Managing and Unifying Heterogeneous Resources in Cloud Environments

Dapeng Dong, Paul Stack, Huanhuan Xiong and John P. Morrison
Boole Centre for Research in Informatics, University College Cork, Western Road, Cork, Ireland

Keywords: Architecture, Heterogeneous Resource, Platform Integration, Cloud, HPC.

Abstract: A mechanism for accessing heterogeneous resources through the integration of various cloud management platforms is presented. In this scheme, hardware resources are offered using virtualization, containerization and as bare metal. Traditional management frameworks for managing these offerings are employed and invoked using a novel resource coordinator. This coordinator also provides an interface for cloud consumers to deploy applications on the underlying heterogeneous resources. The realization of this scheme in the context of the CloudLightning project is presented and a demonstrative use case is given to illustrate the applicability of the proposed solution.

1 INTRODUCTION

Cloud computing is maturing and evolving at pace. This evolution is mainly driven by consumer needs and technological advancement. Recently, there has been increasing demand to support High Performance Computing (HPC) applications, such as weather forecast (Krishnan et al., 2014), medical imaging (Serrano et al., 2014), and fluid simulation (Zaspel and Griebel, 2011). These application have traditionally been confined in clustering environments. To effectively support these applications and to demonstrate comparable performances in the cloud, specialized hardware and networking configurations are required. These requirements pose challenges for effectively integrating and efficiently managing a wide variety of heterogeneous resources in clouds.

Currently, several frameworks and platforms exist for managing virtualized environments (such as, OpenStack Nova (Nova, 2016)), container environments (such as, Kubernetes (Kubernetes, 2016), Mesos (Hindman et al., 2011), and Docker Swarm (Docker Swarm, 2016)), containers in virtualized environments (such as, OpenStack Magnum (Magnum, 2016)), bare metal servers (such as, OpenStack Ironic (Ironic, 2016)), and traditional cluster management frameworks for High Performance Computing and High Throughput Computing (HPC/HTC) workloads. These frameworks and platforms have sufficiently matured and/or have begun to find practical applications in many public and private clouds. However, in general, current data center deployment focuses on managing homogeneous resources through a single resource abstraction method. The scale and diversity of HPC application migrating to the cloud dictates that multiple resource abstraction methods should be simultaneously available in a single cloud deployment.

In this paper, a number of generically applicable techniques are introduced to address the management challenges associated with the evolution of the current homogeneous cloud infrastructure into the heterogeneous cloud infrastructure required to support HPC applications. We present a unified platform for managing heterogeneous hardware resources including general purpose Central Processing Units (CPUs), high performance GPUs, and specialized computation units, for example, Many Integrated Cores (MICs) and Data Flow Engines (DFEs), as well as cluster computing environments such as High Performance Computing, High Throughput Computing (HPC/HTC), and Non-uniform Memory Access (NUMA) machines, these techniques have been investigated and applied in the context of the CloudLightning project (Lynn et al., 2016).

The following explores some relevant related work and present mechanisms that abstract the underlying virtualization methodologies and exploits each appropriately to support the diverse ecosystem of HPC applications hosted on heterogeneous hardware resources. Subsequently, the realization of these mechanisms in the context of the CloudLightning

Dong, D., Stack, P., Xiong, H. and Morrison, J.
Managing and Unifying Heterogeneous Resources in Cloud Environments.
DOI: 10.5220/0006300901430150
In *Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER 2017)*, pages 115-122
ISBN: 978-989-758-243-1

project(Lynn et al., 2016) is described. and a demonstrative use case application is given to illustrate the applicability of the proposed solution. Finally, some conclusions are drawn.

2 RELATED WORK

The rapid adoption of cloud computing in both public and private sectors is resulting in hyper-scale cloud deployment. This trend poses challenges to cloud management and cloud architecture design. Existing cloud platforms regardless whether they make use of virtualization, containerization or bare metal offerings all focus on the management of homogeneous resources with respect to the desirable non-functional requirements, for example, scalability and elasticity.

Google Borg (Verma et al., 2015) is a platform for managing large-scale bare metal environments used by Google, internally. Borg manages tens of thousands of servers simultaneously. The Borg architecture consists of three main component types: Borg masters, job schedulers, and Borglet agents. A typical Borg instance consists of a single Borg master, a single job scheduler and multiple Borglet agents. The Borg master is the central point for managing and scheduling jobs and requests. A Borg master and job scheduler are replicated in several copies for high-availability purpose, however, only a single Borg master and a single job scheduler are active at one time. This centralized management approach requires Borg masters and job schedulers (the original and all the replicas) to be large enough to scale out as required. The Borg job scheduler may potentially manage a very high volume of jobs at simultaneously, this has made Borg more suitable for long-running services and batch jobs, since that those job profiles reduce the loads on the job scheduler. In contrast, Fuxi (Zhang et al., 2014) platform from Alibaba Inc., uses a similar monolithic scheduling approach, but with incremental communication and locality tree mechanisms that support rapid decision making.

More contemporary systems are becoming distributed to address the scalability issue, nevertheless, masters continue to retain their centralized management approach. In contrast, job schedulers are becoming ever more decentralized in their management decisions. This decentralized approach sometimes results in scheduling conflicts, however, the probability of this happening is low. Examples of these systems include Google Omega (Schwarzkopf et al., 2013) and Microsoft Apollo (Boutin et al., 2014). Google Omega employs multiple schedulers working in parallel to speed up resource allocation and job scheduling. Since there is no explicit communication between these schedulers, it cannot be said that this approach improves resource allocation and job scheduling decisions, rather it increases the number of such decisions being made per unit time. Microsoft Apollo (Boutin et al., 2014) employs a similar scheduling framework. But it is also incorporates global knowledge that can be used by each scheduler to make optimistic scheduling decisions. Apollo enables each scheduler to reason about future resource availability and implement a deferred correction mechanism to optimistically defer any corrections until after tasks are dispatched. Identified potential conflicts may not be realized in some situations since the global knowledge is by definition imperfect. Consequently, all other thing being equal, by delaying conflict resolution to the latest possible opportunity, at which time they may disappear, Apollo may perform better than Google Omega. Google Borg, Google Omega and Microsoft Apollo work with bare metal servers and schedule jobs onto physical nodes. In contrast, Kubernetes, Mesos and OpenStack attempt to improve resource utilization by introducing containerization and virtualization.

Kubernetes (Kubernetes, 2016)(Burns et al., 2016) is another Google technology and an evolution of Google Omega. In the Kubernetes system, schedulers cooperate in making scheduling decisions and hence attempt to improve resource allocation. This cooperation comes at the cost of sharing the entire cluster's status information, whereas, conflicting scheduling decisions can be avoided, this comes at the cost of dynamically making the distributed scheduling decisions. Kubernetes is designed to work exclusively with containers as a resource management technology. It improves service deployment and resource management in a complex distributed container environment.

Apache Mesos (Hindman et al., 2011) is another management platform which enables multiple different scheduling frameworks to manage the same environment. This is achieved by employing a coordinator service assigning resources controls to a single scheduler during its decision making processes. This can potentially lead to an inefficient use of resources when the request is lightweight and available resources are significantly large.

OpenStack (Nova, 2016) is an open-source cloud platform focusing on the management of a virtualization environment. OpenStack uses a front-end API server to receive requests and a centralized coordinator service (*nova-conductor*) for coordinating various components (e.g., networking, image, storage, and compute). The *nova-conductor* uses a scheduler

service (*nova-scheduler*) to find physical server(s) for deploying virtual machine(s) based on the configurations specified in the user requests (iterative filter) and together with *weight* of each of the available physical server in the cloud. Multiple conductors and multiple scheduler may be created in a OpenStack environment, these components working a partition domains, hence conflicts cannot rise.

3 HETEROGENEOUS RESOURCE INTEGRATION

Different resource hardware types require appropriate resource management techniques. A goal of this paper is to support resource heterogeneity in pursuit of HPC in the cloud. A consequence of this goal is the need to realize a mechanism for integration heterogeneous resources and their respectively management techniques in a single unified scheme. An overview of the proposed scheme is shown in Figure 1.

In this scheme, hardware resources are virtually partitioned based on the abstraction methods (virtualization, containerization, bare metal, and shared queues) most appropriate for the respective hardware type. A corresponding management framework is then adopted to manage groups of hardware of the same type. A central Resource Coordinator component is provided as an interface to be used by cloud consumers to deploy applications on the underlining resources. More importantly, the Resource Coordinator component coordinates the deployment for cloud application components on, potentially, various types of resources across those virtual partitions.

3.1 Service Delivery Work-flow

To fully exploit the benefits offered by these versatile service and resource options. It is necessary to carefully manage the resources used in providing them. This can be challenging for both the service provider and for the service consumer, especially when the components of a cloud application may require to be deployed on different types of resource. Moreover, leaving aside the difficulties of working with those heterogeneous hardware environment, to fully exploit these hardware resources and accelerators, expert knowledge related to the deployment of cloud application components is usually also required. However, this configuration complexity and deep domain-specific knowledge should be made transparent to consumers. The approach is to allow consumers to compose their tasks into a work-flow of its constituent service(s). Work-flows of this kind are often referred to as *blueprints*.

A cloud application blueprint can be designed using graphical interfaces. An entity in a blueprint presents a functional component of the cloud application and its associated resources and necessary configurations; connections indicate the communication channels between components.

A cloud application blueprint is firstly submitted to a Resource Coordinator. Many Resource Coordinators may potentially work in parallel to load-balance requests arriving at high frequency. Each cloud application blueprint is processed by a single Resource Coordinator. The Resource Coordinator decomposes the blueprint into groups of resource requests depending on the resource abstraction types. For example, a complex blueprint, requiring coordinating across different management platforms, may describe an application that requires front-end web servers to collect data which is subsequently processed using accelerators, thus, the resources required for this blueprint deployment may be a set of virtual machines running on CPUs managed by OpenStack, for example, and a set of containers running on servers having Xeon Phi co-processors and managed by Mesos. After the blueprint decomposition process, the Resource Coordinator analyses the relationships between the groups of resource requests and makes further amendments to the blueprint to realize the communications between blueprint components that will be deployed across virtual resource partitions. The Resource Coordinator then forwards each group of resource requests to designated virtual resource partitions that are managed by corresponding management platforms.

3.2 Platform Integration

Heterogeneous hardware resources are managed through various frameworks and platforms. This may raise interoperability issues, however, as each platform manages a virtual partition of the resources, in the same management domain, the resulting interoperability issues reduced to a technical integration action and are not exacerbated by having to consider the interests of multiple entities. Figure 1 shows how the integration scheme may use OpenStack Nova (Nova, 2016) to manage virtual machines, may use Kubernetes (Kubernetes, 2016), Mesos (Hindman et al., 2011), and/or Docker Swarm (Docker Swarm, 2016) to manage containers, and may use OpenStack Ironic (Ironic, 2016) to manage bare metal servers. Each platform offers a different set of Application Programming Interfaces (APIs) and utilities for sim-

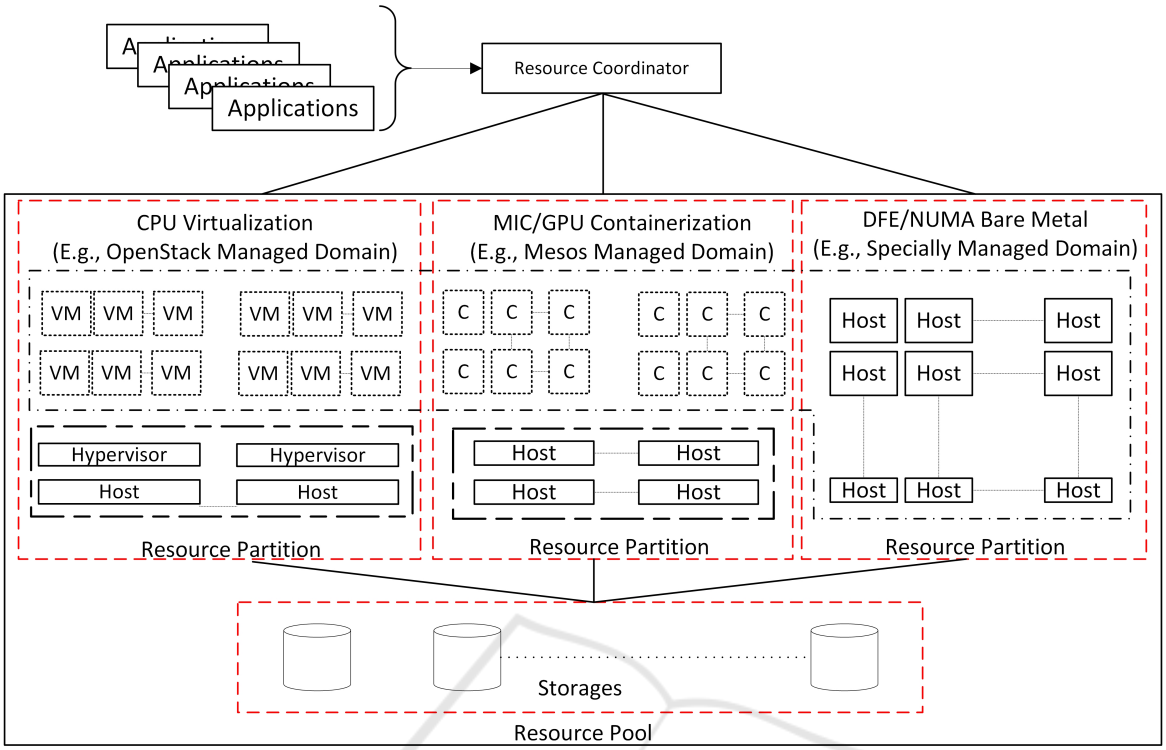

Figure 1: Managing and accommodating heterogeneous hardware resources through multiple integrated platforms.

ilar resource management operations, such as, creating virtual machines and/or containers. The Resource Coordinator uses a Plug & Play Interface that defines a set of common operations for managing underlying resources, and these operations are then translated to platform-specific API calls or commands using the Plug & Play implementation modules to carry out service deployment processes. Additionally, storage systems are organized and managed independently. Processing units can be easily configured to use volume-based and/or flat storage systems.

3.3 Networking Integration Strategy

Two schemes are available for networking integration. The first scheme is to treat networking in each virtual partition independently as shown in Figure 2. Cloud application components are deployed independently in their corresponding virtual partition and virtual networks are created accordingly within each virtual partition. After the independent cloud application components deployment, network bridges are created in order to establish communication channels across virtual partitions. The scheme does not require any modification to the respective resource management frameworks. This gives the flexibility of integrating other resource management frameworks, for example, Kubernetes and Docker Swarms. The concerns about

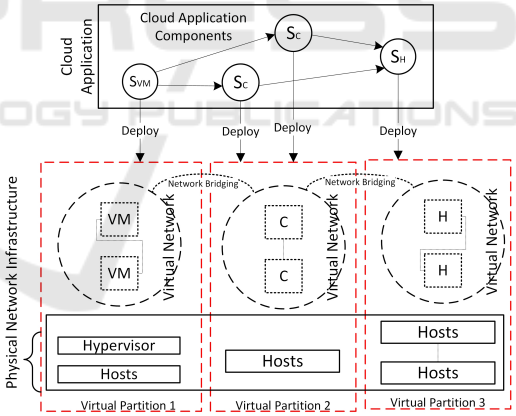

Figure 2: Networking integration scheme 1.

this scheme arise from the differences associated with each of the networking approaches taken by each of the the respective resource management frameworks. Considering that different platforms offer different type of network services at various level, for example, an OpenStack managed network often uses the Neutron framework, which offers rich functionalities including firewalls, load-balancers, etc., these may not be available in the container environment if it is managed by Mesos.

The second scheme employs the Neutron framework (OpenStack Neutron,) for building and manag-

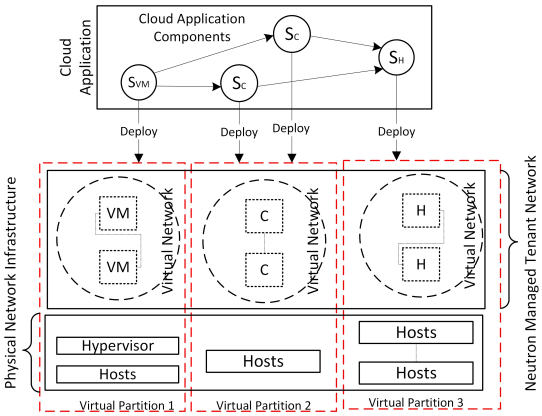

Figure 3: Networking integration scheme 2.

ing virtual network infrastructure. Figure 3 shows the simplified networking plan. All hardware resources are connected to the same networking infrastructure, but logically, they are managed by corresponding platforms independently. From a consumers perspective, all resources are in a single resource pool. In the case that multiple components of a single cloud application need to be deployed on both virtual machines and containers which are managed by different platforms, this requires a dedicated virtual network for the cloud application over the tenant network. Thus, there is a need for a unified virtual network infrastructure management component across all platforms horizontally. In addition, the tenant networks must be managed in a seamless fashion. The second networking planning scheme adopts OpenStack Neutron for this purpose. In general, frameworks and services developed under the OpenStack Big Tent Governance natively support Neutron services. In contrast, container technologies such as Kubernetes, Mesos, and Docker Swarm employ different networking models. For example, Kubernetes can use Flannel (Flannel, 2016), Weave Net (WeaveNet, 2016) frameworks operating in various modes; Docker uses libnetwork (Libnetwork, 2016) by default. In the context of this work, the Kuryr network driver (Kuryr, 2016) is employed to link Neutron and container networks. Thus, consumers of clouds will experience seamless connections between all types of heterogeneous hardware resources.

4 EMPIRICAL STUDY

The initial implementation and the deployment of the proposed scheme has been realized in the context of CloudLightning project (Lynn et al., 2016)[change to link]. To cover a wide range different types

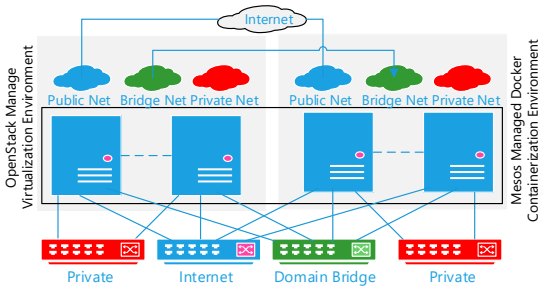

Figure 4: Testbed configuration.

of HPC applications in clouds, the CloudLightning project concentrate on three dispirit use cases: Oil & Gas simulation, Genomic Processing, and Ray-Tracing. In the following experiment, the thrird use case based on the Intel's Ray-Tracing application (Embree, 2016) is used to demonstrate the need for a unified platform to manage a cloud environment composed of heterogeneous resources.

4.1 Testbed Configuration

The experimental environment consists of an OpenStack managed virtualization environment (Newton release) which consists of eight Dell C6145 compute servers in total having 384 cores, 1.4TB RAM, 12TB storage and a Mesos managed Docker containerization environment (v1.1.0) which consists of five IBM 326e servers in total having 10 cores, 40GB RAM, 200GB storage. In this deployment configuration, all physical servers have multiple dedicated network connections to three different networks including a *public*, a *private* and a *bridge* network. The *public* network connects to the Internet, the *private* networks are private to OpenStack or Mesos, the *bridge* network provides interconnections between virtual machines (managed by OpenStack) and containers (managed by Mesos). In the context of OpenStack, the *private* network is equivalent to the Neutron *Tenant* network, the *public* and *bridge* networks are the Neutron *Provider* networks. In the Mesos managed Docker environment, three Docker Bridge networks are created with each connecting to the *public*, *private* and *bridge* network respectively. This deployment configuration is flexible to allow for future platforms, if needed, to be integrated with the existing environments.

4.2 Use Case Blueprint

The Intel's Ray-Tracing application use case is composed of two parts, one is the Ray-Tracing engine and two is a Web interface. Both the engine and

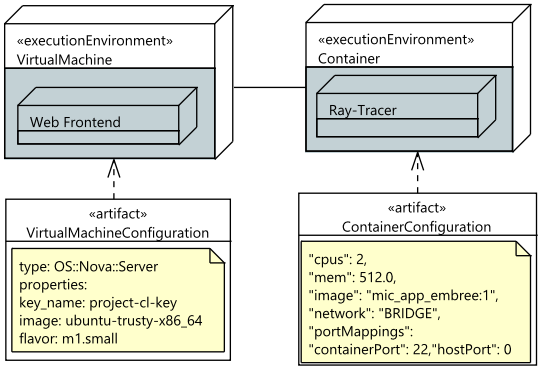

Figure 5: Ray-Tracer use case blueprint.

the Web interfaces should be respectively deployed on the most appropriate back-end resources including virtual machines and containers with access to hardware accelerators, such as Intel Multiple Integrate Core (MIC). In this experiment, a blueprint is constructed which specifies that the Web interface should be deployed on virtual machines and the Ray-Tracing engine should be deployed in a container. In particular, it has been demonstrated that the Ray-Tracing application can gain better performance when run on MICs (Benthin et al., 2012) (Wald, 2012), which in general requires containerization or bare metal servers.

A graphical representation of the use case blueprint is shown in Figure 5. This blueprint is expressed using XML. A blueprint consists of four main components: (1) Execution Environments, specifying the resource types such as virtual machines, containers, bare metal, and so on. (2) The Cloud Application, detailing the software component(s) to be deployed in an Execution Environment. (3) Artifacts, containing configurations for each Execution Environment or cloud application components. (4) Connections, specifying the connectivity between Execution Environments.

In the CloudLightning environment, there is a clear separation of concerns between the cloud application description and the resources on which that application will eventually run. The CloudLightning environment uses a SOSM engine to dynamically determine the most appropriate resources available at that time within the cloud resource fabric to host a particular application. These resources are dynamically written into the application blueprint once they have been discovered and a deployment engine subsequently deploys the application component onto those resources as required. In the CloudLightning system a number of components including the SOSM engine together act as the Resource Coordinator.

The Resource Coordinator is responsible for parsing, decomposing and transforming blueprint components to a format, that can be understood by the underlying cloud management platforms, to facilitate application deployment.

The Resource Coordinator categorizes Execution Environments in to groups based on resource types (EE-Group), such as virtual machines, containers, or bare metal. Within each EE-Group, Execution Environments will be further partitioned into separate groups based on connectivities (C-Group), for example, if another given blueprint consists of three virtual machines without specifying connections between them, then this blueprint will be partitioned into one EE-Group and three C-Group within that EE-Group. In the use case scenario described there, there are two EE-groups, and one C-Group within each EE-Group. This grouping can be determined by formulating the blueprint topology into a graph $G(V,E)$ by identifying connectivities using the Union-Find algorithm as illustrated in Algorithm 1. Where V indicates the vertices in the graph corresponding to the execution environments in the blueprint, and E denotes the edges in the graph corresponding to the connections between Execution Environments.

Data: G(V, E)
Result: List{C-Group{v}}
for $v_i \subseteq V$ *in* G **do**
  new C-Group$_i${v_i}
end
foreach *Edge* $e(v_i, v_j) : E$ **do**
  C-Group$_i$ = find(e.v$_i$);
  C-Group$_j$ = find(e.v$_j$);
  if *C-Group*$_i$ == *C-Group*$_j$ **then**
    continue;
  end
  else
    union(C-Group$_i$, C-Group$_j$)
  end
end

Algorithm 1: Blueprint execution environment grouping using Union-Find.

The algorithm assumes the connections are symmetric (if Execution Environment A is connected to Execution Environment B, then B is connected to A) and transitive (if Execution Environment A is connected to B, B is connected to C, then A is connected to C). Additional constraints can be added to make blueprint connections asymmetric and non-transitive.

When the grouping process is completed, the Resource Coordinator seeks connections between C-Groups across EE-Groups. A connection indicates the Execution Environments from both C-Groups should be placed in the *bridge* network or need to be at-

```
blueprint-id: c97e718674c34adf815316ad4cec93cf
heat_template_version: 2016-10-14
resources:
  embree_web_frontend:
    type: OS::Nova::Server
    properties:
      image: Ubuntu14.04_LTS_svr_x86_64
      flavor: m1.small
      key_name: cl-project
      networks:
        - network: bridge-provider
      user_data:
        template: |
          #!/bin/bash -v
          apt -y install httpd
          ......
```

Figure 6: Web Frontend in OpenStack managed virtual machines.

```
{"blueprint-id" : "c97e718674c34adf815316ad4cec93cf",
  curl -X POST -H "Content-type: application/json"
  marathon:8080/v2/apps -d '{
   "id" : "embree",
   "cpus" : 2,
   "mem" : 512.0,
   "container" : {
     "type" : "DOCKER",
     "docker": {
       "image" : "mic-app-embree:1",
       "network": "BRIDGE",
       "portMappings":[{"containerPort":22,"hostPort":0}]
} }  }' }
```

Figure 7: Ray-Tracer in Mesos managed Docker containers using Marathon.

tached to the *bridge* network, to establish cross platform communications. Execution Environments from completely isolated C-Groups should be placed in a *private* network, if Internet access is desired, then each Execution Environment must be attached to the *public* network, independently.

Once the networks are identified, Execution Environments with their corresponding configurations in each EE-Group will be transformed into deployment templates that are compatible with the corresponding management platforms. The snapshot of the deployment templates of the Ray-tracing application are shown in Figure 6 and 7. The Resource Coordinator initiates the deployment process and subsequently manages the life-cycle of the blueprint.

5 CONCLUSIONS

In this work, mechanisms are introduced for provisioning heterogeneous resources through the integration of various existing platforms in which each platform manages a set of homogeneous hardware resources independently. Globally, all types of resources are virtually presented in a unified resource pool to consumers. It is must be noted that, in some circumstances, for example, an orchestrated service that has been deployed on various types of resources across different platforms, may encounter network congestion issues. Additionally, as each management platforms (e.g., OpenStack and Mesos) have their built-in resource schedulers, the proposed schemes are limited in how they control and optimize resources at a coarse-grained level. To this end, a unified cloud platform that can natively support heterogeneous hardware is needed. The CloudLightning project is attempting to provide initial solutions to this challenge. This sets the directions for the future work.

ACKNOWLEDGEMENT

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

REFERENCES

Benthin, C., Wald, I., Woop, S., Ernst, M., and Mark, W. R. (2012). Combining single and packet-ray tracing for arbitrary ray distributions on the intel mic architecture. *IEEE Transactions on Visualization and Computer Graphics*, 18(9):1438–1448.

Boutin, E., Ekanayake, J., Lin, W., Shi, B., Zhou, J., Qian, Z., Wu, M., and Zhou, L. (2014). Apollo: Scalable and coordinated scheduling for cloud-scale computing. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, pages 285–300, Berkeley, CA, USA. USENIX Association.

Burns, B., Grant, B., Oppenheimer, D., Brewer, E., and Wilkes, J. (2016). Borg, omega, and kubernetes. *Commun. ACM*, 59(5):50–57.

Docker Swarm (2016). `https://github.com/docker/swarm`. [Accessed on 15-June-2016].

Embree, I. (2016). `https://embree.github.io`. [Accessed on 05-December-2016].

Flannel (2016). `https://github.com/coreos/flannel#flannel`. [Accessed on 16-June-2016].

Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A. D., Katz, R., Shenker, S., and Stoica, I. (2011). 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'11, pages 295–308, Berkeley, CA, USA. USENIX Association.

Ironic, O. (2016). http://docs.openstack.org/developer/ironic/deploy/user-guide.html. [Accessed on 14-June-2016].

Krishnan, S. P. T., Krishnan, S. P. T., Veeravalli, B., Krishna, V. H., and Sheng, W. C. (2014). Performance characterisation and evaluation of wrf model on cloud and hpc architectures. In *High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICESS), 2014 IEEE Intl Conf on*, pages 1280–1287.

Kubernetes (2016). http://kubernetes.io/. [Accessed on 14-June-2016].

Kuryr (2016). http://docs.openstack.org/developer/kuryr/. [Accessed on 14-June-2016].

Libnetwork (2016). https://github.com/docker/libnetwork. [Accessed on 16-June-2016].

Lynn, T., Xiong, H., Dong, D., Momani, B., Gravvanis, G., Filelis-Papadopoulos, C., Elster, A., Khan, M. M. Z. M., Tzovaras, D., Giannoutakis, K., Petcu, D., Neagul, M., Dragon, I., Kuppudayar, P., Natarajan, S., McGrath, M., Gaydadjiev, G., Becker, T., Gourinovitch, A., Kenny, D., and Morrison, J. (2016). Cloudlightning: A framework for a self-organising and self-managing heterogeneous cloud. In *Proceedings of the 6th International Conference on Cloud Computing and Services Science*, pages 333–338.

Magnum, O. (2016). https://github.com/openstack/magnum. [Accessed on 13-June-2016].

Nova, O. (2016). http://docs.openstack.org/developer/nova/. [Accessed on 14-June-2016].

OpenStack Neutron. https://github.com/openstack/neutron. [Accessed on 14-June-2016].

Schwarzkopf, M., Konwinski, A., Abd-El-Malek, M., and Wilkes, J. (2013). Omega: Flexible, scalable schedulers for large compute clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems*, EuroSys '13, pages 351–364, New York, NY, USA. ACM.

Serrano, E., Bermejo, G., Blas, J. G., and Carretero, J. (2014). Evaluation of the feasibility of making large-scale x-ray tomography reconstructions on clouds. In *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, pages 748–754.

Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E., and Wilkes, J. (2015). Large-scale cluster management at google with borg. In *Proceedings of the Tenth European Conference on Computer Systems*, EuroSys '15, pages 18:1–18:17, New York, NY, USA. ACM.

Wald, I. (2012). Fast construction of sah bvhs on the intel many integrated core (mic) architecture. *IEEE Transactions on Visualization and Computer Graphics*, 18(1):47–57.

WeaveNet, W. (2016). https://www.weave.works/docs/net/latest/introducing-weave/. [Accessed on 16-June-2016].

Zaspel, P. and Griebel, M. (2011). Massively parallel fluid simulations on amazon's hpc cloud. In *Network Cloud Computing and Applications (NCCA), 2011 First International Symposium on*, pages 73–78.

Zhang, Z., Li, C., Tao, Y., Yang, R., Tang, H., and Xu, J. (2014). Fuxi: A fault-tolerant resource management and job scheduling system at internet scale. *Proc. VLDB Endow.*, 7(13):1393–1404.