cloud@Home is to extend the security mechanisms of Clouds to the protection of data from local access.

Last but not least, *interoperability* or *intercloud* [4] is one of the most important goal of Cloud@Home. Cloud interoperability will enable Cloud infrastructures to evolve into a worldwide, transparent platform in which applications are not restricted to enterprise Clouds and Cloud service providers. Good starting points are the "open Cloud manifesto" [5], the Cloud security alliance drafts [3], the open Cloud computing interface (OCCI) [6], and the open virtualization format (OVF) [7] to address problems of compatibility among different infrastructures, security, trustiness and virtual machines (VM) monitor issues, respectively.

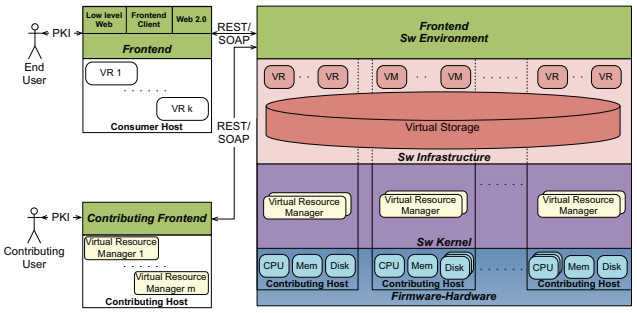

Figure 1. Cloud@Home Concepts Abstraction.

In order to identify mechanisms and tools able to satisfy the requirements above specified, in [8] we defined an abstraction of concepts, problems and issues to address in Cloud@Home according to the ontology provided in [9], which is synthesized in Fig. 1. Two types of users are there distinguished according to the role they assume in the Cloud: *end users*, if they only interface the Cloud for submitting requests, and/or *contributing users* if they make available their resources and services for building up and supporting the Cloud.

Specifically, the Cloud@Home *software environment* layer implements the user-infrastructure frontend interface. It is in charge of the resources and services management (enrolling, discovery, allocation, coordination, monitoring, scheduling, etc.) from the global Cloud system's perspective.

The Cloud@Home software environment is split into two parts, as shown in Fig. 1: the server side, implementing the resources management and related problems, and the client side, only providing mechanisms and tools for authenticating, enrolling, accessing and interacting with the Cloud services and resources.

The virtualization of physical resources offers to end users a homogeneous view of Cloud's services and resources. The Cloud@Home *software infrastructure* groups mechanisms and tools for virtualizing physical resources into *virtual resources* (VR), in order to implement homogeneity. The model of Fig. 1 is conceptual thus providing an abstraction on also the types of physical resources that can be considered. Thus, generally, any physical resource is represented by such a model.

The Cloud@Home *software kernel* provides to the software infrastructure, mechanisms and tools for locally managing the Cloud physical resources in order to implement the Cloud service. It usually includes the operating system, protocols, packages, libraries, compilers, programming and development environments, etc. Moreover, in order to adequately manage the resources provided by a specific host/device, it is necessary that one or more *virtual resource manager*s are installed into the contributing host for virtualizing the physical resources.

The Cloud@Home *firmware/hardware* is composed of a *"cloud"* of generic contributing nodes and/or devices geographically distributed across the Internet. They provide to the upper layers the physical-hardware resources for implementing the execution and storage services.

III. The Resource Management System

In this section we specifically focus on the resource management problem in Cloud@Home, trying to specifying an architecture by which identify the main tasks and assign them to specific components, blocks or modules. More specifically, the goals of the Cloud@Home resources management system are:

- Scalability: the increasing in the number of volunteer resources implies adding branches in the hierarchy, therefore the performance of schedulers are not significantly influenced by changes in the amount of available resources.
- Adaptability: the algorithm is able to detect changes in the logic organization of nodes and to react to these changes in real-time.

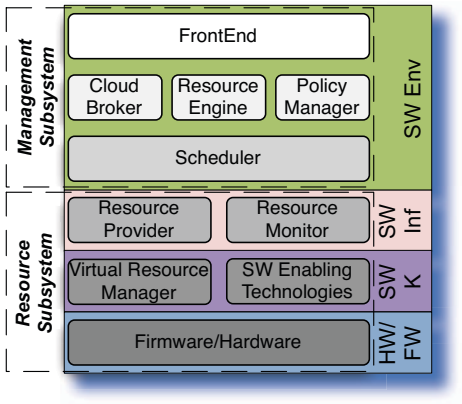

Figure 2. The Cloud@home resource management system.

- Autonomy: the algorithm is provided with autonomic technologies, which make it able to auto-configure nodes and auto-manage VMs and resources.
- Elasticity: the algorithm specifies reconfiguration policies in order to optimize the Cloud@Home infrastructure after contributors' join and leave.
- Fault tolerance: in order to deal with the degradation of performance and availability of the whole infrastructure due to unpredictable join and leave of contributors, redundancy techniques and job status tracking and monitoring are used.

The blocks implementing the Cloud@Home functional abstraction discussed in the previous section are pictorially depicted in the layered model of Fig. 2. It reports the core structure of the Cloud@Home resource management system. The Cloud@Home core structure is subdivided into two subsystems: *management* and *resource subsystems*. Such subsystems are strictly interconnected: the management subsystem implements the upper layer of the functional abstraction, while the resource subsystem implements the lower level functionalities.

A. Management Subsystem

In order to enrol and manage the distributed resources and services of a Cloud, providing a unique point of access for them, specific services and tools are required, grouped into the management subsystem. It is composed of the following parts: the *frontend*, the *Cloud broker*, the *resource engine*, the *policy manager* and one or more schedulers.

The frontend provides tools for Cloud@Home-user interactions, usually through a Web portal. It collects and manages the users' requests issued by the Cloud@Home clients. All such requests are forwarded to the blocks composing the underlying layer (resource engine, Cloud broker and policy manager) for processing.

The Cloud broker collects and manages information about the available Clouds and the services they provide (both *functional* and *non-functional* parameters, such as QoS, costs and reliability, *request formats' specifications* for Cloud@Home-foreign Clouds translations, etc.). Moreover, the Cloud broker collaborates with the resource engine to fulfil resource discovery, becoming *"inter-clouds"*.

The policy manager provides and implements the Cloud's access facilities. This task falls into the security scope of identification, authentication, authorization, and permissions management. To achieve this target, the policy manager uses an infrastructure based on PKI, smartcard devices, Certification Authority and SSO. The policy manager also manages the information about users' QoS policies and requirements.

The resource engine is the hearth of Cloud@Home. It is responsible for the resources' management, the equivalent of a Grid *resource broker* in a broader Grid environment. To achieve such goal, the resource engine applies a hierarchical policy. It operates at higher level, indexing all the resources of the Cloud. Incoming requests are delegated to *schedulers* that, in a distributed fashion, manage the requests incoming to the Cloud@Home, by assigning them to the resources, coordinated by the resource engine, as better explained in the following section.

The scheduler is a peripheral resource broker of the Cloud@Home infrastructure, to which the resource engine delegates the management of Cloud resources and services. It establishes which, what, where and when allocate a resource, moreover it is in charge of moving and managing services (for example VM migrations). From the end user point of view a virtual resource is allocated somewhere on the Cloud, therefore its migration is transparent for the end user that is not aware of any VM migration mechanism.

Since a scheduler can become a bottleneck if the system grows, to avoid the congestion further decentralized and distributed scheduling algorithms can be implemented. Possible strategies and tricks for facing the problem are:

- implementing a hierarchy of schedulers with geographic characterization: local, zone (as termed by Amazon), meta-zones (Eucalyptus), area, region, etc.
- replicating schedulers, which can communicate each other for synchronization;
- autonomic scheduling.

B. Resource Subsystem

The resource subsystem contains all the blocks implementing the local and distributed management functionalities of Cloud@Home. It is important to remark that the architecture shown in Fig. 2 is general, any kind of resource or any type of service can be implemented adequately characterizing the blocks to the resource hardware or to the specific type of service.

Thus, considering a generic Cloud service built on top of a specific type of resource (i.e., computing, storage, sensors, etc.), the resource subsystem should provide tools for managing virtual resources according to users' requests and requirements coming from the management subsystem.

It is composed of three main blocks: the *resource provider*, the *resource monitor* and *virtual resource manager*, locally managing the virtual resources on the physical site hosting them.

The resource provider implements functions for allocating, managing and destroying a virtual resource on the corresponding host. The resource monitor allows to take under control the local computing resources, according to requirements and constraints negotiated in the setup phase with the contributing user. Together with the resource monitor, the resource provider establishes if and when a virtual resource allocation request can be satisfied or it should be rejected. Moreover, if the local resources crash or become insufficient to keep the virtual resource service, the local resource monitor asks the upper level scheduler to suspend or to migrate the virtual resource elsewhere.

The virtual resource manager is the software providing the primitives, the API for managing a virtual resource. In case of virtual machines, the virtual resource manager is the hypervisor; in case of storage resources it is a distributed file system module, and so on. As software enabling technologies, we instead identify a class of "basic" software such as the operating system, protocols, libraries, compilers, packages, programming environments, etc. Finally all such system bases on the firmware/hardware provided by the physical node, characterizing the lowest layer of the resource management system.

IV. Hierarchical Clustering of Cloud@Home Resources

Once the resource management system architecture has been specified, identifying the main functions to provide and the components that implement them, it is necessary to explain how such components have to interact. In this section we focus on the description of the cluster-based hierarchy of elements, on which the Cloud@Home resource management system works.

Our aim is to decentralize the management task, assigning to the single node the local management of its resources autonomously coordinated with the other nodes, in an autonomic self-adapting fashion. We choose to organize the overall infrastructure as a hierarchical cluster of intercommunicating nodes with different tasks according to their positions in the hierarchy. Each node can directly communicate with "colleague" nodes belonging to the same level, while communications toward the other clusters/nodes are possible by going up in the hierarchy. In this way we obtain a connected graph, as shown in Fig. 3, that is not a tree since nodes are connected with all the nodes belonging to the same cluster at the same level (horizontal connections), and moreover they can have children, thus identifying new clusters. A cluster has only one scheduler that is the cluster-head and, in analogy with the network theory, it is also the gateway to the upper level, i.e., it can connect any node immediately enclosed in its cluster scope to a node in a cluster which cluster-head/scheduler is directly connected to it.

In such graph each logic functionality of Fig. 2 is represented by a specific node. In practice, two or more functionalities, such as the Cloud broker, the resource engine and the policy manager, can be grouped in the same physical site. Moreover, in order to achieve reliability and availability goals, it is possible to adequately replicate specific functionalities in different sites.

In the graph, it is possible to categorize nodes into two main classes: *schedulers* and *worker nodes*. Schedulers, as discussed above, manage the requests incoming from Cloud@Home end-users, at different level. The root of the hierarchy, the main scheduler, is the resource engine, characterizing the whole infrastructure domain, identified as cluster 0. Although, generally, a node could be at the same time both scheduler and worker node, in this work, for the sake of simplicity but without loss of generality, we assume a node can exclusively be a scheduler or a worker node, not both.

Lowest level schedulers monitor all the worker nodes in the corresponding cluster, detect if they join/leave the Cloud@Home infrastructure, dispatch the virtual resources of the cluster, assigning them to specific requests, and check if the corresponding jobs terminate or abort. Higher level schedulers manage clusters of schedulers instead of worker nodes, checking their availability both in terms of internal resources (CPU, memory, disk, ...) and related to the overall underlying requests of the corresponding subgraph. In fact, each scheduler keeps two types of information for each node in its cluster: the node ID and the load. The node ID univocally identifies the specific node while the load is a parameter weighting the node condition in terms of CPU, memory, disk, or specific parameters related to the specific type of resource considered. Such parameter is used to implement the scheduling policies as described in the following section.

The hierarchical clustering guarantees the isolation of the activities of nodes belonging to the same cluster from those of different clusters. In fact, a scheduler, as also the resource engine, operates within its own cluster scope, assigning the incoming requests to the schedulers/worker nodes enclosed in the cluster in the level immediately below. It also directly communicates with all the schedulers in the same level belonging to the immediately above cluster both enclosing them and the considered scheduler. Therefore direct intra-cluster communications are possible for schedulers and worker nodes. This implies that the nodes have a local knowledge of the hierarchy and, in particular, they know just the elements of the cluster they belong to. For example, the scheduler of the cluster $CL1.1....1$ of Fig. 3 knows its upper-level scheduler $CL1.1...$, the worker node in its own cluster $CL1.1....1$, and the colleague schedulers $CL1.1....2$-

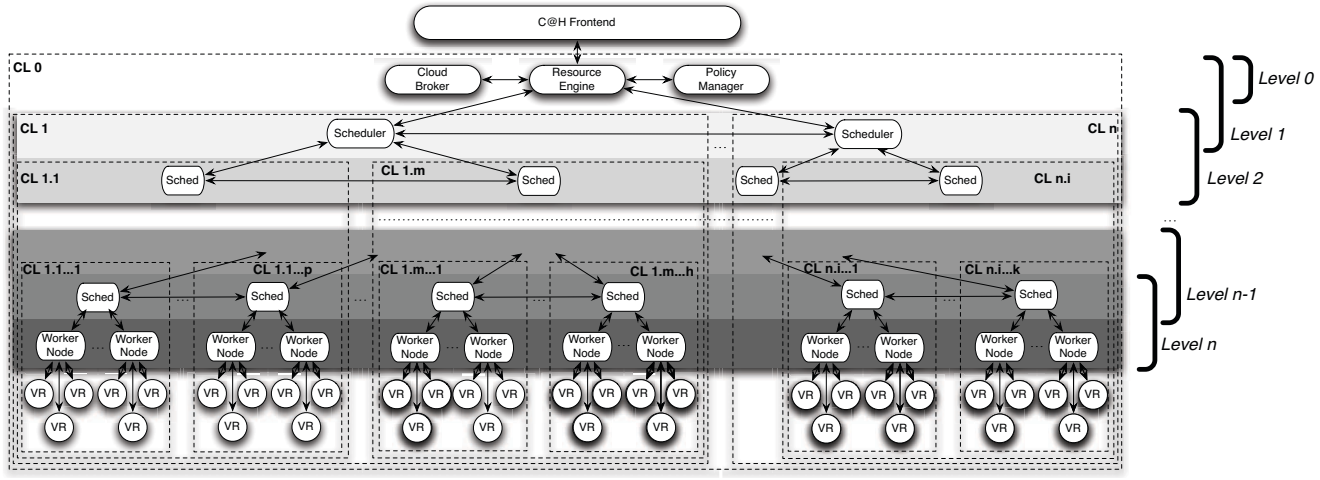

Figure 3. Hierarchical cluster resource management in Cloud@Home

$CL1.1....p$. It does not know how many levels, nodes and schedulers are in the hierarchy. It is important to remark that, inter-cluster communications are very unlikely-rare events, since a generic node in the hierarchy only know its adjacent nodes and that belonging to the same cluster. As introduced above, in this way inter-cluster communications are also possible, but indirectly by going up in the hierarchy. In fact, as detailed in the following, inter-cluster communications are only used to restore from common cause/cascading faults.

Worker nodes create, destroy and manage virtual resources according to the incoming request processing and satisfaction. Assuming a *job* is a request processing on a specific worker node, the job management, execution and monitoring is in charge of the corresponding worker node. It has to check the status of each job and to inform the system as soon as the job terminates.

As already introduced, the non-leaf nodes work as schedulers. Assuming, for the sake of simplicity, that all the virtual resources provided by the Cloud@Home are of the same type (e.g., virtual machines), users that access the Cloud@Home architecture ask for a specific number of virtual resources, but they are unaware on the physical location of such virtual resources. Depending on the scheduling policies and on the infrastructure load condition, a virtual resources' request can be satisfied by allocating virtual resource jobs on different worker nodes. However, the allocation of specific virtual resources for a request is transparent to the user and is automatically performed by the Cloud@Home schedulers.

V. Cloud@Home Algorithm For Resource Management

The nodes in a Cloud@Home infrastructure can change their availability during the time, since some sites can turn off or disconnect the network, whereas new nodes join the system. Therefore, the Cloud@Home infrastructure has to automatically organize and reconfigure nodes in worker nodes and schedulers in order to provide an efficient hierarchical infrastructure for the resource management. We named the automatic process able to reorganize nodes in the system as *Cloud@Home Algorithm for Resources Management* (CHARM). Whenever a new volunteer contributor joins the network, the Cloud@Home infrastructure grows. This consequently corresponds to an increase of the number of the infrastructure worker nodes and, hence of the number of available virtual resources. However, the performance of a scheduler are strongly related to the number of worker nodes it manages since when increases the scheduler could be overloaded and, as a consequence, its performance degrades till compromising its functioning. Also the amount of messages necessary to coordinate nodes in the cluster may collapse communications inside the same cluster. It is therefore necessary to adequately deal with such issues through specific strategies that act on the hierarchy establishing a trade-off between the number of worker nodes and that of schedulers in the Cloud@Home system. Similar considerations can be done whenever resources leave the network. To avoid wasting of resources in under-loaded nodes, a reconfiguration strategy of the nodes in the hierarchy has to be introduced, for example by merging two or more clusters. Moreover, one of the main goal of CHARM is to implement a distributed solution, and therefore, with specific regards to the reconfiguration strategies, it specifies autonomous mechanisms and tools able to deal with such problems.

In Cloud@Home, each user that accesses the system can provide its own hardware resources to contribute to the Cloud@Home services. To aggregate its resources with all the resources of the system, it has to become a node of the hierarchical cluster. To this purpose, at first, it contacts the resource engine, that implements the interface to the Cloud@Home infrastructure. Then, it exchanges messages

with schedulers across the hierarchy in order to identify the cluster to which it can be associated and the role it has to assume in a such hierarchy. If there are no nodes in the Cloud@Home infrastructure the new node becomes scheduler and waits for new nodes to include in its cluster.

Once the first cluster is composed/created, the infrastructure autonomously evolves by creating the cluster hierarchy. Such evolution follows the contribution resource dynamics due to joins and leaves, to which the Cloud@Home management system has to react adapting the infrastructure cluster hierarchy accordingly. More specifically, the system has to implement an elastic mechanism able to expand the hierarchy when the number of contribution resources increases, while it has to collapse the hierarchy in correspondence to significant reductions of contribution resources. The functions regulating such process are the *split* and the *merge* ones, respectively. Discursively, the split allows to grow the cluster hierarchy if a scheduler is overloaded, creating a new scheduler if it is not possible to migrate sub-graphs from the overloaded schedulers to others. On the other hand, the merge implements the split complementary operation of reducing the cluster hierarchy if a scheduler is underloaded or underexploited.

Two types of split an merge operations are specified, as better detailed in the following: *horizontal*, if the split/merge exclusively involves nodes belonging to the same level of the cluster hierarchy, or *vertical* when the split/merge is performed by adding/removing a level in the hierarchy. The latter is in particular used to decongest the resource engine, since it is not possible to split the resource engine into two or more nodes. To avoid single point of failure, standby redundancy is instead used, by which a resource engine is replicated and its spares are periodically updated by the actual state of the infrastructure. If the resource engine fails, its failure is promptly detected through a mechanism such as the heartbeat monitor and a hot standby resource engine spare is immediately activated and synchronized, so that the final user does not notice anything.

Split and merge policies are based on the $load$ parameter, that provides an indirect estimation of the cluster size starting from the residual capacity of a specific node. It comes from measurements of the node resource utilization and, specifically, it is obtained as follows:

$$load = \alpha * RAM + \beta * CPU + (1 - \alpha - \beta) * DSK \quad (1)$$

where RAM is the fraction of used memory over the total memory capacity, CPU is the percentage of CPU utilization and DSK is the fraction of used storage resources over the total storage capacity. α and β are two parameters in the range $[0, 1[$ such that $\alpha + \beta \leq 1$, weighting the measured parameters according to the specific type of service provided by the infrastructure. They are set in the resource engine and conveyed to all the nodes.

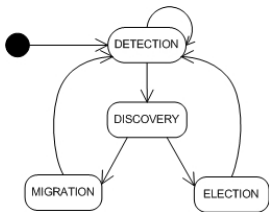

Figure 4. State diagram of the split management on a overloaded scheduler.

Periodically, each node evaluates its $load$ to verify its state. Two threshold values of such parameter are specified for characterizing the overloaded state ($T_{overloaded}$) and the underloaded state ($T_{unloaded}$) of a node. Thus, if the load of a node is higher than the $T_{overloaded}$ threshold, the split algorithm is triggered on the same node. On the other hand, if the load of a node is lower than the $T_{unloaded}$ threshold, the merge algorithm is triggered by the node.

The election algorithm used in CHARM to establish the scheduler/cluster head of a cluster is based on the *bully algorithm* [10], which is a well known procedure in distributed computing for dynamically selecting a coordinator by process ID numbers. In the following, we therefore detail the main algorithms to manage a Cloud@Home infrastructure hierarchical cluster, i.e., the split and the merge ones.

A. Split algorithm

Two clusters are generated from one cluster by splitting nodes in the cluster in two groups. Of course, the split algorithm invokes the bully election algorithm to identify schedulers as cluster-heads of the new clusters. The hierarchical cluster is expanded either horizontally by creating a new cluster at the same level of the overloaded one, or vertically by adding a new level, according to the status of the schedulers in the hierarchy. Fig. 4 shows the state diagram of the split process on a scheduler. It is composed by four states: DETECTION, DISCOVERY, MIGRATION, ELECTION.

As soon a node becomes scheduler, such algorithm enters the DETECTION state, in order to periodically check whether the node is overloaded or not. When it detects an overloading of its activities, it stops the scheduling task, enters the DISCOVERY state and looks for some scheduler able to accept its sub-graphs. If the discovery process succeeds, the scheduler enters the MIGRATION state. It asks to the available schedulers to accept one or more of its node sub-graphs and, in case of acknowledgements, the migration of these sub-graphs is performed, thus modifying the graph and consequently the cluster hierarchy. If the discovery process fails, no schedulers are available to accept its node sub-graphs, the overloaded scheduler enters the ELECTION state in order to start up the election process, which identifies a new scheduler among the nodes in its cluster, thus expanding the hierarchy. The new elected scheduler is able to accept nodes of the overloaded scheduler. As soon

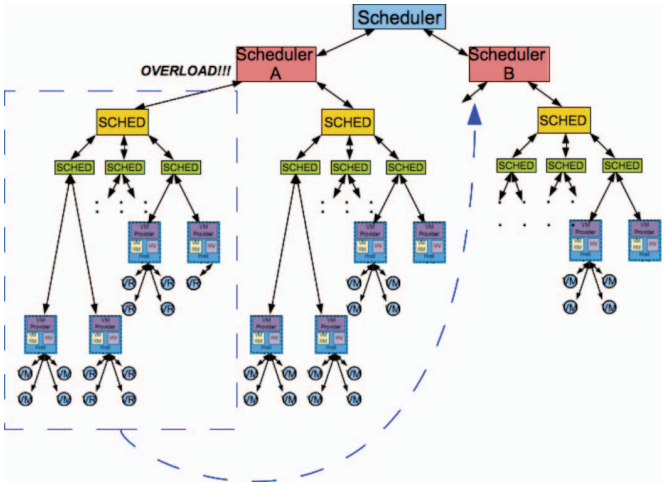

Figure 5. Split migration due to the overload of the scheduler A.

as the election process is activated, the scheduler switches to the MIGRATION state, and then, once the migration is performed, transitions to the DETECTION state if its load comes back below the load threshold after the migration.

The transition between the ELECTION and DETECTION states allows the scheduler to properly work during the election phase. In fact, the election process can be long in comparison to the activities running on the nodes. An alternative solution could be to wait the end of the election process to transit from the ELECTION to the MIGRATION state. However, keeping the scheduler in the ELECTION state during the election process means to block all the activities in the cluster. In order to avoid such busy waiting, we have designed the protocol to work in the DETECTION state during the election phase, thus limiting the impact on the performance of the overall infrastructure.

Fig. 5 shows how the load of clusters is reorganized as a consequence of the scheduler A overload detection and the following migration to scheduler B.

In the system, also the resource engine can be overloaded. However, as stated above, it is unique and therefore it is not possible to apply an horizontal split to this. Thus, whenever the resource engine is overloaded, a new level has to be added to the hierarchy, implementing a vertical growing of the hierarchical clustering. In this way, the resource engine can be in just two states: DETECTION state, in which it checks if it is overloaded or not, and in the ELECTION state, if it is overloaded. Before enter the ELECTION state, the resource engine stops all its services and then it waits for the election of a new scheduler in a new level immediately below the resource engine one, which becomes the cluster-head of the whole infrastructure. Probably, in the next future, such scheduler will quickly be in overload, thus triggering a horizontal split till the system will reach its balance. After the election of the scheduler, the resource engine restarts its activities and the Cloud services become again available.

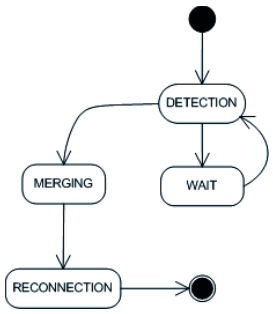

Figure 6. State diagram of the merging process.

Differently from schedulers in the ELECTION state, the resource engine does not transition in the DETECTION state when the election process is started, but when it is finished.

B. Merge algorithm

The splitting algorithm allows to increase the size of the hierarchical cluster balancing the load of resource engine and schedulers at different levels. However, the volunteer nature of contributors implies a very dynamic infrastructure where resources join and leave the system. When nodes leave the system, the load of some schedulers can decrease. However, keeping up useless schedulers is a cost for the infrastructure in terms of both complexity and available resources. To this purpose CHARM implements a merging algorithm which allows to modify the hierarchical cluster organization by reallocating redundant schedulers in the system. For example, a scheduler without worker nodes is useless for the architecture, but it can be very helpful if it changes its role becoming a worker node.

As for the split, the merge of schedulers in the Cloud@Home infrastructure can occur horizontally or vertically. The vertical merging allows to remove unnecessary levels in the hierarchy. A scheduler X detects that its level in the hierarchy is redundant if the following conditions are all satisfied:

$$load_X \leq T_{unloaded}\ AND \\ num\_w - nodes_X \neq 0\ AND \\ num\_Sched_{brother} == 0 \quad (2)$$

where $num\_w - nodes_X$ is the number of worker nodes in the cluster of X and $num\_Sched_{brother}$ is the number of "brother" schedulers at the same level of X. Thus, $num\_Sched_{brother} == 0$ means that there are not other schedulers at the same level of X, and consequently the system can automatically delete such level if a scheduler is unloaded and it is alone at that level. To describe the behavior of a scheduler during a vertical merging, we can refer to Fig. 6. Four states characterize the scheduler by the merge algorithm: DETECTION, WAIT, MERGING, RECONNECTION. The scheduler X in the DETECTION state periodically checks if the conditions of (2) are verified.

However, the merging process can not start if the scheduler is already involved in other management processes, such as election or resource discovery. In this case, the scheduler enters in the WAIT state in order to wait that the process in progress terminates. This approach allows to give higher priority to management functionalities instead of the resizing of the infrastructure. If conditions (2) are verified and the scheduler is not involved in any other process, it enters the MERGING state. It communicates to nodes at the immediately upper and lower levels that it is going to reconnect to the system. In this way, the nodes in the cluster will hook on their new scheduler. At the same time, scheduler X sends a message to inform its scheduler about the nodes of its cluster that it will inherit. In Fig. 7 we show the effect of vertical merging at level 1.

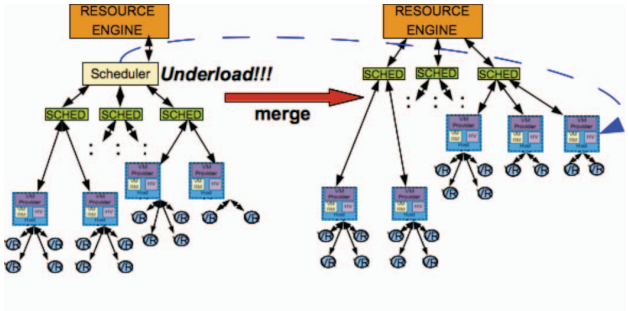

Figure 7. Vertical merging of levels 1-2.

After this step, scheduler X transits in the RECONNECTION state, where it restarts the procedure of allocation inside the cluster hierarchy, as a new resource entering the system.

The horizontal merging of the system occurs whenever a scheduler have no more nodes in its cluster, that is when $num\_w - nodes_X = 0$. The management of horizontal merging is very similar to the vertical one and scheduler X can be in one among the four state shown in Fig. 6. Fig. 8 shows the effect of the horizontal merging at level 2.

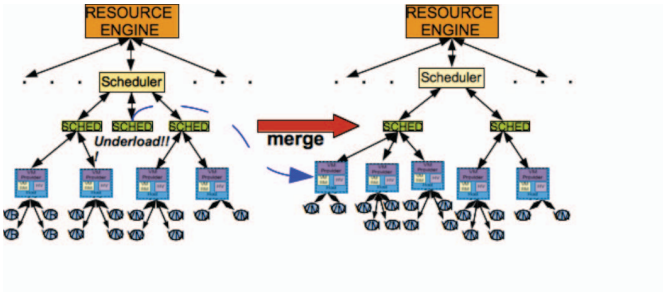

Figure 8. Horizontal merging at the level 2.

VI. Conclusions

This paper deals with the resource management issue in Cloud@Home, a Cloud built on top of volunteer resources that can asynchronously join and leave the system. The solution proposed aims at structuring the overall infrastructure into a hierarchical cluster, in order to achieve separation of concerns and of communication domains at the basis of a decentralized approach. In fact the Cloud@Home resource management system detailed in the paper adopts and specifies autonomic and self-adapting algorithms that allows to built up a Cloud@Home hierarchical-cluster infrastructure without any centralized management or control. According to such technique, a node autonomously identifies its position in the hierarchy by communicating with the other nodes, through a specific election mechanism. It also autonomously reacts to infrastructure changes due to nodes join and leave, through split and merge algorithms implementing the hierarchy elasticity. Ongoing and future work are mainly focused on testing it on real infrastructures in order to prove its scalability and effectiveness.

References

[1] D. P. Anderson and G. Fedak, "The computational and storage potential of volunteer computing," in *CCGRID '06*, pp. 73–80.

[2] G. Fedak, C. Germain, V. Neri, and F. Cappello, "Xtremweb: a generic global computing system," *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on*, pp. 582–587, 2001.

[3] The Cloud Security Alliance Founding Members, "The Cloud Security Alliance (CSA) ."

[4] R. Buyya, R. Ranjan, and R. Calheiros, "Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services," *Algorithms and Architectures for Parallel Processing*, pp. 13–31, 2010.

[5] The Open Cloud Manifesto Supporters, "The Open Cloud Manifesto," http://www.opencloudmanifesto.org/opencloudmanifesto1.htm, Tech. Rep., 2009.

[6] R. Nyren, A. Edmonds, A. Papaspyrou, and T. Metsch, *Open Cloud Computing Interface - Core*, OCCI working group - Open Grid Forum, http://ogf.org/documents/GFD.183.pdf, June 2011.

[7] X. I. VMWare Inc., "The open virtual machine format whitepaper for ovf specification," 2007, http://www.vmware.com/appliances/learn/ovf.html.

[8] S. Distefano, V. D. Cunsolo, and A. Puliafito, "A taxonomic specification of Cloud@Home," in *Proceedings of the 6th International Conference on Intelligent computing*, ser. ICIC'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 527–534.

[9] L. Youseff, M. Butrico, and D. Da Silva, "Toward a unified ontology of cloud computing," in *Grid Computing Environments Workshop, 2008. GCE '08*, Nov. 2008, pp. 1–10.

[10] H. Garcia-Molina, "Elections in a distributed computing system," *IEEE Trans. Comput.*, vol. 31, pp. 48–59, January 1982.