# A Flexible Inter-locale Virtual Cloud For Nearly Real-time Big Data Applications

Huan Zhou*‡, Paul Martin*, Jinshu Su‡, Cees de Laat* and Zhiming Zhao*
*University of Amsterdam, Faculty of Science, Amsterdam, 1098XH, Netherlands
‡National University of Defense Technology, Changsha, China
Email: {h.zhou, p.w.martin}@uva.nl, sjs@nudt.edu.cn, {delaat, z.zhao}@uva.nl

***Abstract*—The rapid growth of emerging applications such as social network analysis and ecosystem monitoring has led to an explosion in the amount of data being generated. On the other hand, the increased capabilities of cloud computing afford more flexible computing resources to achieve better performance. However, it is still difficult to determine how to provision resources to fit large scale applications across different locales. It is essential not only to be able to manage the greater volumes of data but also to ensure that the QoS (quality of service) requirements of processing are reliably satisfied. This paper presents a Cloud engine that can provision a flexible network-transparent virtual Cloud. In this virtual Cloud, complex virtual infrastructure is distributed over multiple data centres or Cloud providers. The Cloud engine can help application developers to customise and provision virtual infrastructure for applications with critical performance constraints that may influence how and where application components should be hosted. First, we introduce the capacity gap between such applications and the Cloud. We then describe the cloud engine along with its key technologies, including transparent network connectivity and standardised multi-level infrastructure descriptions, as well as its applicability to particular scenarios. Following these descriptions, we discuss and demonstrate how to apply the Cloud engine to help in a specific big data application use case and satisfy its QoS constraints.**

***Keywords*—Cloud engine; nearly real-time; networked infrastructure; inter-locale.**

## I. Introduction

Nowadays, big data applications not only consider the amount of data they can process but also consider the processing time. Data needs to be processed and fed back to users in order to inform decisions and improve runtime steering of applications. This constitutes the so-called 'nearly real-time' constraint on application feedback and steering. Network transmission time is an important factor to influence the processing time. However, data collectors may be distributed in different locations for certain large-scale big data applications [1]. There can be a significant physical distance between the data collector and the data user. For distributed applications executed over the Internet, the effective processing time is often mainly determined by the network transmission time.

This kind of big data application is a type of quality critical application, which often requires customised virtual infrastructure with tailored SLAs (service level agreements) when migrated into a cloud environment [2]. The Cloud can be a powerful solution for big data processing, because the computing ability of servers is constantly increasing and more data centers are being set up around the world that can be used to support these applications. It is still a problem however for application developers to access and manage their cloud resources. There exist studies which discuss using Cloud to solve big data application challenges: Ji et al. [3] introduce some famous tools and Cloud platforms to do large-scale big data processing, while Rajiv [1] proposes a high-level architecture of large-scale data processing service. The underlying resource layer of the overall architecture is scalable across multiple data centers or even Clouds, but both of these studies do not mention how to help application developers manage virtual Cloud resources and achieve better performance. Moreover, some other Cloud tools, such as Chef [4], only focus on SaaS (software-as-a-service) platforms to orchestrate services, or do not take nearly real-time constraints into account.

In response to the current academic and industrial state-of-the-art, we design and implement a flexible inter-locale Cloud engine for quality critical applications to help satisfy nearly real-time requirements for highly distributed big data processing. This Cloud engine is able to provision a networked infrastructure, recover from sudden failures quickly, and scale across data centers or Clouds automatically. The key technologies used include transparent network connection and standardised multi-level infrastructure description. In this paper, we use the scenario of a nearly real-time large scale big data application as an example to describe our engine's operation.

The remainder of this paper is organised as follows. Section 2 describes the architecture of large-scale big data applications and introduces the challenges of migrating these kinds of application. Section 3 presents the key technologies of our Cloud engine and its possible application scenarios. Section 4 proposes a solution for a specific use case and describes how to apply it using our engine. We then do some experiments to demonstrate the feasibility of this solution in Section 5. Finally, Section 6 summarises the particular innovations of the engine and concludes the paper.

## II. Problem statement and challenges

Figure 1 illustrates a typical architecture and stages of a big data application. The kind of nearly real-time big data application we discuss in this paper however is particularly large-scale. The data collectors can be distributed all around the world, especially when the big data application is combined with IoT (Internet of Things) [5] or sensor networks [6]. An eddy covariance data processing service [7] is an example of a typical nearly real-time big data application. It measures wind and gas concentration at sites over different ecosystems. It

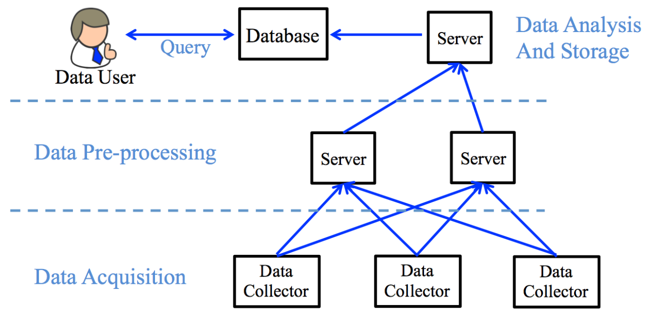


Fig. 1. Typical architecture of big data application

keeps collecting these data all year round and it need to calculate the net exchange of gases, energy and temperature between ecosystems and atmosphere under some time constraints.

According to the current state of the Cloud in industry, we infer the following challenges and gaps when migrating this kind of quality critical application onto Cloud, focusing mainly on infrastructure provisioning.

**1) Networked infrastructure.** The applications workflow becomes more complex with a lot of components that need to communicate with each other. Separated instances cannot complete the whole job. For instance, the components in different stages in Figure 1 need to communicate with each other and transfer data. The virtual infrastructure must therefore realise a particular network topology. Most current cloud providers cannot support this however; for example, Amazon EC2 can only allow users to describe private subnets, making it hard to build a complete topology.

**2) Nearly real-time constraints.** Nearly real-time applications require that most task deadlines be met over the lifetime of the application. Missing one deadline does not lead to immediate failure of the application, but continued failure to meet deadlines is unacceptable. We identify two particular types of nearly real-time constraints in this paper. The first type is of static constraints on network transmission time as data is processed, which restrict task scheduling before provisioning. The second type is of runtime constraints restricting the time the application has to recover from sudden failures—because the application is running all the time and some failures cannot be avoided, especially where the Cloud is remote and not totally reliable. Currently developers generally put all components in one data centre. If that data centre is not accessible, then we have to re-provision the whole infrastructure within another data centre, which is a costly operation.

**3) Geography.** Figure 1 shows that not all the components of an application are on the cloud. Data collectors such as (for example) cameras providing video of a live event are not on the Cloud themselves. The geographic location of any virtual infrastructure therefore has to be considered to satisfy the nearly real-time constraints on data delivery [6].

**4) Auto provisioning and federated cloud.** Since these applications are complex, we need a way to provision the whole infrastructure and deploy applications automatically. Currently, some tools can only provision automatically at instance level, for example Chef [4]. On the other hand, we may need more resources from other Clouds to provision a large scale infrastructure [8]. It is a problem to combine these resources across multiple locales however.

## III. Inter-locale Cloud Engine

### A. *Methodology*

To address these challenges, we design and develop a Cloud engine to set up the virtual Cloud. This virtual Cloud is an encapsulation of different data centres or other Clouds. With the help of this Cloud engine, the Cloud user can provision networked virtual infrastructure and manage all virtual resources together on the one virtual Cloud. This engine relies on transparent network connection methods and standardised multi-level infrastructure descriptions.

This engine applies two different methods to settle the problem of connectivity between partitioned topologies in different locales, which is a key step for provisioning across multiple data centres or Clouds. These two connection methods have been briefly discussed in our previous paper [9].

Figure 2 illustrates the first connection method. It shows how one packet gets through the public network between two sub-topologies. It is mainly based on NAT. The proxy node works as a mirror of the node in another topology and is not made visible to the Cloud user. At the same time, VM1 and VM2 can communicate via private IP addresses, which are selected by the application developer.

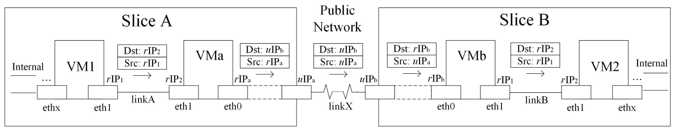


Fig. 2. Connection technique with proxy nodes.

The second method to connect these sub-topologies is using IP tunnelling. This method is shown in Figure 3. With the IP tunnelling technique, the original packet, which uses the internal private network addresses provided by the application developer, can be wrapped in another packet which allows the original packet to be delivered through the public network.

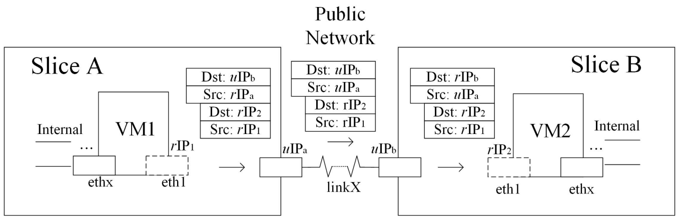
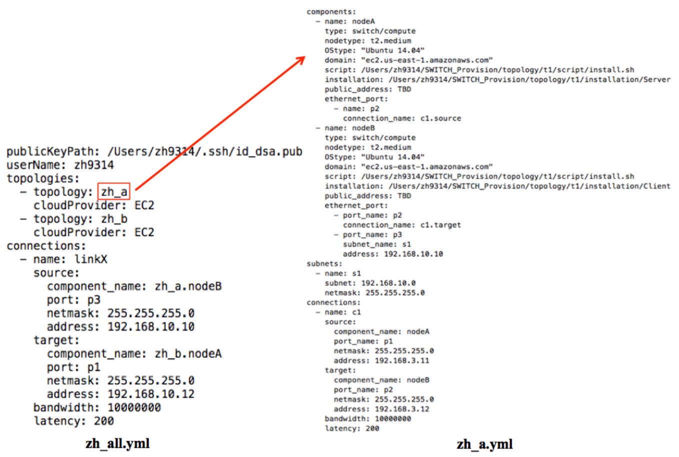


Fig. 3. Connection method with IP tunnel.

The advantage of the second method is that it does not add the extra overhead of proxy nodes for every link that crosses sub-topologies, in contrast with the first method. However, only some versions of Linux support IP tunnelling by default. If the customer adopts (for example) Windows for the virtual machines to run on, then the second method cannot be easily made to work. Another disadvantage of the second method is that we need to re-configure the original nodes provided by the developer. It is therefore not totally transparent when



```
publicKeyPath: /Users/zh9314/.ssh/id_dsa.pub
userName: zh9314
topologies:
  - topology: zh_a
    cloudProvider: EC2
  - topology: zh_b
    cloudProvider: EC2
connections:
  - name: linkX
    source:
      component_name: zh_a.nodeB
      port: p3
      netmask: 255.255.255.0
      address: 192.168.10.10
    target:
      component_name: zh_b.nodeA
      port: p1
      netmask: 255.255.255.0
      address: 192.168.10.12
    bandwidth: 10000000
    latency: 200
```

**zh_all.yml**

```
components:
  - name: nodeA
    type: switch/compute
    nodetype: t2.medium
    OStype: "Ubuntu 14.04"
    domain: "ec2.us-east-1.amazonaws.com"
    script: /Users/zh9314/SWITCH_Provision/topology/t1/script/install.sh
    installation: /Users/zh9314/SWITCH_Provision/topology/t1/installation/Server
    public_address: TBD
    ethernet_port:
      - name: p2
        connection_name: c1.source
  - name: nodeB
    type: switch/compute
    nodetype: t2.medium
    OStype: "Ubuntu 14.04"
    domain: "ec2.us-east-1.amazonaws.com"
    script: /Users/zh9314/SWITCH_Provision/topology/t1/script/install.sh
    installation: /Users/zh9314/SWITCH_Provision/topology/t1/installation/Client
    public_address: TBD
    ethernet_port:
      - port_name: p2
        connection_name: c1.target
      - port_name: p3
        subnet_name: s1
        address: 192.168.10.10
subnets:
  - name: s1
    subnet: 192.168.10.0
    netmask: 255.255.255.0
connections:
  - name: c1
    source:
      component_name: nodeA
      port_name: p1
      netmask: 255.255.255.0
      address: 192.168.3.11
    target:
      component_name: nodeB
      port_name: p2
      netmask: 255.255.255.0
      address: 192.168.3.12
    bandwidth: 10000000
    latency: 200
```

**zh_a.yml**

Fig. 4. Infrastructure description.

compared with the proxy node method. We therefore adopt both methods and choose which one to apply depending on the specific situation. Meanwhile, we have tested to confirm that the network performance will not significantly drop with use of either of these methods.

Another key part of our solution lies with infrastructure description. The infrastructure specification used by our engine adopts the YAML format, which is human readable and allows for compatibility with TOSCA (Topology and Orchestration Specification for Cloud Applications) standard. The multi-level description is used to provision infrastructure provided by different data centres or even different Clouds. Figure 4 shows an example of the files used.

In Figure 4 the file `zh_all.yml` provides a top-level infrastructure description. It specifies different sub-topologies and their providers. The field "topologies" defines the whole topology. The subfield "topology" of this field defines the name of the sub-topology. It is also the name of the low-level description file, which describes the detailed infrastructure further. The user should also define which cloud provider this sub-topology belongs to. The field "connections" describes how the two sub-topologies are connected. Besides these, the fields "publicKeyPath" and "userName" are important to set up the virtual Cloud. The user generates a RSA key pair. He keeps the private key and publishes the public key within the field "publicKeyPath". After the virtual resources are provisioned, the user can then login to every instance with the corresponding private key and the user name defined in the configuration file. Otherwise, the user would need different private keys to access resources from different cloud providers. The default user-name would also be different.

File `zh_a.yml` is an example of the low-level infrastructure description. The infrastructure resources described in one file are all in one data centre. The field "components" describes the computing resources of VM nodes. The fields "subnets" and "connections" describe the network resources. Among them, the field "subnets" is used to describe several nodes in one subnet. The field "connections" defines a specific link between two nodes. This field makes it easy to describe the network topology. It is worth mentioning that the user can specify the installation file and installation script path in each node description. With these fields, the applications developed by developers can be automatically deployed after provisioning.

These files are human readable and standardised. It is easy for application developers to directly design the infrastructure with these files. These files can also be generated by other components, such as an automated infrastructure planner; however, that is beyond the scope of this paper.

## *B. Application scenarios*

The Cloud engine can be used in a number of scenarios to satisfy the static and runtime requirements of big data applications.

**1) Provisioning networked infrastructure.** While the user can describe network topologies using networked infrastructure providers such as ExoGENI [10], the user cannot get network topology on other providers such as EC2 or EGI [11] Cloud, as shown in Figure 5. EC2 and EGI Cloud represent the current state of most cloud providers whether private or public. With our Cloud engine, the user can describe his own network topology even on these Clouds by defining the field "connections" in infrastructure descriptions. In addition, it is transparent to the provider, which means that the cloud provider does not need to do anything to support this feature. Thus our Cloud engine is able to set up a networked virtual Cloud across these Clouds.

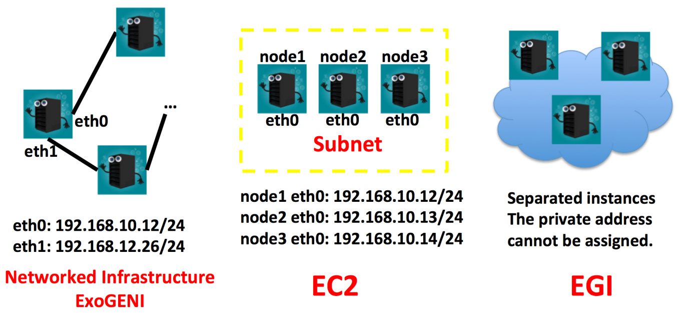


Fig. 5. Provisioning networked infrastructure.

**2) Fast failure recovery.** Figure 6 describes the process of failure recovery with our Cloud engine. There are two key components of the Cloud engine that are relevant to this scenario: the provisioning agent and the monitoring agent. When some data centre is down or inaccessible, a probe previously installed on the node can detect this. The monitoring agent can then invoke the provisioning agent to perform recovery. The provisioning agent then just needs to provision the specific part of the application hosted on the failed infrastructure. As the infrastructure description is already partitioned, it is easy for the agent to provision the same topology in another data centre. Meanwhile, the connection method will keep the topology identical to the previous one. From the application point of view, the topology is the same and the application does not need to be changed. Avoiding the re-provisioning of the whole infrastructure can save a lot of time and make the overall infrastructure more reliable.

**3) Auto scaling among data centers or Clouds.** Currently, the user can only define an auto-scaling group in one data

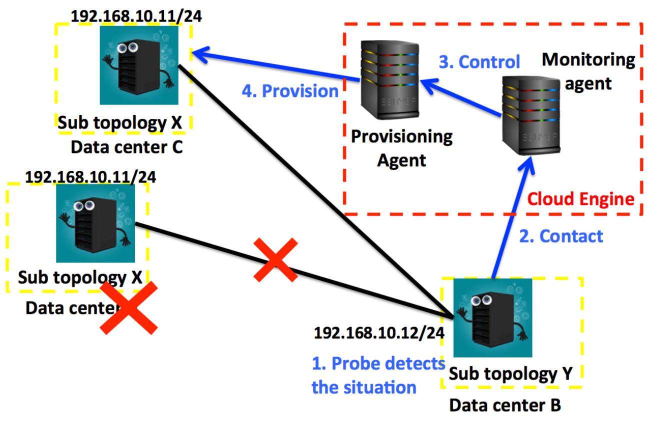


Fig. 6. Fast failure recovery.

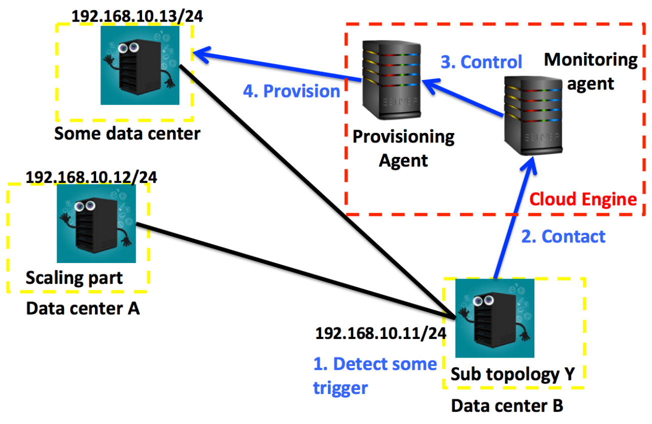


Fig. 7. Auto-scaling among data centres or Clouds.

centre as in the example of Amazon EC2. Moreover, most cloud providers do not even afford this function. With our Cloud engine, the user just needs to define an address pool for auto-scaling. Figure 7 shows the process. The scaling part can then be provisioned from another data centre or Cloud at runtime. More importantly, the address pool can be defined in the range of private IP addresses. The application can then be configured to know where the scaling part is before execution. Otherwise, the application needs to be configured manually at runtime. This is also useful for large-scale applications; when the resources are exhausted or limited in one data centre or Cloud currently in use, the Cloud engine can make the infrastructure scale-out to use resources from other locations.

## IV. Solution for Nearly Real-Time Big Data Application

Following the typical architecture of big data application shown in Figure 1, we propose a solution architecture for those kinds of nearly real-time big data application using our Cloud engine, specifically for the use case of an eddy covariance data processing service as formulated within the ENVRIPLUS project, shown in Figure 8. In the ENVRIPLUS project use case, the data collectors are significantly spread out geographically and often do not have high-quality network access to the Internet. If the data collector is geographically far from the processing server, then the network performance will be too low to satisfy real-time requirements, for instance to transfer a certain amount of data within a particular time limit. Jiang et al. [12] points out however that the emergence of private back-bones in recent years to connect globally distributed data centers can serve as a readily available infrastructure for a managed overlay network. Haq et al. [13] use cloud-based overlays to afford a packet recovery service. We can adopt this idea of using the cloud-based network instead of the pure Internet-based network to try and satisfy the nearly real-time requirements of the application.

The problem is that it is very hard for the application developer to apply and manage many virtual resources. He or she needs to interact with different data centres or even cloud providers. It is also difficult for them to develop their applications, because while if the application is deployed in one data centre, then all the components on different hosts can communicate with each other with predefined private IP addresses, if the components are hosted on different data centres then communication addresses can only be known after the instances are provisioned. Hence, the application developer cannot design his or her own infrastructure and there is still a lot of manual configuration work to be done to make the components connected after all resources are provisioned and the relevant public addresses are known.

The Cloud engine developed by us is therefore a key component of the architecture shown in Figure 8, acting as an automatic provisioning agent which not only provisions the virtual computing resources but also the network resources. With the connection techniques mentioned in Section 3, the Cloud engine is able to integrate resources from different data centres or Clouds as a single infrastructure. From the application developer's point of view, the Cloud engine set up a virtual Cloud for them. They do not need to consider about where the virtual resources come from. They can always use their own account and private key, which are defined in the infrastructure description files, to manage all the resources on the virtual Cloud. With this help, developers can focus on designing the infrastructure with the necessary private IP addresses and develop their applications. The Cloud engine can then take the infrastructure description files as input to provision resources and run the application automatically.

According to the use case, the data collectors collect data from different ecosystems. In order to satisfy the first type of nearly real-time constraint mentioned in Section 2, the Cloud

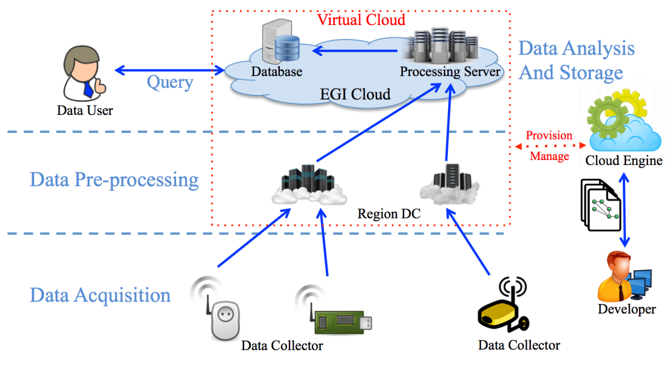


Fig. 8. Solution architecture for nearly real-time big data application.

TABLE I. PROPERTIES OF OBJECTS IN THE EXPERIMENT

| Number | Subject | Computing Properties | | | Access Network Properties | | | Geography Properties | |
|---|---|---|---|---|---|---|---|---|---|
| | | CPU Core | OS | Memory | Mode | Upload | Download | Cloud Provider | Location |
| ① | Laptop | 1 GB | MacOS | 8 | WIFI[1] | 0.94 Mbps | 8.59 Mbps | -[2] | Amsterdam |
| ② | Laptop | 1 GB | MacOS | 8 | WIFI[3] | 193 Mbps | 305 Mbps | - | UvA |
| ③ | VM | 1 GB | Ubuntu 14.04 | 3 | Ethernet | - | - | ExoGENI | UvA |
| ④ | VM | 1 GB | Ubuntu 14.04 | 3 | Ethernet | - | - | ExoGENI | CA, US |

[1] It is connected with the home network.
[2] '-' means unknown.
[3] It is connected with Eduroam.

engine must provision the resources from regional data centres close to data collectors. As EGI Cloud only has data centres in Europe, we can choose other cloud providers as supplements, such as EC2, ExoGENI, etc. The reduced latency between the acquisition and pre-processing stages makes data collectors forward data more efficiently. The network performance between pre-processing and analysis is better than that of directly sending all data from collectors to the final processing servers. It is out of the scope of this paper to actually determine how best to distribute resources across data centres to satisfy time constraints, but our engine provides the mechanisms to perform the optimal distribution when it is identified. During runtime, the application must keep running all year round. The Cloud engine makes it recover from failures fast to satisfy the second type of nearly real-time constraint as discussed in Section 3. Moreover, the scalability across Clouds makes the infrastructure more flexible for meeting dynamic constraints at runtime.

## V. Evaluation and Analysis

In this section, we set up experiments to test the feasibility of the solution provided by the Cloud engine, supplementing the network experiments of [9]. In order to simulate the real situation, we create four objects in the experiment. The detailed properties of these objects are listed in Table I. We use a laptop to act in the role of data collector and put it in different network environments. For object 1, the laptop is connected with the home network via WIFI. This object is designed to simulate the situation where the data collector is far from the regional data centre and does not have a particularly good network connection. Object 2 is deployed within the campus network of UvA (University of Amsterdam) to simulate the situation where the data collector is close to the regional data centre and does have a very good network connection. Objects 3 and 4 are two VM nodes provisioned by our Cloud engine within different locales provided by the ExoGENI infrastructure platform. They are connected via private IP addresses far from each other geographically. We adopt the second connection method described in Section 2 (IP tunnelling). Object 3 acts in the role of virtual resources provisioned in the regional data centre in Figure 8 while object 4 acts in the role of remote virtual resources close to the data user. There are two main scenarios we need to compare in this section. The first scenario is the deployment of all the components in one data centre without use of our engine. The second scenario is to adopt our solution, which is to distribute the components on the virtual Cloud set up by our engine.

We therefore design the first experiment to test the latency in these two scenarios. The results are shown in Figure 9. We start sixty ping requests one by one between different objects of Table I. From the legend in the figure, we can tell which link between two objects each plot belongs to. In addition, "S1" preceding the legend indicates that it refers to the first scenario (without engine) described above and "S2" for the second scenario (with engine). It is obvious that the latency is lower when the data collector is closer to the server. In the first scenario without our solution, despite the fact that the data collector has good network connectivity, the average latencies are nearly ten times higher than those in the second scenario. Moreover, the latencies in scenario 1 are not stable, especially when network access is bad, which is common for real data collectors. It is also worth pointing out that real data collectors are typically not as powerful as the laptop used in this experiment, and so the real performance may be even worse. Adopting our solution dramatically reduces the latency.

The second experiment is to test the bandwidth in these two scenarios. Figure 10 shows the results. We measure the bandwidth continuously over 200 seconds. The corresponding y-axis of all blue lines in this figure is on the left, measured in Mbps. The corresponding y-axis of the green line is on the right, measured in Kbps. This figure shows that the quality of the cloud-based network is better. The link between the two VMs (objects 3 and 4) provisioned by our Cloud engine use a cloud-based network which exhibits superior bandwidth. If we deploy the application without our solution, data collectors are needed to directly connect to the faraway server. Two lines in Figure 10 with "S1" denotes the performance. Although the object 2 is in a very good network environment, the average bandwidth is 26 Mbps less when it is directly connected to the faraway server. Moreover, it is obvious that the bandwidth of the cloud-based network is more stable. In addition, the green line shows that when data collectors do not have a good access network, the bandwidth is much worse.

The transmission time for data collectors can therefore be reduced using our solution. Our Cloud engine can set up a virtual Cloud that considers the underlying network in order to better satisfy the nearly real-time requirements of the application to the extent that it is possible. This kind of consideration is essential for data collectors to work more efficiently as part of a larger distributed system. In this paper, we mainly test the static nearly real-time constraints above. For the runtime nearly real-time constraints, we have demonstrated it partially in paper [9] and will test it further in the future.

## VI. Innovations and Conclusion

There are several innovations demonstrated by our Cloud engine. These innovations can help satisfy the requirements of quality-critical applications such as those described earlier.

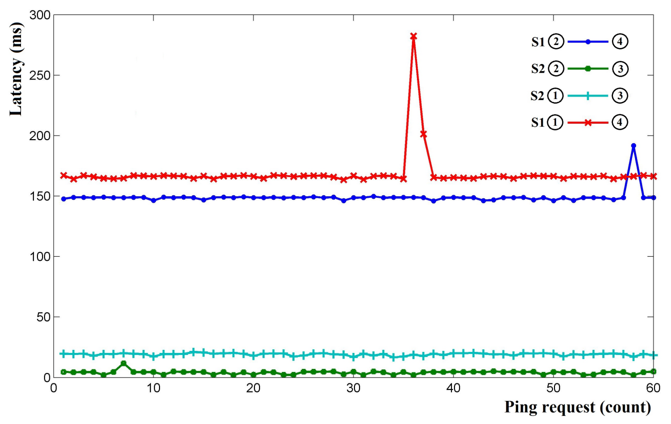


Fig. 9. Latency comparison.

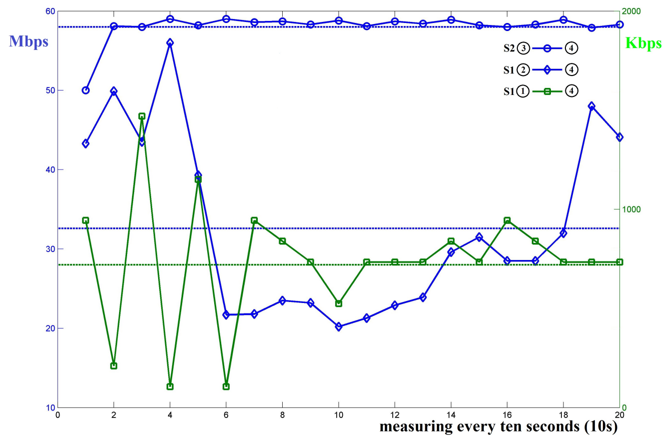


Fig. 10. Bandwidth comparison.

**1) Fast and flexible.** Multiple smaller infrastructures can be provisioned with less overhead. On the other hand, if some part of the infrastructure crashes, we just need to re-provision the smaller sub-infrastructure containing the failed component, not the whole aggregate infrastructure. This property can minimise violations of the real-time constraints of some quality-critical applications. Flexibility in where parts of the application are provisioned can also help satisfy any geographic requirements of the application.

**2) Flexible scaling.** As cloud providers often have limitations on the scale of infrastructure provisioned for a particular application, our mechanism puts forward a way to provision large-scale infrastructure across multiple domains. The infrastructure can then even scale across cloud providers.

**3) Transparency.** Our mechanism is not only transparent to cloud providers but also to cloud users. From the providers' point of view, there is nothing required of them to support this kind of provisioning. From the point of view of application developers, the infrastructure is provisioned as designed, including selected IP addresses, the precise locations of components hidden in the network configuration. It is also totally transparent to use the tools like Hadoop or Spark, as long as they are configured with the proper private IP addresses.

**4) Standardised infrastructure level auto-provisioning.** The Cloud engine only takes as input description files like those illustrated in Figure 4. The files are human readable and can be written compatible with the emerging TOSCA standard. Hence, they are easy to standardise. Compared with other automatic provisioning tools, it not only provisions the separate instances but also the network as defined by the user. Moreover, the application can be installed and run automatically after the infrastructure is provisioned.

This paper presents the Cloud engine we have been developing. With this engine, application developers can design and deploy their applications on an inter-locale virtual Cloud. We are drawing upon specific use cases from the ENVRIPLUS project in order to test our approach in the context of large scale distributed environmental research, and have proposed a solution architecture for such applications. The results of the simple experiments we have so far conducted demonstrate the feasibility and potential efficiency of our solution.

## VII. Acknowledgments

This research has received funding from the European Union's Horizon 2020 research and innovation program under grant agreements 643963 (SWITCH project), 654182 (ENVRIPLUS project) and 676247 (VRE4EIC project). The research is also partially funded by the COMMIT project.

## References

[1] R. Ranjan, "Streaming big data processing in datacenter clouds," *IEEE Cloud Computing*, vol. 1, no. 1, pp. 78–83, 2014.

[2] Z. Zhao, P. Martin, J. Wang, A. Taal, A. Jones, I. Taylor, V. Stankovski, I. G. Vega, G. Suciu, A. Ulisses *et al.*, "Developing and operating time critical applications in clouds: The state of the art and the switch approach," *Procedia Computer Science*, vol. 68, pp. 17–28, 2015.

[3] C. Ji, Y. Li, W. Qiu, U. Awada, and K. Li, "Big data processing in cloud computing environments," in *2012 12th International Symposium on Pervasive Systems, Algorithms and Networks*. IEEE, 2012, pp. 17–23.

[4] CHEF. [Online]. Available: https://www.chef.io/chef/

[5] M. Chen, S. Mao, and Y. Liu, "Big data: a survey," *Mobile Networks and Applications*, vol. 19, no. 2, pp. 171–209, 2014.

[6] A. Alamri, W. S. Ansari, M. M. Hassan, M. S. Hossain, A. Alelaiwi, and M. A. Hossain, "A survey on sensor-cloud: architecture, applications, and approaches," *International Journal of Distributed Sensor Networks*, vol. 2013, 2013.

[7] The eddy covariance fluxes of GHGs. [Online]. Available: https://wiki.envri.eu/display/EC/IC_13+The+eddy+covariance+fluxes+of+GHGs

[8] Z. Zhang, D. Li, and K. Wu, "Large-scale virtual machines provisioning in clouds: challenges and approaches," *Frontiers of Computer Science*, vol. 10, no. 1, pp. 2–18, 2016.

[9] H. Zhou, Y. Hu, J. Wang, P. Martin, J. Su, C. De Laat, and Z. Zhao, "Fast resource co-provisioning for time critical application based on networked infrastructure," in *IEEE International Conference on CLOUD (CLOUD)*. IEEE, 2016.

[10] ExoGENI. (2015). [Online]. Available: http://www.exogeni.net/

[11] EGI: European Grid Infrastructure. [Online]. Available: https://wiki.egi.eu/wiki/

[12] J. Jiang, R. Das, G. Ananthanarayanan, P. A. Chou, V. Padmanabhan, V. Sekar, E. Dominique, M. Goliszewski, D. Kukoleca, R. Vafin *et al.*, "Via: Improving internet telephony call quality using predictive relay selection," in *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*. ACM, 2016, pp. 286–299.

[13] O. Haq and F. R. Dogar, "Leveraging the power of cloud for reliable wide area communication," in *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*. ACM, 2015, p. 19.