Proceedings of the Federated Conference on Computer Science and Information Systems pp. 749–758
DOI: 10.15439/2017F274
ISSN 2300-5963 ACSIS, Vol. 11

CloudLightning: a Self-Organized Self-Managed Heterogeneous Cloud

Huanhuan Xiong[1], Dapeng Dong[1], Christos Filelis-Papadopoulos[2], Gabriel G. Castañé[1], Theo Lynn[3], Dan C. Marinescu[4], John P. Morrison[1]

[1]Department of Computer Science, University College Cork, Cork, Ireland
{h.xiong, d.dong, g.gonzalezcastane, j.morrison}@cs.ucc.ie

[2]Department of Electrical and Computer Engineering, Democritus University of Thrace, Xanthi, Greece
cpapad@ee.duth.gr

[3]Dublin City University, Dublin, Ireland
theo.lynn@dcu.ie

[4]Department of Computer Science, University of Central Florida, Orlando, FL 32814, USA
dcm@cs.ucf.edu

***Abstract*—The increasing heterogeneity of cloud resources, and the increasing diversity of services being deployed in cloud environments are leading to significant increases in the complexities of cloud resource management. This paper presents an architecture to manage heterogeneous resources and to improve service delivery in cloud environments. A loosely-coupled, hierarchical, self-adapting management model, deployed across multiple layers, is used for heterogeneous resource management. Moreover, a service-specific coalition formation mechanism is employed to identify appropriate resources to support the process parallelism associated with high performance services. Finally, a proof-of-concept of the proposed hierarchical cloud architecture, as realized in CloudLightning project, is presented.**

***Index Terms*—Hierarchical architecture, heterogeneity, cloud computing, resource management, coalition formation**

I. Introduction

OVER the last decade, large scale cloud services have been created by service providers such as Amazon, Microsoft, Google and Rackspace. In order to meet the needs of their consumers, cloud service providers have built data centers at unprecedented scales. In 2013 Steve Ballmer estimated that Google, Microsoft and Amazon are running roughly one million servers each [1]. These data centers are large industrial facilities containing computing infrastructure: servers, storage arrays and networking equipment. This core equipment requires supporting infrastructure in the form of power, cooling and external networking links. Reliable service delivery depends on the holistic management of all of this infrastructure as a single integrated entity: the warehouse-scale computer (WSC) [2].

The WSC design suggests that a **hierarchical** top-down model is used to manage large-scale cloud infrastructures. Meanwhile, the growth in cloud raises the question of how far we can push the limits of computing and communication systems, while still being able to support effective policies for resource management and their implementation mechanisms. Software running on cloud environments is becoming increasingly complex, consisting of more and more layers. Thus, the challenge of controlling these large-scale systems is exacerbated. Control theory tells us that accurate state information, and a tight feedback loop, are critical elements for effective control of a system. In a traditional hierarchical organization, the quality of state information degrades as it is propagated from the bottom to the top; only local information about the state of a server is, by definition, accurate. Moreover, the value of this information is time sensitive, it must be acted upon promptly because the state changes rapidly. WSC-like architectures employ centralized resource management associated with the upper layers of the hierarchy. These models, based on monitoring, are costly, since the communication overhead can be more than two orders of magnitude higher than that required by decentralized resource management strategies as illustrated in [3].

The increasing **heterogeneity** of cloud servers, and the diversity of services demanded by the cloud user community, including access to High Performance Computing (HPC), are some of the reasons why it is imperative to devise new resource management strategies. These strategies should aim to significantly increase the average server utilization and the computational efficiency measured as the amount of computations per Watt of power, make cloud computing more appealing and lower the costs for the user community. Finally, they should simplify the mechanisms for cloud resource management.

Current cloud infrastructures are mostly homogeneous, centrally managed and made available to the end user through the three standard delivery models: Infrastructure as a Service

(IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). In the traditional PaaS and SaaS models, the user is completely unaware of the physical resources being used to run services. This is also the case for IaaS (if bare metal offerings are ignored), since the infrastructure offered is most often virtualized on top of commodity hardware. The drive towards connecting specialized hardware to the cloud (such as dedicated HPC machines and servers clustered on dedicated high-speed networks) and towards augmenting servers with specialized accelerators (such as Graphics Processing Units - GPUs, Many Integrated Cores - MICs, and Field Programmable Gate Arrays - FPGAs) has the potential to shift the operational dynamics of the cloud. By incorporating diverse hardware, the cloud becomes heterogeneous and an opportunity is created for offering discriminated services based on the operational characteristics of these different hardware types. Thus, the possibility exists for realizing the same service at different costs and at different performance levels. However, exploiting different architectures poses significant challenges. To efficiently access heterogeneous resources, to exploit these resources to reduce application development effort, to make optimizations easier and to simplify service deployment, requires a re-evaluation of our approach to service delivery.

In this paper, a hierarchical cloud architecture is proposed to address heterogeneous resource management and advanced service delivery. The remainder of the paper is organized as follows. Section II reviews the related work from the literature, while Section III introduces the hierarchical cloud architecture for heterogeneous resource management and service delivery. A self-organizing self-managing system is presented as a proof-of-concept of our proposed hierarchical cloud architecture in Section IV, and concluding remarks and future work are presented in Section V.

II. Related work

A. Hierarchical frameworks in cloud

Warehouse Scale Computers (WSCs) consist of thousands of commodity parts including processors, memory, disk, network, servers, etc., which are attached together to form a warehouse of interconnected machines [4]. The WSC is widely used as large-scale datacenter by Google, Yahoo, Amazon, Facebook, Microsoft and Apple [5]–[7].

In a WSC the hierarchical architecture is formed from a collection of physical machines and interconnects. A server is composed of a number of processor sockets, local shared and coherent DRAM, and a number of directly attached disks. The DRAM and disk resources within a rack are connected via a first-level rack switch, and all resources in all racks are connected through a cluster-level switch.

Within the physical hierarchy of a WSC, control theoretic feedback loop techniques are often used to enable system stability [8] and effective resource allocation [9], as well as to improve system performance [10] and power efficiency [11], [12]. The previous study [8] of using queuing theory with feedback control theory for performance guarantees in QoS-aware systems shows that the combined schemes perform significantly to achieve QoS specifications in highly unpredictable environments. Performance control of a web server by using classical feedback control theory was studied in [10], achieving overload protection, performance guarantees, and service differentiation in the presence of load unpredictability. Wang et al. [11] proposed a cluster-level control architecture that coordinates individual power and performance control loops for virtualized server clusters. The higher layer controller determines capacity allocation and VM migration within a cluster, while the lower layer controllers manage the power level of individual servers.

B. Heterogeneity in cloud

Limitations on power density, heat removal and related considerations require a different architecture strategy for improved processor performance by adding identical, general-purpose cores [13]. Unlike traditional cloud infrastructure built on an identical processor architecture, heterogeneity assumes a cloud that makes use of different specialist processors that can accelerate the completion of specific tasks or can be turned off when not required, thus maximizing both performance and energy efficiency [14]. Another previous study [15] proposes a resource allocation strategy in a heterogeneous cluster (integration of core nodes and accelerator nodes) to realize a scheduling scheme that achieves high performance and fairness.

Very recently, larger cloud infrastructure providers have been offering commercial heterogeneous cloud services, e.g. Amazon Web Services offers a variety of GPU and FPGA services [16]. Similarly, OpenStack also supports GPU and FPGA accelerators on provisioned VM instances [17]. As demand for better processor price and power performance increases, it is anticipated that larger infrastructure providers will need to cater for several of these processor types and specifically for the emerging HPC public cloud market [18].

Currently, many EU-funded projects are attempting to bring heterogeneous resources into cloud environments. The Hardware- and Network-Enhanced Software Systems for Cloud Computing (HARNESS) project [19] brings innovative and heterogeneous resources (such as FPGAs, GPUs) into cloud platforms by improving performance, security and cost-profiles of cloud-hosted applications. Heterogeneous Secure Multi-level Remote Acceleration Service for Low-Power Integrated System and Devices (RAPID) [20] proposes the development of an efficient heterogeneous CPU-GPU cloud computing infrastructure, which can be used to seamlessly offload CPU-based and GPU-based (using OpenCL API) tasks of applications running on low-power devices(such as smartphones, tablets, portable/wearable devices, etc.) to more powerful devices over a heterogeneous network (HetNet).

Managing different architectures independently and integrating with an existing general purpose cloud architecture can be very challenging. The adoption of heterogeneous resources dramatically increase the complexity of an already complex cloud ecosystem.

C. Resource management frameworks in cloud

Apache Mesos [21] is a platform for abstracting compute resources (e.g., CPU, memory, storage) away from machines (physical or virtual) and sharing commodity clusters between multiple diverse applications (e.g., Hadoop, Spark and MPI). Aiming to share clusters efficiently between different applications, Mesos introduces a two-level scheduling mechanism called resource offers. Over a series of resource allocation steps, the Mesos master decides how many resources to offer each application, while applications decide on which resources to accept and which computations to run on them.

Google Borg system [22] is a cluster manager running hundreds of thousands of jobs, from many thousands of different applications, across a number of clusters each with up to tens of thousands of machines. The Borg system consists of a logically centralized controller called the Borgmaster and an agent process called the Borglet that runs on each machine.

Mesos and Borg share the same fundamental approach of a centralized resource manager and multiple application frameworks are supported. Borg and Mesos work With bare-metal machines and, as such, are not directly concerned with virtualization. Furthermore, being monolithic schedulers, scalability is an issue.

Google Omega [23] introduces a new cluster manager scheduling architecture using shared state and lock-free optimistic concurrency control mechanisms.

Kubernetes [24] is an open-source platform for placing applications in Docker containers onto multiple host nodes, which runs both physical and virtual resources. The Kubernetes architecture is defined by a master server and multiple minions (nodes). The command line tools connect to the API endpoint in the master, which manage and orchestrate all the minions, and Docker hosts that receive the instructions from the master and run the containers.

Omega and Kubernetes have been developed to support multiple parallel schedulers and to place applications in Docker containers separately. However, there are still many outstanding issues with Omega and Kubernetes, such as massive message passing between the parallel schedulers, many housekeeping activities within each scheduler, without reference to server utilization.

D. Self-organization self-management approach

Self-organization is a powerful technique for addressing complexity and borrows heavily from the natural sciences and the study of natural systems [25] [26]. It has been applied successfully in complex engineering projects [27] [28]. In the computing context, Heylighen and Gershenson [29] define organizations as structure with function and self-organization as a functional structure that appears and maintains spontaneously. Alan Turing [30] once observed that global order arises from local interactions. In this context, global order is achieved through propagation and adaptation. Components in a self-organizing system are mutually dependent and typically only interact with nearby components. However, the system is dynamic and therefore the components can change state to meet mutually preferable, satisfactory or stable states [29]. As they meet these states, they adapt and achieve *fit* and this propagation of *fit* results in system growth. Structural complexity is driven by increasing the interchangeability and individuality of components capable of achieving *fit*. As more and more components adapt and become assimilated within the system, complexity increases to incorporate the individual characteristics of components. Growth only stops when resources have been exhausted and self-maintenance is the *de facto* purpose of the system. As such, self-organizing systems are defined by their robustness, flexibility, and adaptivity [31].

Self-management has been posited as a solution to complexity in IT infrastructure development generally and cloud computing specifically [32] [33]. It has its roots in autonomic computing. Such systems are designed to react to internal and external observations without human intervention to overcome the management complexity of computer systems [34]. As such, self-managing systems are described in terms of four aspects of self-management, namely, self-configuration, self-optimization, self-protection and self-healing [35].

The application of self-organization and self-management principles to cloud computing is at an early stage. Zhang et al. [34] posit that cloud computing systems are inherently self-organizing and while they exhibit autonomic features are not self-managing as they do not have reducing complexity as a goal. Marinescu et al. [36] argue that cloud computing represents a complex system and therefore self-organization is an appropriate technique to address this complexity. They propose an auction-driven self-organizing cloud delivery model based on the tenets of autonomy of individual components, self-awareness, and intelligent behavior of individual components. Extending work on self-manageable cloud services by Brandic [37] at an individual node level, Puviani and Frei [33] propose self-management as a solution for managing complexity in cloud computing at a system level. They propose using a catalog of adaptation patterns based on requirements, context and expected behavior. These patterns are further classified according to the service components and autonomic managers.

III. Hierarchical cloud architecture

An overview of a hierarchical cloud architecture for heterogeneous resource management and service delivery in shown in (see Fig.1). The resource management framework is composed of a logical hierarchy and is described in Section III-A. At the bottom level of this hierarchy a service-specific coalition formation mechanism is used to support the process parallelism of high performance services and this is presented in Section III-B.

A. Hierarchical resource management

Our resource management framework is based on a loosely coupled, logically hierarchical, decentralized management model across multiple layers. Each layer can be considered as a self-contained system/component, which may be influenced

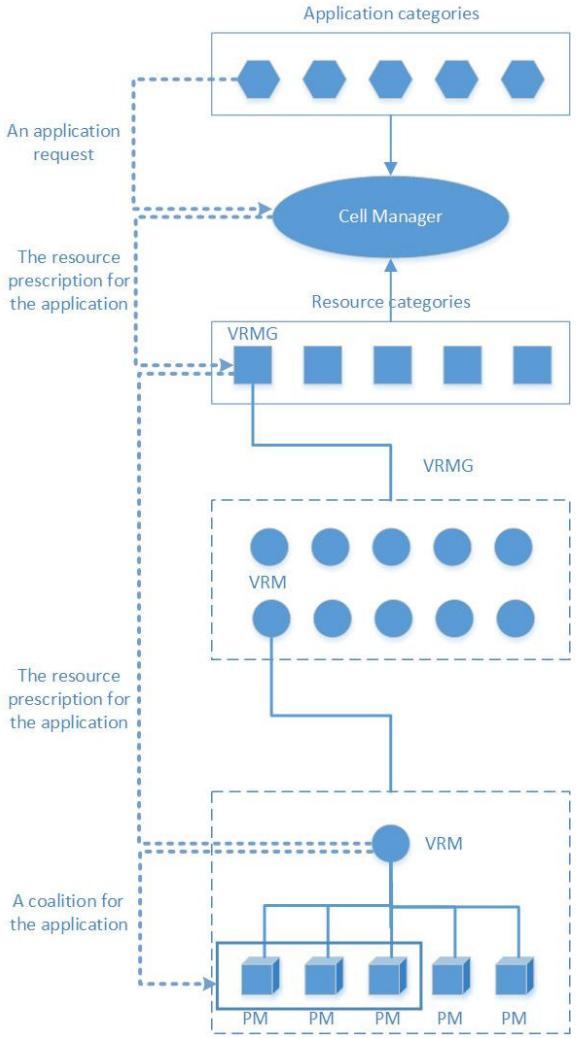

Fig. 1: Hierarchical cloud architecture for heterogeneous resource management and service delivery

by its neighboring layers for the purposes of system adaption and evolution overtime.

1) Cell Manager: At the top of the hierarchy, a Cell Manager partitions the space of heterogeneous resources into multiple zones. Each zone is composed of homogeneous hardware types and an associated resource abstraction method. In the CloudLightning architecture, heterogeneous resources are managed using various frameworks and platforms which are widely recognized to be effective and efficient for managing virtualized, containerized and bare metal resources, respectively [38]. For example, CloudLightning may use OpenStack Nova [39] to manage virtual machines on commodity servers, it may use Kubernetes [40], Mesos [41], and/or Docker Swarm [42] to manage containers on GPUs and MICs, and it may use OpenStack Ironic [43] to manage bare metal deployed on FPGAs.

The Cell Manager aims to make an optimal match between service requests and available resources based on multiple objectives, such as service level agreements (SLA), quality of service (QoS), and specific application constraints (e.g., high performance computing or high throughput computing). Fuzzy pattern recognition [44], heuristic algorithms [45], [46] and evolutionary algorithms [47] are commonly used in multi-objective optimization for high-level (i.e., coarse-grained) resource allocation, which attempt to find an appropriate solution between system performance (e.g., response time) and user-oriented constraints (e.g., SLAs). Similarly, liner programming or dynamic programming can be used to solve VM placement problems [48]–[50] for low-level (fine-grained) resource placement, focusing on optimizing resource utilization and power consumption [51].

2) vRack Managers within a zone: Each zone contains a group of vRack Managers (vRMs), each of which is a local resource manager sitting at the bottom of the local resource management hierarchy. Within a zone, vRMs can interact with each other using shared or distributed state information. In the shared state approach, vRMs communicate with each other through a locally centralized mechanism to decide on the best candidate to host the next service. This mechanism may use multi-objective optimization algorithms, and bidding algorithms, for example. In the distributed state approach, vRMs cooperate with neighbors to relocate/reassign the management of physical resources between/among different vRMs to maximize utility for each individual vRM involved. This mechanism may use cooperative game theory and self-organization approaches, for example.

Our previous work suggested using a market-based combinational auction [52] mechanism for delivering an appropriate set of resources in response to service requests. In this work, which is an example of the shared state approach, servers of a WSC bid to host services based on the requirements of those services. This approach was shown to out-perform traditional centralized resource management techniques. However, this study did not take account of resource virtualization, nor were critical problems like overbidding and temporal fragmentation addressed in previous studies.

The work presented in Section IV, which is an example of the distributed state approach, examines a self-organizing and self-managing system developed to demonstrate how vRMs compete and cooperate with in order to achieve local goals while managing virtualized resources.

3) vRack Manager: The layers of the hierarchy between the Cell manager and the vRMs are designed to guide resource requests associated with specific services to locations within the cloud that would be ”most suitable” to host them. This suitability is determined by the availability of resources and the selection of those resources to maximize the nonfunctional behaviors of the cloud. These behaviors include maximizing service delivery, resource utilization and quality of service and minimizing power consumption. The resource request guiding process is implemented using the concept of Perception (see Section IV-B1).

At the bottom level of the hierarchy, vRMs act as local resource managers, each managing a set of physical machines

(PMs). Since the number of these machine is limited and since their associated state information can be monitored frequently and accurately, tight feedback control loops can be established, resulting in efficient resource management. vRMs can make use of the state-of-the-art cloud techniques (e.g., virtualization, and containerization) and can tailor resource allocation to meet the need of specialized application workloads (see Section III-B).

B. Service-specific coalition formation

In general, a description of the resources needed to execute an associated service is created by the Cell Manager and propagated downward though the hierarchy. This request is called a resource prescription - since it prescribes the resources needed to execute a particular service. Some services are capable of exploiting process parallelism and to facilitate an efficient execution, it is necessary to identify a number of independent, co-located, resources that can be used for their execution. When a prescription associated with such a high-performance service is received by a vRM, an appropriate group of physical/virtual resources, known as coalition, are identified and managed by the associated vRM as a coherent compound resource dedicated to delivering the associated service.

There are several strategies for forming coalitions in terms of the application workload characterization. For example, an HPC workloads may require an isotropic distribution of resources for maintaining a balanced execution, since the efficient execution of an application depends solely on the slowest component, due to the tightly coupled execution path. An isotropy preserving strategy in a vRM can be realized by spreading the required VMs among available servers. An example of this strategy is algorithmically given in Algorithm 1.

However, if the application's internal communication is intensive, using a single physical server to accommodate all of the VMs the associated with the resource prescription may be more appropriate. This strategy can be described by Algorithm 2.

Algorithm 1

Let $C = \emptyset$ be an empty coalition
Let N_v be the number of VMs required by a resource prescription
Let N_c be the number of vCPUs per VM
Let N_m be the amount of memory per vCPU
function CFSYMMETRY(N_v, N_c, N_m)
    Let N_{vCPU}^{free} be a vector of the free vCPUs per server arranged in descending order
    Let I be the set with the indices of servers arranged with respect to N_{vCPU}^{free}
    Let N_{memory}^{free} be a vector of the available memory per server with respect to order of N_{vCPU}^{free}
    $counter = 0$
    while $|C| \leq N_v$ and $counter \leq N_v$ **do**
        for $i \leftarrow 1$ to $N_v - |C|$ **do**
            for $j \in I - C$ **do**
                if $\left(N_{vCPU}^{free}\right)_j \geq N_c$ and $\left(N_{memory}^{free}\right)_j \geq N_c N_m$ **then**
                    $C = C \cup \{j\}$
                    $\left(N_{vCPU}^{free}\right)_j = \left(N_{vCPU}^{free}\right)_j - N_c$
                    $\left(N_{memory}^{free}\right)_j = \left(N_{memory}^{free}\right)_j - N_c N_m$
                    $counter = counter + 1$
                    break
            if $counter = N_v$ **then**
                break
        Reorder $N_{vCPU}^{free}, N_{memory}^{free}$ and I with respect to free vCPUs
    if $|C| = N_v$ **then**
        return C
    else
        $C = \emptyset$
        return C

However, if the application's internal communication is intensive, using a single physical server to accommodate all of the VMs the associated with the resource prescription may be more appropriate. This strategy can be described by Algorithm 2.

IV. PROOF-OF-CONCEPT: A SELF-ORGANIZED, SELF-MANAGED, SYSTEM

Fig. 2 depicts the high level architecture for the proposed self-organized, self-managed (SOSM) system.

The Gateway is a front-facing component of the SOSM system, abstracting the inner workings of the back-end components and providing a unified access interface to the SOSM system. The lifecycle of a Blueprint is initiated when a Blueprint is chosen from the Blueprint Catalog, possibly augmented with specific constraints, compiled into a set of Blueprint Requirements and sent via the *Gateway* to a *Cell Manager*. The Cell Manager (CM) identifies one or more solutions meeting those requirements. It then chooses one of these solutions and subsequently sends it to the corresponding vRack Manager Group (vRMG). vRack Managers (vRMs) in the same Group are capable of self-organizing to meet specific objectives, such as reducing power consumption. To do this, they take action based on their local knowledge of underlying resource utilization. vRMs are aware of changes in the environment including new and disappearing resources and adapt, on a negotiated basis, with other vRMs within the same vRMG to meet system objectives.

A. CL-Resource

In pursuit of a service oriented architecture for the heterogeneous cloud, the SOSM system attempts to eliminate the concept of resource from its interactions with users. Instead, a service interface is created and utilization of all resources is the sole concern of SOSM system. To simplify the process of dealing with multiple physical and virtual resources based on commodity hardware in addition to specialized hardware resource types, the SOSM system uses a single abstract concept of resource, known as a CL-Resource. In response to a service request, the SOSM system identifies a specific CL-Resource that will be used for the delivery of that service. The physical realization of a CL-Resource depends on a number of factors. When dealing with commodity hardware, CL-

Algorithm 2

Let $C = \emptyset$ be the an empty coalition
Let N_v be the number of VMs required by a resource prescription
Let N_c be the number of vCPUs per VM
Let N_m be the amount of memory per vCPU
function CFDEPMIN(N_v, N_c, N_m)
    Let N_{vCPU}^{free} be a vector of the free vCPUs per server ordered with the servers that are not independent first
    Let I be the set with the indices of servers arranged with respect to N_{vCPU}^{free}
    Let N_{memory}^{free} be a vector of the available memory per server with respect to the order of N_{vCPU}^{free}
    for $i \leftarrow 1$ to N_v **do**
        for $j \in I$ **do**
            if $\left(N_{vCPU}^{free}\right)_j \geq N_c$ and $\left(N_{memory}^{free}\right)_j \geq N_c N_m$ **then**
                $C = C \cup \{j\}$
                $\left(N_{vCPU}^{free}\right)_j = \left(N_{vCPU}^{free}\right)_j - N_c$
                $\left(N_{memory}^{free}\right)_j = \left(N_{memory}^{free}\right)_j - N_c N_m$
                break
        Reorder $N_{vCPU}^{free}, N_{memory}^{free}$ and I with respect to free vCPUs
    if $|C| = N_v$ **then**
        return C
    else
        $C = \emptyset$
        return C

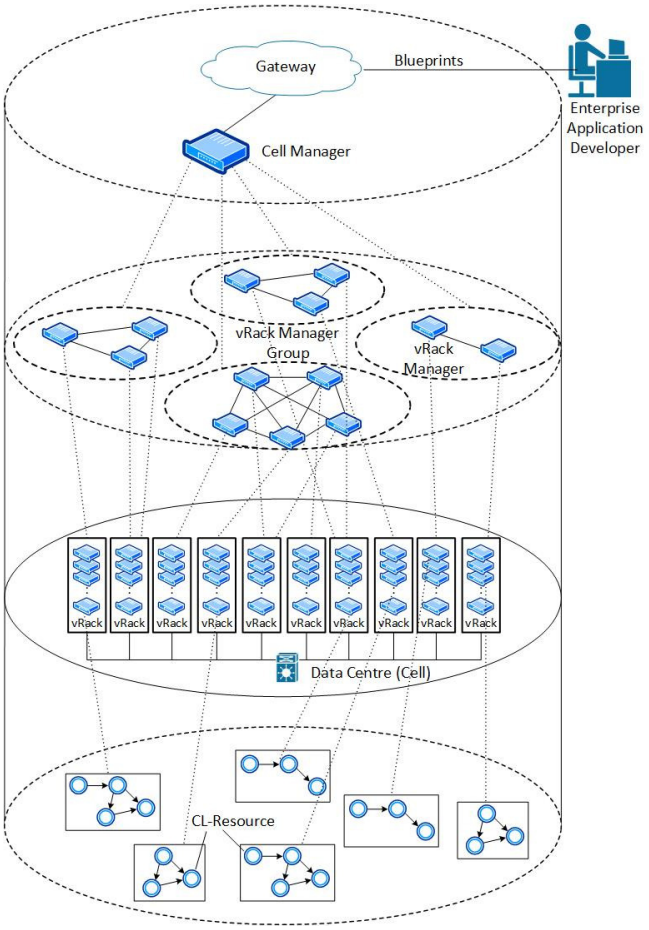

Fig. 2: Proof-of-concept: a self-organizing self-managing (SOSM) system overview

Resources can be bare metal, virtual machines, or containers. In addition, these virtual machines or containers may be created dynamically to suit specific services or they may be persistent and used to host a number of different services at different times. In the latter case, the CL-Resource is considered to be a static virtual resource. Networked commodity hardware may also be treated a single CL-Resource and either offered as a bare metal cluster or as a cluster pre-configured to host distributed applications, for example. Clusters of this type sitting on a dedicated high speed network constitute a specialized CL-Resource that may be employed to host distributed applications having a special requirement for low latency communications.

Servers with attached accelerators such as GPUs, MICs and FPGAs typically can not be virtualized due to the specific nature of the accelerators. As such, the server-accelerator pair are only offered currently as bare metal by cloud service providers. In the SOSM system, these server-accelerator pairs also constitute CL-Resources. In some cases, it may be possible to virtualize the server and to associate a partition of its accelerator with that virtualized component. In that case, the virtual component and the accelerator partition may be seen as a single CL-Resource. The granularity of a CL-Resource is thus dependent on what aspect of the underlying physical hardware is being exposed to the SOSM system.

An example of this can be seen in Fig. 3 which shows a MIC-world composed of four server-MIC pairs connected on a dedicated local network. The complete MIC-world may be exposed via its local resource manager to the SOSM system where it is seen as a single CL-Resource capable of running MIC-world services. In other configurations, the cluster of servers may be exposed as a networked cluster as described above. Yet, another option is to present collection of virtualized containers to the SOSM system, each representing a different CL-Resource. This concept of attaching resources to the SOSM system can be taken to the extreme by connecting specialized high performance machines, which may be composed of their own dedicated resource fabric. The characteristics of the resulting CL-Resources depend on how these machines are attached to the SOSM system. E.g., if they are attached in bare metal mode, they can be discovered and used to host services written explicitly to run on that bare metal hardware. If they are attached differently, the resulting CL-Resources may constitute entry points into the queuing systems of the local resource managers running on those machines.

Thus, CL-Resources can be categorized as follows:

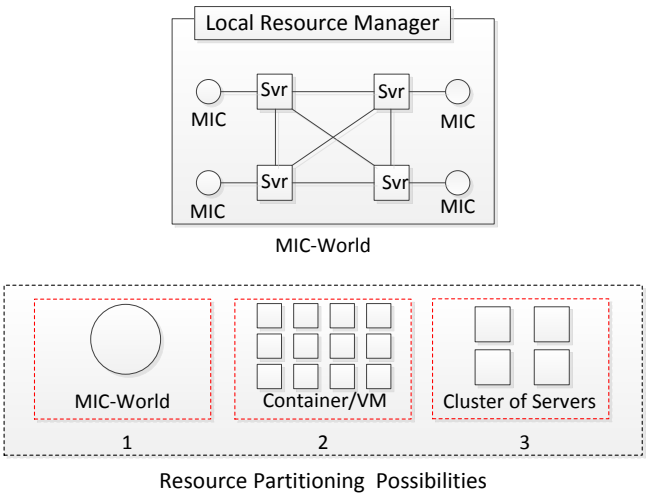

Fig. 3: CL-Resource Concept

1) Physical hardware as a single CL-Resource. This assumes that an appropriate management software exists for each hardware type within the SOSM system. The SOSM system can then use the corresponding APIs to manage this hardware resource.
2) Resource Manager (RM) as a single CL-Resource. A CL-Resource can be a resource manager, exposing the capacity and availability of its subsystem or cluster to the SOSM system. However, in this case, the SOSM system does not manage the subsystem or the cluster behind the resource manager directly. Instead, the SOSM system simply passes the appropriate resource prescriptions to the resource manager according to its real-time available capacities, and the resource manager takes care of the subsequent execution independently of the SOSM system.
3) A networked group of physical hardware as a single CL-Resource. There is benefit in representing hardware resources, of the same type, i.e. *T*, situated on a low-latency network, as a single CL-Resource under the control of the same vRM in the SOSM system. CL-Resources of this type can be placed under the same vRMG hosting CL-Resources representing individual physical hardware of type *T*. However, the CL-Resources representing the networked group and those representing the individual physical hardware, will usually be placed under different vRMs.

B. Self-organized, self-managed framework

A framework for a self-organized, self-managed, hierarchical architecture was proposed in [53].

1) Self-management mechanism: The CloudLightning SOSM system is composed of a number of components in each level in the hierarchy. Each of these components manages its own activity and communicates with neighboring components higher-up and further-down the hierarchy. Thus, a perception reflecting the ability of a component to accept new work is passed upwards and an impetus biasing the evolution of the system is passed downwards, where it becomes part of a weighted calculation affecting the future behavior of the system. The activities associated with self-management include the local calculation of perception and impetus based on the information received from components higher-up and lower-down in the hierarchy. These two concepts are integrated into a single value, known as a Suitability Index. The goal of each component is to maximize its Suitability Index such that perception and impetus achieve dynamic status. In practice, this status can never be achieved since the arrival of resource requests continually perturb the system. At lowest-level in the hierarchy, vRMs manage collections of physical machines under their control. At this level, perceptions are derived from monitoring information and impetuses become associated with weighted assessment functions [53], determining the relative important of these functions in achieving the global system objectives.

Metrics, Weights, Perception, Impetus and Suitability Index can be expressed more formally as:

- A **metric** (m) is a measure of a particular aspect of a components state.
- A **weight** (w) is an influencing factor, usually towards a local goal.
- **Suitability Index** is a measure of how close a component is to the desired state, and hence how suitable its operating characteristics are for contributing to the global goal.

$$\underset{\vec{w},\vec{m}\in\mathbb{R}^N}{\arg\max}\,\eta\left(\vec{I}(\vec{w}),\vec{P}(\vec{m})\right) \tag{1}$$

  where $\vec{w}$ is an N-dimensional vector of weights corresponding to the Impetus and $\vec{m}$ is an N-dimensional vector of metrics obtained from the lower levels.
- **Perception** is a function of the metrics from the immediate lower level in the hierarchy.

$$\vec{m}^{\ell} = \frac{1}{N_{\ell-1}}\sum_{i=1}^{N_{\ell-1}}\vec{m}_i^{\ell-1}, \ell > 1 \tag{2}$$

  where N_ℓ is the number of components in the ℓ-th level. For simplicity we choose the Perception of a component to be a function that averages metrics of its underlying components. The mean of the metrics of the underlying level is an approximation of the state of the underlying resources.
- **Impetus** is a function of the weights obtained from the immediate upper level in the hierarchy.

$$\vec{w}^{\ell} = \frac{1}{2}(\vec{w}^{\ell+1} + \vec{w'}^{\ell}), 0 \leq \ell < 3, \tag{3}$$

  where $\vec{w'}^{\ell}$ are the weights in the previous state and $\vec{w}^{\ell+1}$ are the weights propagated from the upper level. For simplicity, we choose the Impetus to be the average of the weights obtained from the upper level with the current weights of the component. The averaging function is used to attenuate the influence of upper levels to lower levels in order for the system to undergo a smoother transition towards the global goal.

The $\ell + 1$, ℓ and $\ell - 1$ represents the Cell Manager, the vRMGs and vRMs separately. The Cell Manager specifies a global goal state (e.g., to meet a specific business case), which can be expressed as weights and applied to the underlying vRMGs to steer their behavior in a particular direction, represented as $\vec{w}^{\ell+1}$ in Eq.3. An analogous process takes place in each level in the hierarchy.

2) Self-organization mechanism: To achieve local goals and to accommodate resource requests, it is sometimes necessary for components in the same level of the hierarchy to cooperate and to exchange the management role of certain resource. This self-organizing process is driven specific, reconfigurable, strategies. Some self-organization strategies include:

- Dominate: the component with the greater suitability index has precedence and can demand another component of the same type, but with a lower suitability index, to transfer some resources.
- Win-Win: components may cooperate to exchange resources to maximize the suitability index of each.
- Least Disruptive: minimize disruption with respect to management and administration
- Balanced: maximize load-balancing among each cooperating component
- Best Fit: minimize server fragmentation and/or minimize network latency (this strategy may come from some vRM specific objectives)
- Any meaningful combination of the above.

An example strategy demonstrating "Win-Win" shows how the self-organization works within the SOSM system. The "Win-Win" strategy is triggered by service request arriving at a certain vRM, which has the largest Suitability Index, but lacks the available resources to fulfill this prescription. However, available resources will be present in the same Cooperative. Thus, the vRM initiates the procedure of sending requests to the other vRMs to transfer their resources. If the available resources, before acquiring the new ones are less than half of the prescribed, then the vRM will not acquire them. Instead it initiates the creation of a new vRM which will manage the available free resources, if any, together with any newly acquired resources. The aforementioned process can be described by the following algorithmic procedure:

However, this self-organization strategy can be improved by acquiring resources, when necessary, by the vRacks with the maximum Suitability Indices. By using this technique the suitability index of the vRacks that are required to provide resources is enhanced, because their management costs are minimized. Thus, increasing performance and reducing fragmentation. This process can be described by the following algorithmic procedure:

Algorithm 3

Let j be the index of the vRack with maximum suitability index
Let rp be a resource prescription arriving to vRM_j
Let p_j be the set of $free$ resources belonging to vRM_j
function MINADMINCOSTS(rp)
  $a = \emptyset$
  $t = \emptyset$
  if $|p_j| < rp$ **then**
    $required = rp - |p_j|$
    for $i \leftarrow 1$ to N_v with $i \neq j$ **do**
      $send$ request to acquire free resources from vRM_i
      $receive$ p_i from vRM_i
      $required = required - |p_i|$
      $a = a \bigcup \{i\}$
      $t = t \bigcup p_i$
      if $required \leq 0$ **then**
        $remove$ exceeding resources from t
        $required = 0$
        break
    $send$ request to vRMs in a to acquire resources in t
    $receive$ resource handlers from vRMs in a
    if $|p_j| \geq rp/2$ **then**
      return $resource\ handles$ to Gateway Service
    else
      $create$ new vRM_k with resources $p_j \bigcup t$
      return $resource\ handles$ to Gateway Service
  else
    return $resource\ handles$ to Gateway Service

V. CONCLUSION

This paper presented a design for a service oriented architecture of a heterogeneous cloud. The cloud, once a collection of commodity hardware, is becoming more and more heterogeneous with the addition of hardware of different types. The trend for hardware vendors to create more and more specialized offering, capable of providing faster, more accurate and more power efficient solutions, looks set to continue. The increasing demand for this hardware and for access to high-performance computing is driving Cloud Service Providers (CSPs) to make evermore exotic IaaS offering available as bare metal. In this type of transaction, the CSP effectively rents the hardware to the customer. In this transaction, the financial interests of both parties are presumably met. However, from a resource utilization perspective, nothing definitive can be said: the customer has no incentive to use the resource efficiently so long as his needs are being met; the CSP no longer has control over the hardware for the duration of the rental period.

In the CL system, CSPs no longer offer Infrastructure as a Service. Instead the CSP undertakes to provide software services to the customer - effectively executing services on their behalf. From this perspective, the hardware is hidden from the customer. The customer no longer is concerned with *how* solutions are provided, they specify only *what* they want done. Hiding the hardware from the customer gives control back to the CSP to decide on how best to respond to customer needs and to balance these needs with its own, such as maximizing resource utilization. If the cloud is heterogeneous, that is, if it is composed of hardware of different types, and if the same

Algorithm 4

Let j be the index of the vRack with maximum suitability index
Let rp be a resource prescription arriving to vRM_j
Let p_j be the set of $free$ resources belonging to vRM_j
Let s be a vector containing the suitability indices of all $vRMs$ sorted in descending order
Let I_s be a vector containing the indices of vRMs with respect to vector s
function MAXSUITABILITYINDEX(rp)
  $a = \emptyset$
  $t = \emptyset$
  if $|p_j| < rp$ **then**
    $required = rp - |p_j|$
    for $k \leftarrow 2$ to N_v **do**
      $i = I_s(k)$
      $send$ request to acquire free resources from vRM_i
      $receive$ p_i from vRM_i
      $required = required - |p_i|$
      $a = a \bigcup \{i\}$
      $t = t \bigcup p_i$
      if $required \leq 0$ **then**
        $remove$ exceeding resources from t
        $required = 0$
        break
    $send$ request to vRMs in a to acquire resources in t
    $receive$ resource handlers from vRMs in a
    if $|p_j| \geq rp/2$ **then**
      return $resource\ handles$ to Gateway Service
    else
      $create$ new vRM_k with resources $p_j \bigcup t$
      return $resource\ handles$ to Gateway Service
  else
    return $resource\ handles$ to Gateway Service

service solution is available on a multiplicity of these hardware type, solutions with different cost/performance characteristics can potentially be offered to the customer. The complexity of managing resources in this way in an heterogeneous cloud environment should not be underestimated. An heterogeneous cloud at scale embodies many hardware types, each with different cost/performance/power profiles. This, together with attempting to satisfy the disparate needs of a large and varied customer community make the heterogeneous cloud a complex system. Complex systems cannot be managed effectively using a central resource manager. They need to employ tools suited for addressing complex systems. Self-organization and self-management are such tools and this is the approach taken by CloudLightning.

ACKNOWLEDGMENT

This work is funded by the European Union's Horizon 2020 Research and Innovation Programme through the CloudLightning project under Grant Agreement Number 643946.

REFERENCES

[1] Microsoft, "Steve Ballmer: Worldwide partner conference 2013 keynote," Press Release, Houston, Texas, Jul. 2013.
[2] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *Computer*, no. 12, pp. 33–37, 2007.
[3] D. C. Marinescu, A. Paya, J. P. Morrison, and P. Healy, "Distributed hierarchical control versus an economic model for cloud resource management," *arXiv preprint arXiv:1503.01061*, 2015.
[4] L. A. Barroso, J. Clidaras, and U. Hölzle, "The datacenter as a computer: An introduction to the design of warehouse-scale machines," *Synthesis lectures on computer architecture*, vol. 8, no. 3, pp. 1–154, 2013.
[5] L. Tang, J. Mars, X. Zhang, R. Hagmann, R. Hundt, and E. Tune, "Optimizing google's warehouse scale computers: The numa experience," in *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*. IEEE, 2013, pp. 188–197.
[6] M. Ahuja, C. C. Chen, R. Gottapu, J. Hallmann, W. Hasan, R. Johnson, M. Kozyrczak, R. Pabbati, N. Pandit, S. Pokuri *et al.*, "Peta-scale data warehousing at yahoo!" in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. ACM, 2009, pp. 855–862.
[7] J. Hauswald, M. A. Laurenzano, Y. Zhang, C. Li, A. Rovinski, A. Khurana, R. G. Dreslinski, T. Mudge, V. Petrucci, L. Tang *et al.*, "Sirius: An open end-to-end voice and vision personal assistant and its implications for future warehouse scale computers," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2015, pp. 223–238.
[8] J. C. Doyle, B. A. Francis, and A. R. Tannenbaum, *Feedback control theory*. Courier Corporation, 2013.
[9] Y. Lu, T. Abdelzaher, C. Lu, L. Sha, and X. Liu, "Feedback control with queueing-theoretic prediction for relative delay guarantees in web servers," in *Real-Time and Embedded Technology and Applications Symposium, 2003. Proceedings. The 9th IEEE*. IEEE, 2003, pp. 208–217.
[10] T. F. Abdelzaher, K. G. Shin, and N. Bhatti, "Performance guarantees for web server end-systems: A control-theoretical approach," *IEEE transactions on parallel and distributed systems*, vol. 13, no. 1, pp. 80–96, 2002.
[11] X. Wang and Y. Wang, "Coordinating power control and performance management for virtualized server clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 2, pp. 245–259, 2011.
[12] X. Wang, M. Chen, C. Lefurgy, and T. W. Keller, "Ship: A scalable hierarchical power control architecture for large-scale data centers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 1, pp. 168–176, 2012.
[13] S. Crago, K. Dunn, P. Eads, L. Hochstein, D.-I. Kang, M. Kang, D. Modium, K. Singh, J. Suh, and J. P. Walters, "Heterogeneous cloud computing," in *2011 IEEE International Conference on Cluster Computing*. IEEE, 2011, pp. 378–385.
[14] T. R. Scogland, C. P. Steffen, T. Wilde, F. Parent, S. Coghlan, N. Bates, W.-c. Feng, and E. Strohmaier, "A power-measurement methodology for large-scale, high-performance computing," in *Proceedings of the 5th ACM/SPEC international conference on Performance engineering*. ACM, 2014, pp. 149–159.
[15] G. Lee and R. H. Katz, "Heterogeneity-aware resource allocation and scheduling in the cloud." in *HotCloud*, 2011.
[16] J. Novet. (2016, November) Aws launches elastic gpus for ec2, fpga-backed f1 instances, r4 and refreshed t2, c5 and i3 coming in q1. [Online]. Available: http://venturebeat.com/2016/11/30/aws-launches-elastic-gpus-for-ec2-fpga-backed-f1-instances-r4-and-refreshed-t2-c5-and-i3-coming-in-q1/
[17] OpenStack Heterogeneous Accelerator Support , https://wiki.openstack.org/wiki/HeterogeneousInstanceTypes.
[18] S. Conway, C. Dekate, and E. Joseph, "Worldwide highperformance data analysis 2014–2018 forecast," *IDC, Doc*, vol. 248789, 2014.
[19] J. G. F. Coutinho, O. Pell, E. ONeill, P. Sanders, J. McGlone, P. Grigoras, W. Luk, and C. Ragusa, "Harness project: Managing heterogeneous computing resources for a cloud platform," in *International Symposium on Applied Reconfigurable Computing*. Springer, 2014, pp. 324–329.
[20] L. López, F. J. Nieto, T.-H. Velivassaki, S. Kosta, C.-H. Hong, R. Montella, I. Mavroidis, and C. Fernández, "Heterogeneous secure multi-level remote acceleration service for low-power integrated systems and devices," *Procedia Computer Science*, vol. 97, pp. 118–121, 2016.

[21] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center." in *NSDI*, vol. 11, 2011, pp. 22–22.
[22] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at google with borg," in *Proceedings of the Tenth European Conference on Computer Systems*. ACM, 2015, p. 18.
[23] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: flexible, scalable schedulers for large compute clusters," in *Proceedings of the 8th ACM European Conference on Computer Systems*. ACM, 2013, pp. 351–364.
[24] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Computing*, no. 3, pp. 81–84, 2014.
[25] "Simplicity and complexity in the description of nature," *Engineering and Science*, vol. 57, no. 3, pp. 2–9, 1988.
[26] P. Schuster, "Nonlinear dynamics from physics to biology," *Complexity*, vol. 12, no. 4, pp. 9–11, 2007.
[27] Y.-Y. Liu, J.-J. Slotine, and A.-L. Barabasi, "Controllability of complex networks," *Nature*, vol. 473, no. 7346, pp. 167–173, May 2011.
[28] P.-A. Noël, C. D. Brummitt, and R. M. D'Souza, "Controlling self-organizing dynamics on networks using models that self-organize," *Phys. Rev. Lett.*, vol. 111, p. 078701, Aug 2013.
[29] F. Heylighen and C. Gershenson, "The meaning of self-organization in computing," in *IEEE Intelligent Systems, Section Trends & Controversies - Sel-Organization and Information Systems*, 2003.
[30] A. M. Turing, "The chemical basis of morphogenesis," *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, vol. 237, no. 641, pp. 37–72, 1952.
[31] F. Heylighen *et al.*, "The science of self-organization and adaptivity," *The encyclopedia of life support systems*, vol. 5, no. 3, pp. 253–280, 2001.
[32] J. Kramer and J. Magee, "Self-managed systems: an architectural challenge," in *Future of Software Engineering, 2007. FOSE'07*. IEEE, 2007, pp. 259–268.
[33] M. Puviani and R. Frei, "Self-management for cloud computing," in *Science and Information Conference (SAI), 2013*. IEEE, 2013, pp. 940–946.
[34] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of internet services and applications*, vol. 1, no. 1, pp. 7–18, 2010.
[35] M. Parashar and S. Hariri, "Autonomic computing: An overview," in *Unconventional Programming Paradigms*. Springer, 2005, pp. 257–269.
[36] D. C. Marinescu, A. Paya, J. P. Morrison, and P. Healy, "An auction-driven self-organizing cloud delivery model," *arXiv preprint arXiv:1312.2998*, 2013.
[37] I. Brandic, "Towards self-manageable cloud services," in *2009 33rd Annual IEEE International Computer Software and Applications Conference*, vol. 2. IEEE, 2009, pp. 128–133.
[38] D. Dong, H. Xiong, P. Stack and J. P. Morrison, "Managing and Unifying Heterogeneous Resources in Cloud Environments," *The 7th International Conference on Cloud Computing and Services Science (CLOSER 2017), 24-26 April 2017, Porto, Portugal*.
[39] OpenStack Nova, http://docs.openstack.org/developer/nova/.
[40] Kubernetes, http://kubernetes.io/.
[41] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation (NSDI 2011)*, 2011, pp. 295–308.
[42] Docker Swarm, https://github.com/docker/swarm.
[43] OpenStack Ironic, http://docs.openstack.org/developer/ironic/deploy/user-guide.html.
[44] Z. Wang and X. Su, "Dynamically hierarchical resource-allocation algorithm in cloud computing environment," *The Journal of Supercomputing*, vol. 71, no. 7, pp. 2748–2766, 2015.
[45] A. Konak, D. W. Coit, and A. E. Smith, "Multi-objective optimization using genetic algorithms: A tutorial," *Reliability Engineering & System Safety*, vol. 91, no. 9, pp. 992–1007, 2006.
[46] T. Saber, A. Ventresque, J. Marques-Silva, J. Thorburn, and L. Murphy, "Milp for the multi-objective vm reassignment problem," in *Tools with Artificial Intelligence (ICTAI), 2015 IEEE 27th International Conference on*. IEEE, 2015, pp. 41–48.
[47] E. Zitzler and L. Thiele, "Multiobjective optimization using evolutionary algorithmsa comparative case study," in *International Conference on Parallel Problem Solving from Nature*. Springer, 1998, pp. 292–301.
[48] A. Beloglazov and R. Buyya, "Energy efficient resource management in virtualized cloud data centers," in *Proceedings of the 2010 10th IEEE/ACM international conference on cluster, cloud and grid computing*. IEEE Computer Society, 2010, pp. 826–831.
[49] X. Li, Z. Qian, S. Lu, and J. Wu, "Energy efficient virtual machine placement algorithm with balanced and improved resource utilization in a data center," *Mathematical and Computer Modelling*, vol. 58, no. 5, pp. 1222–1235, 2013.
[50] J. Dong, X. Jin, H. Wang, Y. Li, P. Zhang, and S. Cheng, "Energy-saving virtual machine placement in cloud data centers," in *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*. IEEE, 2013, pp. 618–624.
[51] S. Srikantaiah, A. Kansal, and F. Zhao, "Energy aware consolidation for cloud computing," in *Proceedings of the 2008 conference on Power aware computing and systems*, vol. 10. San Diego, California, 2008, pp. 1–5.
[52] D. C. Marinescu, A. Paya, and J. P. Morrison, "Coalition formation and combinatorial auctions; applications to self-organization and self-management in utility computing," *arXiv preprint arXiv:1406.7487*, 2014.
[53] C. Filelis-Papadopoulos, H. Xiong, A. Spataru, G. Castane, D. Dong, G. Gravvanis and J. P. Morrison, "A Generic Framework Supporting Self-organisation and Self-management in Hierarchical Systems," *The 16th International Symposium on Parallel and Distributed Computing (ISPDC 2017), 3-6 July 2017, Innsbruck, Austria*.