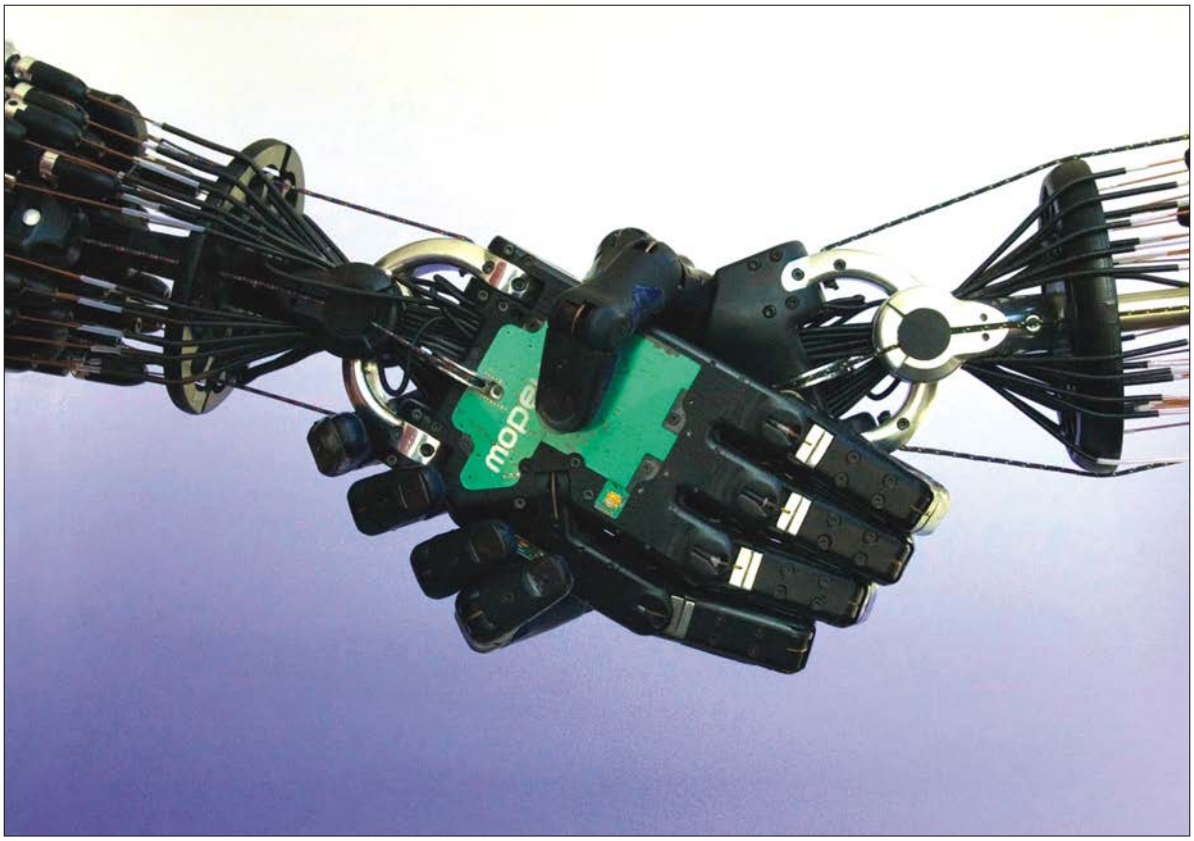

Journal of Artificial Intelligence

ISSN 1994-5450

Journal of Artificial Intelligence 5 (4): 161-169, 2012
ISSN 1994-5450 / DOI: 10.3923/jai.2012.161.169
© 2012 Asian Network for Scientific Information

A Dynamic Cloud Discovery Framework for Deploying of Scientific Computing Services over a Multi-cloud Infrastructure

[1]C.D. Karthic, [2]S. Sujatha and [2]V. Praveenkumar
[1]M.E-Pervasive Computing Technologies, Anna University of Technology, Tiruchirappalli Tamil Nadu, India
[2]Department of MCA, Anna University of Technology, Tiruchirappalli, Tamil Nadu, India

Corresponding Author: C.D. Karthic, M.E-Pervasive Computing Technologies, Anna University of Technology, Tiruchirappalli, Tamil Nadu, India

ABSTRACT
Traditional scientific computing systems relied on consecrated network of systems that arrives with complex maintenance and are deficient in flexibility and scalability. These issues necessitate immense level configuration process in result of a change in the setup (changing of resources, change of platform). Cloud computing, an on demand and cost effective means of provisioning computing resources, leverages all the above difficulties. The practical applications of clouds for scientific computing applications are still one of the largest unexplored areas in the cloud computing domain. Java Enterprise Platform (J2EE) has a successful journey in the field of distributed interoperable systems. This study explores the J2EE capabilities such as dynamic service discovery based on publishing of XML based configuration information of the service to the cloud. The Optimal Cloud Platform Selection (OCPS) algorithm evaluates the clouds for the optimal deployment of the scientific computing services based on various computational parameters like ram, processor cycles etc along with computational cost under consideration. This framework is flexible enough to cope with the dynamic nature of the clouds. The implementation has been tested in a hybrid cloud infrastructure using eucalyptus open source private cloud platform along with Google app-engine as a gateway to the outside world.

Key words: Java J2EE, dynamic cloud service discovery, optimized cloud platform selection, multi cloud infrastructure

INTRODUCTION
Scientific computing has been shifted to the cloud environments because of the added benefits of the cloud environment such as dynamic resource allocation, reduction in capital expenditure, increased flexibility and scalability (Kalin, 2009). Cloud environment simulates the notion of a pay as you go model of using the cloud resources. The client can assess the cloud facilities with minimal change in the infrastructure. The cloud allocates the extra resources to the clients in their computation process dynamically without any intervention to the computational task which is devastatingly possible in a traditional distributed computing environment. Scientific computing tasks require enormous computing resources that ass to the computation cost (Keahey and Freeman, 2008). The level of computation is a cloud is dogged by factors such as memory, processor cycles, data transfer rate and other latencies. Improper selection of the cloud results in unnecessary capital expenditure which is a critical asset of the organization or user. This study constructs a

framework for the optimal deployment of scientific computing services in a multi-cloud infrastructure. This framework utilizes the dynamic service discovery feature of web services to the cloud for discovering the suitable cloud environment. The selected cloud environments are analyzed by the OXPS algorithm based on various computing parameters along with cost for the optimal deployment of scientific computing services.

The ability to identify an apt cloud and its pricing scheme is a problematical task provided the varied cloud platforms and architectures. Inappropriate selection of clouds results in unexplained capital expenditures and execution delays. Porting the application between supported cloud platforms cannot be done without a major modification in configuration setup. Incompatible porting leads to inexplicable application behavior. Cloud computing are a distributed computing paradigm in which the abstracted computing resources are available across the network through virtualization (Kalin, 2009). Virtualization enables secure abstraction and sharing of innumerable instances the actual computing resources of a computer in software over the actual hardware of the provider. The client can execute the task in the abstracted server resources through methods such as SSH (Secure shell). Private cloud is implemented inside the private network of an organization to implement sensitive operations and full time cloud based research that is economically impossible in public cloud implementation. Eucalyptus, an open source Java based cloud platform provides an efficient choice to deploy a private Infrastructure as a Service (IaaS) cloud environment using the existing hardware infrastructure (Hoffa *et al.*, 2008). Eucalyptus provides support for major open source operating systems and utilizes virtualization concept to abstract the cloud instances using major hypervisors such as KVM and XEN. Eucalyptus can support all major operating systems as virtualized guests and is highly flexible such that the number of systems used for implementation can range from one to limitless based on the configuration type. Eucalyptus provides support for REST and SOAP based interfaces to the outside world for programmatic interaction and Amazon web services API for programmatic instance management. Eucalyptus provides a community cloud for users to implement a cloud environment after a simple registration process.

Efficient deployment of scientific computing applications in a multi cloud infrastructure is an area that remains unexplored. Traditional method of scientific computation has been performed in dedicated distributed systems that are narrower minded. Hence, we have concentrated on particular level of computation. The scenario to deploy a computing cluster on the top of a multi-cloud infrastructure, for solving loosely coupled Many-task Computing applications were explained (Moreno-Vozmediano *et al.*, 2011). Many task computing provides the high-performance computations comprising multiple distinct activities, coupled via file system operations (Walker *et al.*, 2006). The scientific jobs with a single uniform interface, using the feature-rich functionality found in these job management environments were studied (Walker *et al.*, 2006). The general issues in mapping applications and the functionality were described (Deelman *et al.*, 2005). The VM configurations based on user expectation were discussed (Keahey and Freeman, 2008). Scientific grid computing provides the scalability and high performance functionality in cloud (Chin *et al.*, 2005). Virtual computational domains that safely "trade" machines between them were described in VioCluster (Ruth *et al.*, 2005).

In-memory approaches to collective operations for parallel programming, builds on fast local file systems to provide high-speed local file caches for parallel scripts, uses a broadcast approach to handle distribution of common input data and uses efficient scatter/gather and caching techniques for input and output were detailed (Zhang *et al.*, 2008). Disk File System for Large Computing

Clusters delivers how the distributed locking and recovery techniques were extended to scale to large clusters (Schmuck and Haskin, 2002). The use of a cloud system as a raw computational on-demand resource for a grid middleware and Eucalyptus open-source cloud platform provides the guidance to work in cloud environment (Caron *et al*., 2009). This study aims at a simple existing model of distributed computing capabilities of the Java platform into the world of cloud computing. Even though we are able to implement a framework for porting an application on a PaaS cloud over an IaaS layer, there exist difficulties in porting the application at the IaaS level. The difficulties arise include proposing a format that is compatible among the machine images, able to transmit the bare application components at a wire level, able to reconstruct the contents from the wire level to the application level and making the reconstructed content interpretable by the new cloud environment.

PROPOSED FRAMEWORK

This study implements a framework that is the first of its kind in utilizing the dynamic service discovering capabilities of web services to the cloud platform for the efficient deployment of scientific computing services. The framework is implemented using two private eucalyptus clouds in our existing infrastructure using minimal configuration (2 systems per cloud). Google app engine is used as the cloud repository (analogous to UDDI in web services) that houses the details of the registered clouds. The client applications can discover the cloud capabilities dynamically using the cloud configuration files in the same manner as web services discover themselves.

The framework consists of the following modules:

- Master module
- Client module
- Updater module

Master module: Figure 1 depicts the module implemented as a web service that hosts the optimal cloud platform selection algorithm. The module determined the current cloud platform from the client request. This framework concentrates on the Java platform. Once the client requested platform is determined the module creates a rest/soap based query for the Google app engine cloud

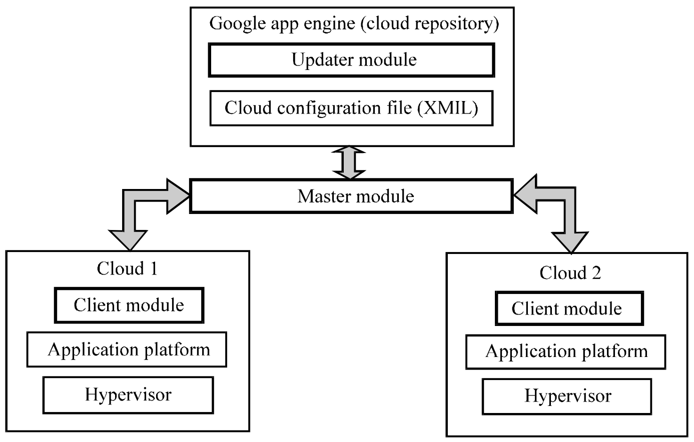

Fig. 1: Optimal cloud platform selection architecture

repository to select the list of cloud that supports the requested platforms. The Optimal Cloud Platform Selection (OCPS) algorithm determined the cloud for the optimal deployment of scientific computing services. The Optimal Cloud Platform Selection (OCPS) algorithm selects the optimal cloud platform based on comparison of the various computing parameters along such as ram, processor clock cycles along with computation cost. Once the optimal cloud platform is selected the corresponding cloud details is send to the client as the rest/soap response.

Mathematical model of OCPS algorithm: The mathematical model of optimal cloud platform selection algorithm aids in selecting the optimal cloud for deploying the scientific computing application. The optimal cloud is selected based on the expenditure of the computational cost (i.e.) the cost spend in performing the scientific computing in the clouds. The cost calculation is done as follows:

$$R = \frac{NPV}{NPC} \tag{1}$$

where, NPV is the net present value of the investment and NPC is the net present capacity of the cluster.

The NPV of an investment with annual amortized cash flow CT for Y years and cost of capital k, is computed as follows:

$$NPV = \sum_{T=0}^{Y-1} \frac{C_T}{(1+k)^T} \tag{2}$$

The NPC of a computing cluster in a cloud over an operational life span of Y years id computed as follows:

$$NPV = TC\left(\frac{1-\left(\frac{1}{\sqrt{2}}\right)^{y}}{1-\left(\frac{1}{\sqrt{2}}\right)}\right) \tag{3}$$

where, TC is the total capacity of the cluster, which is computed as follows:

$$TC = TCPU \times H \times \mu_r \tag{4}$$

where, TCPU is the total CPU cores in the computer cluster infrastructure, H is the estimated time of computation and μ_r is the expected cluster utilization.

Client module: The client module in Fig. 2 is implemented as a web service client in the private cloud that creates SOAP/REST based queries to the master module based on client requirements. The client requirements include the querying of available supported cloud platforms, deployment of a scientific computing service to the cloud.

The client action monitoring module monitors the cloud for configuration changes such as addition or removal of images, change of pricing schemes and various changes in hardware

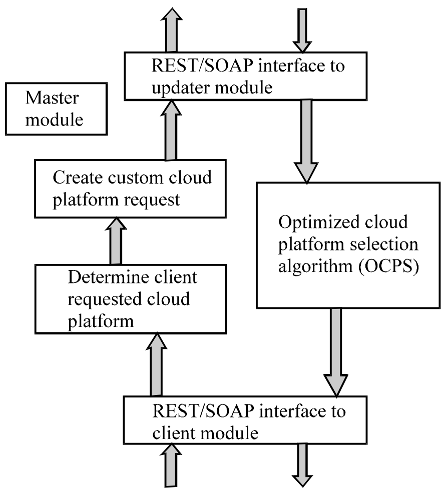
REST/SOAP interface to updater module
Master module
Create custom cloud platform request
Optimized cloud platform selection algorithm (OCPS)
Determine client requested cloud platform
REST/SOAP interface to client module

Fig. 2: Master module architecture

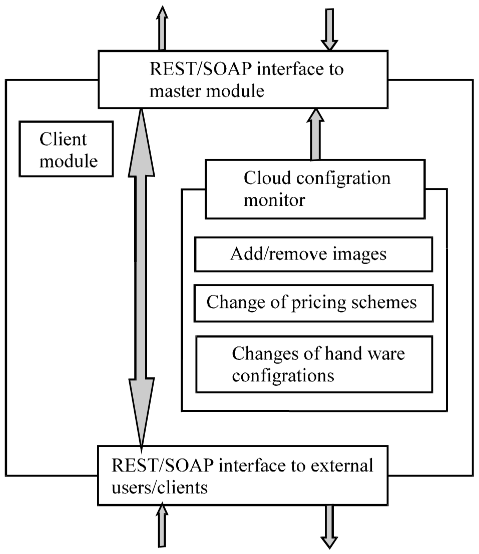
REST/SOAP interface to master module
Client module
Cloud configration monitor
Add/remove images
Change of pricing schemes
Changes of hand ware configrations
REST/SOAP interface to external users/clients

Fig. 3: Client module architecture

configurations. Once a change is detected, the updates are sent as REST/SOAP based queries to the master module automatically which then performs the corresponding action in the updater module of cloud repository.

Updater module: Figure 3 indicates the updater module implemented in the cloud repository. The updater module is a web based service hosted in the Google App Engine that serves the client applications. This framework makes Google App Engine, the cloud repository resembling UDDI in web services that houses XML based cloud configuration files. The configuration files resembles WSDL in web services that provides information about the cloud capabilities like type of cloud service, supported images and platforms, the pricing schemes, the URL to access the cloud and various other cloud details regarding the ownership information. The pricing scheme which determines the computation cost in clouds differs according to location, type of service, time of service (day/night). The updater module contains a request monitor that monitors each incoming and outgoing signals from the updater module. The requests made to the module are either for updating the XML based cloud configuration files or retrieving the cloud information according to the client specified platform. The updating of the cloud information configuration files are done by simple Java based XML API. XML language provides a simple user defined way to structure the data that is to the transferred.

Figure 4 indicates the object based storage and retrieval model of Google app engine data store. Google App Engine data store uses the concept of big tables to store the data rather than using traditional relational storage of data (RDBMS). In big table, each row is represented as an entity with a specific unique identifier across the entire App Engine platform, which can be programmatically altered at real time. Big tables ensure language based direct access to the data store rather than using a specific driver based access. In Driver access method each database uses a specific driver for a specific language, which can be used to access the database contents programmatically. Google App Engine's big table's concept frees the developer from maintaining the driver for accessing the data store contents as well as the freedom from learning SQL code for accessing the data from the database.

Figure 5 shows the XML based cloud configuration file that exposes the capabilities of the cloud. XML is a simplified format for data representation. Any complex data can be represented in a simplified model using XML and can be processed programmatically using simple XML based parsers. The cloud configuration provides a detailed representation of the cloud capabilities such as cloud type, availability, pricing scheme, location and various access parameters in a simplified XML format. The cloud configuration file also provides information such as location of cloud, URL to access the cloud and details of the provider of the cloud for accessing queries about cloud configuration and various other troubleshooting.

Figure 5 represented the XML based configuration file model. This file resembles the WSDL in web services. This file enables the client applications to learn about the capabilities of the cloud dynamically. This dynamic cloud capability detection enables the automation of the deployment of applications across supporting clouds. This relieves the user from manually finding the cloud

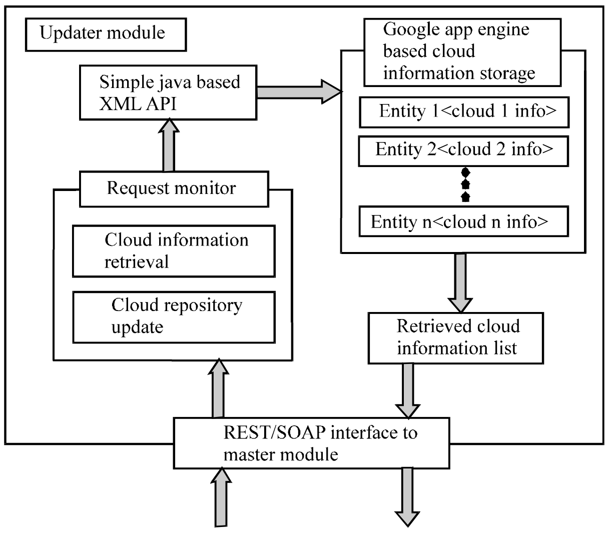
Updater module
Simple java based XML API
Google app engine based cloud information storage
Entity 1<cloud 1 info>
Entity 2<cloud 2 info>
Entity n<cloud n info>
Request monitor
Cloud information retrieval
Cloud repository update
Retrieved cloud information list
REST/SOAP interface to master module

Fig. 4: Updater module architecture

```
<?xml version ="encoding ="UTF-8"?>
<cloud-info>
 <cloud name ="eucalyptus" type =" IaaS">
<cloud-provider>encalyptus</cloud-provider>
<cloud-location>India</cloud-location>
<cloud-url>http://ind.eualyptus.com</cloud-url>
<images>
<os machine-id ="emi-123awrf">
<platform>Linux</platform>
<name>Fedora</name>
<version>16</version>
<architecture>x86</architecture>
```

Fig. 5: Cloud configuration file model

platform to deploy a particular computing application and discussing the computing cost etc. The Optimized Cloud Platform Selection (OCPS) algorithm enables the automatic selection of a cloud platform where the cost of computation does not exceed when compared to that of the computing resources provided by the platform. The client does not have to worry about the computation cost as well as the time needed to complete the computation. These cloud configuration files are easy to maintain because they are base on the simplified XML format and can be processed by a simple Java based XML parser.

The optimal cloud platform selection component is executed in the Google App Engine that provides a simple entity based programmatic data retrieval. Here, the emphasis to cloud selection is based on cost of computation which is a major asset in cloud based scientific computing process. The data store service of Google App Engine provides an excellent choice for programmatic data retrieval that is different from the traditional RDBMS (relational database management system) where a database specific driver is used to interact with the data stored. Google App Engine based data store provided significant performance improvement over the driver based data access in traditional RDBMS. The framework uses the cloud concept at the minimum configuration level because of economical constraints. The Google App Engine provides a free PaaS based cloud application hosting by limited resources allocation. Figure 6 show Code sample for Optimum Cloud Platform Selection.

EXTENSIONS TO THE ARCHITECTURE

This framework implements an optimal method for the deployment of scientific computing services in a multi-cloud environment using PaaS architecture. The experimental setup has been implemented using two eucalyptus private clouds which have been pre-initialized to resemble a PaaS platform. This framework concentrates on the Java based distributed computing capabilities. The same notion can be extended to various other

```
//cloud parameters

String title list() = {"imageId", "image Name", "category", "version", "ram", "storage", "numCpu", "cost", "architecture"};

Array list<String> least cost cloud = new Array list<String>0;


//identifer of cloud

Key key = Key factory.create key ("image", "image");

Datastore service datastore = Datastore service factory.get Datastore service 0;

Query query = new Query ("image", key);
```

Fig. 6: Code sample for optimum cloud platform selection

programming platforms like. NET, python Perl etc. The deployment of applications can be done in PaaS using minimal configuration changes. The configuration can be extended to advanced configurations which included hundreds of systems.

CONCLUSION AND FUTURE WORK

In this study, we have proposed a framework based on the existing distributed computing platform provided by Java to that of a cloud environment. This solution proposes a simple optimized solution to implement an efficient methodology for deploying a computing application over a multi-cloud infrastructure in a PaaS (Platform as a Service) cloud infrastructure. This can be extended to the IaaS (Infrastructure as a Service) platform where the porting of applications has been the real problem. The proposed work consists of the formulation of an interoperable format that is applicable between various cloud engines or at least be acceptable by various engines with minimal configuration changes. Also the work can be extended to include capabilities for on the fly dynamic change when being ported to a suitable cloud environment.

REFERENCES

Caron, E., F. Desprez, D. Loureiro and A. Muresan, 2009. Cloud computing resource management through a grid middle ware: A case study with diet and eucalyptus. Proceedings of the IEEE International Conference on Cloud Computing, September 21-25, 2009, Bangalore, pp: 151-154.

Chin, J., S. Harvey, S. Jha and P.V. Coveney, 2005. Scientific grid computing: The first generation. Comput. Sci. Eng., 7: 24-32.

Deelman, E., G. Singh, M. Su, J. Blythe and Y. Gil *et al*., 2005. *Pegasus*: A framework for mapping complex scientific workflows onto distributed systems. Sci. Program. J., 13: 219-237.

Hoffa, C., G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman and J. Good, 2008. On the use of cloud computing for scientific workflows. Proceedings of the IEEE Fourth International Conference on eScience, December 7-12, 2008, Indianapolis, IN., USA., pp: 640-645.

Kalin, M., 2009. Java Web Services Up and Running. O'Reilly Media, USA., ISBN-13: 9780596521127, Pages: 316.

Keahey, K. and T. Freeman, 2008. Science clouds: Early experiences in cloud computing for scientific applications. Proceedings of the Conference on Cloud Computing and Its Applications, October 22-23, 2008, Chicago, IL., USA.

Moreno-Vozmediano, R., R.S. Montero and I.M. Llorente, 2011. Multicloud deployment of computing clusters for loosely coupled MTC applications. IEEE Trans. Parallel Distrib. Syst., 22: 924-930.

Ruth, P., P. McGachey and D. Xu, 2005. VioCluster: Virtualization for dynamic computational domains. Proceedings of the IEEE International Conference on Cluster Computing, September 26-30, 2005, Burlington, MA., USA., pp: 1-10.

Schmuck, F. and K. Haskin, 2002. GPFS: A shared- disk file system for large computing clusters. Proceedings of the 1st Conference on File and Storage Technologies, January 28-30, 2002, Monterey, CL., USA.

Walker, E., J. Gardner, V. Litvin and E. Turner, 2006. Creating personal adaptive clusters for managing scientific jobs in a distributed computing environment. Proceedings IEEE International Workshop on Challenges of Large Applications in Distributed Environments, June 19, 2006, Paris, pp: 95-103.

Zhang, Z., A. Espinosa, K. Iskra, I. Raicu, I. Foster and M. Wilde, 2008. Design and evaluation of a collective I/O model for loosely-coupled petascale programming. Proceedings of the Workshop on Many-Task Computing on Grids and Supercomputers, November 17, 2008, Austin, TX., USA., pp: 1-10.