Investigating Techniques for Automating the Selection of Cloud Infrastructure Services'

MIRANDA ZHANG[1, 2], RAJIV RANJAN[1], DIMITRIOS GEORGAKOPOULOS[1], PETER STRAZDINS[2], SAMEE U. KHAN[3], ARMIN HALLER[1]

[1] Information Engineering Laboratory, CSIRO ICT Centre, Australia
[2] Research School of Computer Science, ANU, Australia
[3] Electrical & Computer Engineering Dept, North Dakota State University, USA

The Cloud infrastructure services landscape advances steadily leaving users in the agony of choice. As a result, Cloud service identification and discovery remains a hard problem due to different service descriptions, non-standardised naming conventions and heterogeneous types and features of Cloud services. In this paper, analysis the research challenges and present a Web Ontology Language (OWL) based ontology, the Cloud Computing Ontology (CoCoOn). It defines functional and non-functional concepts, attributes and relations of infrastructure services. We also present a system, CloudRecommender, that implements our domain ontology in a relational model. The system uses regular expressions and Structured Query Language (SQL) for matching user requests to service descriptions. We briefly describe the architecture of the CloudRecommender system, and demonstrate its effectiveness and scalability through a service configuration selection experiment based on a set of prominent Cloud providers' descriptions including Amazon, Azure, and GoGrid.

Keywords: Cloud Computing, Semantic Web, Recommender System, Service Computing, Operation Research

---

1. INTRODUCTION

1.1 Overview

The Cloud computing [Armbrust et al. 2010, Wang et al. 2010] paradigm is shifting computing from in-house managed hardware and software resources to virtualized Cloud-hosted services. Cloud computing assembles large networks of virtualized services: hardware resources (compute, storage, and network) and software resources (e.g., web server, databases, message queuing systems and monitoring systems.). Hardware and software resources form the basis for delivering Infrastructure as a Service (IaaS) and Platform as a Service (PaaS). The top layer focuses on application services (SaaS) by making use of services provided by the lower layers. In this paper, we focus on IaaS that is the underpinning layer on which the PaaS/SaaS services are hosted.

Cloud computing embraces an elastic paradigm where applications establish on-demand interactions with services to satisfy required Quality of Service (QoS) such as response time, throughput, availability and reliability. QoS targets are encoded in Legal Service Level Agreement (SLA) documents, which state the nature and scope of the QoS parameters. However, selecting and composing the right services meeting application requirements is a challenging problem. From a service discovery point of view, the selection process on the IaaS layer is based on a finite set of functional (e.g. CPU type,

memory size, location) and non-functional (costs, QoS, security) configuration properties that can be satisfied by multiple providers. Similarly, there is a service discovery problem associated with the SaaS and PaaS offerings. However, we are not considering these issues in this paper. IaaS providers [Wang et al. 2010, Wang et al. 2011] include Amazon Web Services (AWS)[1], Microsoft Azure[2], Rackspace[3], GoGrid[4] and others. They give users the option to deploy their application over a pool of virtually infinite services with practically no capital investment and with modest operating costs proportional to the actual use. Elasticity, cost benefits and abundance of resources motivate many organizations to migrate their enterprise applications (e.g. Content management system, Customer relationship management system and Enterprise resource planning system) to the Cloud. Although Cloud offers the opportunity to focus on revenue growth and innovation, decision makers (e.g., CIOs, scientists, developers, engineers, etc.) are faced with the complexity of choosing among private, public, and hybrid Cloud options and selecting the right service delivery and deployment model.

1.2 Motivation

With Cloud providers and service offerings having grown in numbers, the migration of applications (e.g. multi-layered enterprise application, scientific experiments, video-on-demand streaming application, etc.) to the Cloud demands selecting the best mix of services across multiple layers (e.g. IaaS and PaaS) from an abundance of possibilities. Any such Cloud service selection decision has to cater for a number of conflicting criteria, e.g. throughput and cost, while ensuring that QoS requirements are met. The problem is further aggravated by the fact that different applications have heterogeneous QoS requirements. For example, requirements for scientific [Wang et al. 2010] experiments (e.g., deadline) differ from video-on-demand streaming application (e.g., streaming latency, resolution, etc.).

Existing service selection methods have not kept pace with the rapid emergence of the multiple-layer nature of Cloud Services. Notably, techniques for web service selection cannot be adopted for Cloud Service Selection, because they do not cater for the diverse sets of criteria and their dependencies across multiple layers of Cloud Services. Although popular search engines (e.g., Google, Bing, etc) can point users to these provider web sites (blogs, wikis, etc.) that describe IaaS service [Wang et al. 2010] offerings, they are not designed to compare and reason about the relations among the different types of

[1] http://aws.amazon.com/
[2] http://www.windowsazure.com/en-us/
[3] http://www.rackspace.com/
[4] http://www.gogrid.com/

Cloud services and their configurations. Hence service description models and discovery mechanisms for determining the similarity among Cloud infrastructure services are needed to aid the user in the discovery and selection of the most cost effective infrastructure service meeting the user's functional and non-functional requirements.

In order to address these aforementioned problems, we present a semi-automated, extensible, and ontology-based approach to infrastructure service discovery and selection and its implementation in the CloudRecommender system. We identify and formalize the domain knowledge of multiple configurations of infrastructure services. The core idea is to formally capture the domain knowledge of services using semantic Web languages like the Resource Description Framework (RDF) and the OWL. The contributions of this paper are as the following:

1. Identification of the most important concepts and relations of functional and non-functional configuration parameters of infrastructure services and their definition in an ontology;
2. Modelling of service descriptions published by Cloud providers according to the developed ontology. By doing so, we validate the expressiveness of ontology against the most commonly available infrastructure services including Amazon, Microsoft Azure, GoGrid, etc.
3. An implementation of a design support system, CloudRecommender, based on our ontological model for the selection of infrastructure Cloud service configurations using transactional SQL semantics, procedures and views. The benefits to users of CloudRecommender include, for example, the ability to estimate costs, compute cost savings across multiple providers with possible tradeoffs and aid in the selection of Cloud services.

The rest of this paper is organized as follows. In section 2 we explain the research problem in more details. In section 3 we identify and formalize the domain ontology of multiple configurations of infrastructure services. In section 4 we present the implementation of the CloudRecommender system, and discuss the benefits of using a declarative logic-based language. In section 5 we illustrate the usage of CloudRecommender with a few case studies. In section 6 we compare our approach with some related works. In section 7 we conclude the paper and propose some future directions.

2. RESAERCH PROBLEMS

2.1 Automatic service identification and representation

Manually reading Cloud providers' documentation to find out which services are suitable for building their Cloud-based application is a cumbersome task for decision makers (e.g., a biologist intending to host his genomics experiment in the Cloud). The multi-layered organization (e.g., SaaS, PaaS, and IaaS) of Cloud Services, along with their heterogeneous types (Compute, Storage, Network, web server, databases, etc.) and features (Virtualization technology, billing model, Cloud location, cost, etc.) makes the task of service identification a hard problem. The use of non-standardized naming terminology makes this problem more challenging. For example, AWS refers to Compute services as Elastic Compute Cloud (EC2) Unit while GoGrid refers to the same as Cloud Servers. In addition, Cloud providers typically publish their service description, pricing policies and SLA rules on their websites in various formats. The relevant information may be updated without prior notice to the users. Furthermore, the structure of their web pages can change significantly leading to confusion. This leads to the following challenges: How to automatically fetch service description published by Cloud providers and present them to decision makers in a human readable way? Can we develop a unified and generic Cloud ontology to describe the services of any Cloud provider which exists now or may become available in the future?

2.2 Optimized Cloud Service Selection and Comparison

Consider an example of a medium scale enterprise that would like to move its enterprise applications to cloud. There are multiple providers in the current cloud landscape that offer infrastructure services in multiple heterogeneous configurations. Examples include, Amazon, Microsoft Azure, GoGrid, Rackspace, BitCloud, Ninefold, FelxiScale and TerreMark among many others. With multiple and heterogeneous options for infrastructure services, enterprises are facing a complex task when trying to select and compose a single service type or a combination of service types. Here we are concerned with simplifying the selection and comparison of a set of infrastructure service offerings for hosting the enterprise applications and corresponding dataset, while meeting multiple criteria, such as specific configuration and cost, emanating from the enterprise's QoS needs. This is a challenging problem for the enterprise and needs to be addressed.

Matching results to decision makers' requirements involves bundling of multiple related Cloud services, computing combined cost (under different billing models and discount offers), considering all possible (or only valuable) alternatives and multiple selection criteria (including specific features, long-term management issues and

versioning support). The diversity of offerings in the Cloud landscape leads to practical research questions: how well does a service of a Cloud provider perform compared to the other providers [Wang et al. 2011]? Which Cloud services are compatible to be combined or bundled together [Wada et al. 2011]? How to optimize the process of composite Cloud service selection and bundling? For example, how does a decision maker compare the cost/performance features of Compute, storage, and network service offered by various providers? Though branded calculators [Amazon Price Calculator 2013; Windows Azure Calculator 2013] are available from individual Cloud providers for calculating service leasing cost, it is not easy for decision makers to generalize their requirements to fit different service offered by different providers (with various quota and limitations) let alone comparing costs. Furthermore, a decision maker may choose one provider for storage intensive applications and another for computation intensive applications.

Despite the popularity of Cloud Computing, existing Cloud Service manipulations (e.g. select, start, stop, configure, delete, scale and de-scale) techniques require human familiarity with different Cloud service types and typically rely on procedural programming or scripting languages. The interaction with services is performed through low-level application programming interfaces (APIs) and command line interfaces. This is inadequate, given the proliferation of new providers offering services at different layers (e.g. SaaS, PaaS, and IaaS). One of the consequences of this state is that accessibility to Cloud Computing is limited to decision makers with IT expertise. This raises a set of research questions: How to develop interfaces that can transform low, system-level programming to easy-to-use drag and drop operations? Will such interfaces improve and simplify the process of Cloud Service Selection and Comparison (CSSC)?

3. THE DOMAIN ONTOLOGY

CoCoOn defines the domain model of the IaaS layer. This ontology facilitates the description of Cloud infrastructure services; and through mappings from provider descriptions, facilitates the discovery of infrastructure services based on their functionality and QoS parameters. The ontology is defined in the OWL [OWL2 2009] and can be found at: http://w3c.org.au/cocoon.owl. To describe specific aspects of Cloud computing, established domain classifications have been used as a guiding reference [Youseff et al. 2008]. For the layering of the ontology on top of Web service models, it builds upon standard semantic Web service ontologies i.e., OWL-S [Martin et al. 2004] and WSMO [Bruijn et al. 2005]. Consequently, modelers can use the grounding model and process model of OWL-S in combination with the presented Cloud computing

ontology to succinctly express common infrastructure Cloud services. We mapped the most prominent set of infrastructure services (i.e. Amazon, Azure, GoGrid, Rackspace, etc.) to CoCoOn. All common metadata fields in the ontology including Organization, Author, First Name etc. are referenced through standard Web Ontologies (i.e. FOAF[5] and Dublin Core[6]).

The Cloud computing ontology consists of two parts: functional Cloud service configurations information parameters; and non-functional service configuration parameters. In the following subsections, we detail on these two parts. We also present parts of the ontology in a visual form produced by the Cmap Ontology Editor tool [Eskridge et al. 2006].

3.1 Functional Cloud service configuration parameters

The main concept to describe functional Cloud service configurations in CoCoOn is a CloudResource that can be of one of the three types: Infrastructure-as-a-Service, Platform-as-a-Service or Software-as-a-Service (see Fig. 1). For the current implementation of the CloudRecommender system, we have defined the Cloud IaaS layer, providing concepts and relations that are fundamental to the other higher-level layers. In future work, we will extend the ontology to cover both PaaS and SaaS layers.

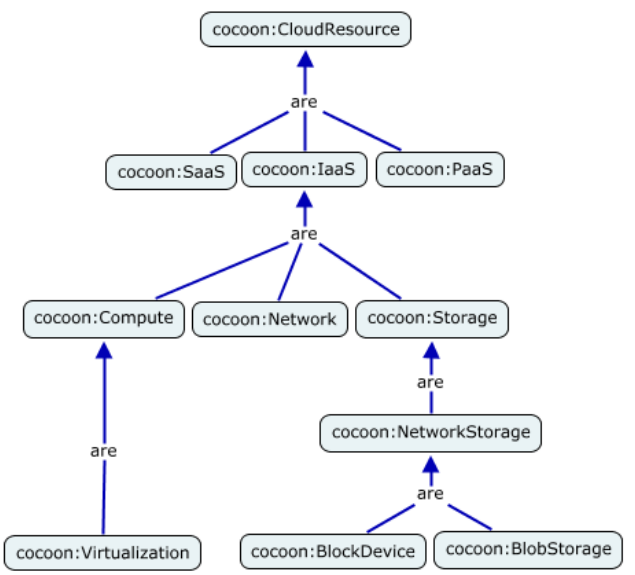


Figure 1: Top Concepts in the IaaS layer

[5] See http://xmlns.com/foaf/spec/

[6] See http://purl.org/dc/elements/1.1

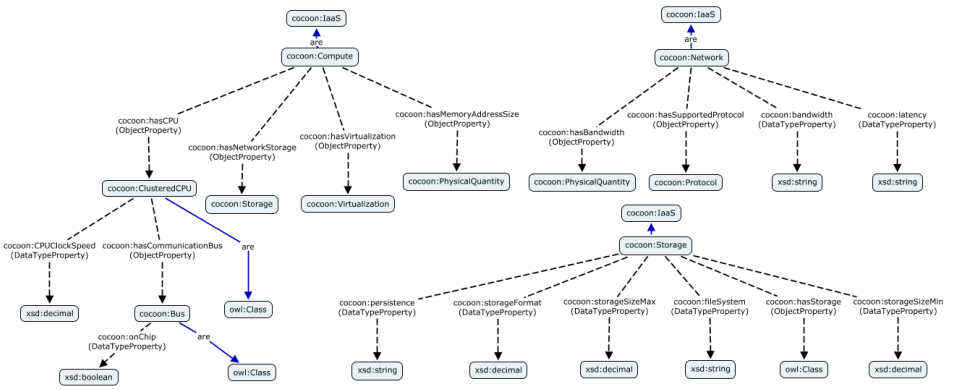


Figure 2: SubClasses and properties for the Compute, Storage and Network class

The Compute class (see Fig. 2) has the following object properties, hasVirtualization, hasCPU, hasMemoryAddressSize and hasNetworkStorage. The hasCPU property links a Compute unit to one or many processors which can be of type CPU or ClusteredCPU. A Compute object can be linked to a Storage object by using the top level object property hasStorage.

There are two different Storage types for a CloudResource: LocalStorage attached to a CPU with the hasLocalStorage property and NetworkStorage attached to a Compute instance with the hasNetworkStorage property. The hasNetworkStorage is an owl:inverseOf property of the isAttached property which can be used to define that a Storage resource is attached to a Compute resource. There is also an important distinction to be made between Storage resources that are attached to a Compute resource and Storage resources that can be attached. The latter is modeled with the isAttachable object property and its inverse property hasAttachable. These relations are important for the discovery of infrastructure services based on a user requirement. For example, in the case of Amazon, we can model that a BlockStorage with a StorageSizeMin of 1GB and a StorageSizeMax of 1TB can be attached to any EC2 Compute resource instance i.e., Standard, Micro, High-Memory, High-CPUCluster, ComputeCluster, GPUHigh-I/O. Consequently, if a user searches for a specific Compute instance with, for example, 5GB persistent storage, the relevant EC2 Compute resource and an Amazon BlockStorage will be returned (possibly among others). That is, because the isAttached relation in the user request can be matched with the definition of the Amazon EC2 unit with a BlockStorage defined to be isAttachable.

3.2 Non-Functional Cloud service configuration parameters

For non-functional Cloud service configuration parameters we distinguish between non-functional properties and QoS attributes. The first are properties of Cloud resources that

are known at design time, for example, PriceStorage, Provider, DeploymentModel, whereas QoS attributes can only be recorded after at least one execution cycle of a Cloud service, for example, avgDiskReadOperations, NetworkInLatency, NetworkOutLatency etc. For QoS attributes, we distinguish MeasurableAttributes like the ones above and UnmeasurableAttributes like Durability or Performance.

The QoS attributes define a taxonomy of Attributes and Metrics, i.e. two trees formed using the rdfs:subClassOf relation where a ConfigurationParameter, for example, PriceStorage, PriceCompute, PriceDataTransferIn (Out) etc. and a Metric, for example, ProbabilityOfFailureOnDemand, TransactionalThroughput, are used in combination to define non-functional properties (e.g. Performance, Cost, etc.). The resulting ontology is a (complex) directed graph where, for example, the Property hasMetric (and its inverse isMetricOf) is the basic link between ConfigurationParameters and Metric trees. For the QoS metrics, we used existing QoS ontologies [Dobson et al. 2005] as a reference whereas for the ConfigurationParameters concepts the ontology defines its independent taxonomy, but refer to external ontologies for existing definitions (e.g. QUDT[7]). Each configuration parameter (compare Table I) has a Name and a Metric (qualitative or quantitative). The Metric itself has a UnitOfMeasurement and a Value. The type of configuration determines the nature of a service by means of setting a minimum, maximum, or capacity limit, or meeting a certain value. For example, the hasMemory configuration parameter of a Compute service can be set to have a Value of 2 and a UnitOfMeasurement of GB.

Service	Configurations Parameters	Range/possible values
	Core	>= 1
	CPUClockSpeed	> 0
	hasMemory	> 0
	hasCapacity	>= 0
	Location	North America, South America, Africa, Europe, Asia, Australia
	CostPerPeriod	>= 0
	PeriodLength	> 0
	CostOverLimit	>= 0
Compute	PlanType	Pay As You Go, Prepaid
	StorageSizeMin	>= 0
	StorageSizeMax	> 0
	CostPerPeriod (e.g. Period = Month) (e.g. UnitOfMeasurement = GB)	>= 0
	Location	North America, South America, Africa, Europe, Asia, Australia
	RequestType	put, copy, post, list, get, delete, search
	CostPerRequest	>= 0
Storage	PlanType	Pay As You Go, Prepaid, Reduced Redundancy
	CostDataTransferIn	>= 0
Network	CostDataTransferOut	> 0

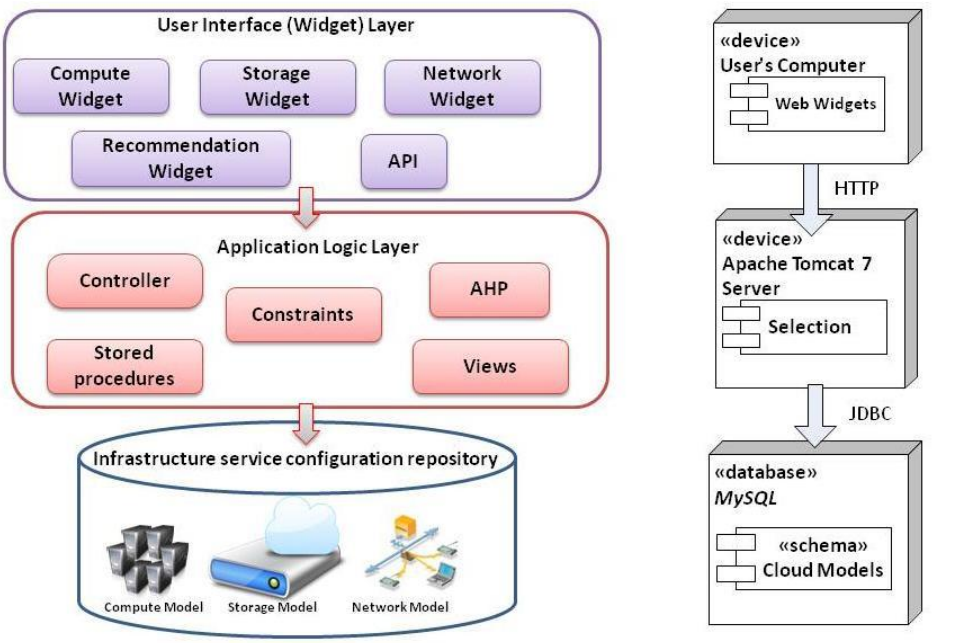
Table I. Infrastructure service types and their configurations

4. A SYSTEM FOR CLOUD SERVICE SELECTION

We propose an approach and a system for Cloud service configuration selection called CloudRecommender, shown in Fig. 3. For our CloudRecommender service, we

[7] See http://www.qudt.org

implemented the Cloud Service Ontology as a relational model and the Cloud QoS ontology as configuration information as structured data (entities) which we query using SQL. The choice of a relational model and SQL as query language was made because of the convenience SQL procedures offers us in regards to defining templates for a given widget type. We use stored procedures to create temporary tables and to concatenate parameters to dynamically generate queries based on the user input. As a future work, we will migrate the infrastructure services definitions to a Resource Description Framework (RDF) database and use, for example, SPIN templates[8] to encode our procedures in SPARQL[9].



(a) System architecture. (b) Deployment structure.

Figure 3: System architecture and deployment structure

We collected service configuration information from a number of public Cloud providers (e.g., Windows Azure, Amazon, GoGrid, RackSpace, Nirvanix, Ninefold, SoftLayer, AT and T Synaptic, Cloud Central, etc.) to demonstrate the generic nature of the domain model with respect to capturing heterogeneous configuration (see Table II) information of infrastructure services. The CloudRecommender system architecture consists of three layers: the configuration management layer, the application logic layer and the User interface (widget) layer. Details of each layer will be explained in the following sub-sections.

[8] http://www.w3.org/Submission/spin-overview/
[9] http://en.wikipedia.org/wiki/SPARQL

Provider	Compute Terminology	Pay As You Go Unit	Other Plans*	Storage Terminology	Pay As You Go Unit	Other Plans*	Trail Period or Value
Windows Azure	Virtual Server	/hr	1	Azure Storage	/GB month	1	90 day
Amazon	EC2 Instance	/hr	2	S3	/GB month	2	1 year
GoGrid	Cloud Servers	/RAM hr	1	Cloud Storage	/GB month		
RackSpace	Cloud Servers	/RAM hr		Cloud Files	/GB month		
Nirvanix				CSN	/GB month		
Ninefold	Virtual Server	/hr		Cloud Storage	/GB month	1	50 AUD
SoftLayer	Cloud Servers	/hr	1	Object Storage	/GB		
AT and T Synaptic	Compute as a Service	vCPU per hour + /RAM hr		Storage as a Service	/GB month		
Cloudcentral	Cloud Servers	/hr					
* Monthly/Quarterly/Yearly Plan, Reserve and Bidding Price Option							

Table II. Depiction of configuration heterogeneities in compute and storage services across providers. (Red) Blank cells in the table mean that a configuration parameter is not supported. Some providers offer their services under a different pricing scheme than pay-as-you-go. In Table II we refer to these schemes as other plans (e.g. Amazon Reduced redundancy, reserved price plans, GoGrid Pre-Paid plans). Table last updated October 2012.

4.1 Infrastructure service configuration repository

The system includes a repository of available infrastructure services from different providers including compute, storage and network services. These infrastructure services have very different configurations and pricing models. Ambiguous terminologies are often used to describe similar configurations, for instance different units of measurements are used for similar metrics. We performed unit conversions during instantiation of concepts to simplify the discovery process. For example, an Amazon EC2 Micro Instance has 613 MB of memory which is converted to approximately 0.599 GB. Another example is the CPU clock speed. Amazon refers to it as "ECUs". From their documentation [AmazonEC2 2012]: "One EC2 Compute Unit provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor. This is also the equivalent to an early-2006 1.7 GHz Xeon processor referenced in our original documentation".

Another example of disparity between different Cloud providers is the price model of "on Demand instances". GoGrid's plan, although having a similar concept to Amazon's On Demand and Reserved Instance, gives very little importance to what type or how many of compute services a user is deploying. GoGrid charges users based on what they call RAM hours – 1 GB RAM compute service deployed for 1 hour consumes 1 RAM Hour. A 2 GB RAM compute service deployed for 1 hour consumes 2 RAM Hour. It is worthwhile mentioning that only Azure clearly states that one month is considered to have 31 days. This is important as the key advantage of the fine grained pay-as-you-go price model which, for example, should charge a user the same when they use 2GB for half a month or 1 GB for a whole month. Other vendors merely give a GB-month price without clarifying how short term usage is handled. It is neither reflected in their usage calculator. We chose 31 days as default value in calculation.

Regarding storage services, providers charge for every operation that an application program or user undertakes. These operations are effected on storage services via RESTful APIs or Simple Object Access Protocol (SOAP) API. Cloud providers refer to the same set of operations with different names, for example Azure refers to storage service operations as transactions. Nevertheless, the operations are categorized into upload and download categories as shown in Table III. Red means an access fee is charged; green means the service is free; and yellow means access fees are not specified, and can usually be treated as green/free of charge. To facilitate our calculation of similar and equivalent requests across multiple providers, we analyzed and pre-processed the price data, recorded it in our domain model and used a homogenized value in the repository (configuration management layer). For example, Windows Azure Storage charges a flat price per transaction. It is considered as transaction whenever there is a "touch" operation, i.e. Create, Read, Update, Delete (CRUD) operation over the RESTful service interface, on any component (Blobs, Tables or Queues) of Windows Azure Storage.

Provider	Storage	Requests		
		Upload	Download	Other
Windows Azure	Azure Storage	storage transactions	storage transactions	
Amazon	S3	PUT, COPY, POST, or LIST Reques	GET and all other Requests	Delete
GoGrid	Cloud Storage	Transfer protocols such as SCP, SAMBA/CIFS, and RSYNC		
RackSpace	Cloud Files	PUT, POST, LIST Requests	HEAD, GET, DELETE Requests	
Nirvanix	CSN		Search	
Ninefold	Cloud Storage	GET, PUT, POST, COPY, LIST and all other transactions		
SoftLayer	Object Storage	Not Specified/Unknow		
AT and T Synaptic	Storage as a Service	Not Specified/Unknow		

Table III. Depiction of configuration heterogeneities in request types across storage services.

For providers that offer different regional prices, we store the location information in the price table. If multiple regions have the same price, we choose to combine them. In our current implementation, any changes to existing configurations (such as updating memory size, storage provision etc.) of services can be done by executing customized update SQL queries. We also use customized crawlers to update provider information's periodically. However, as a future work, we will provide a RESTful interface that can be used for automatic configuration updates.

4.2 Widget Layer

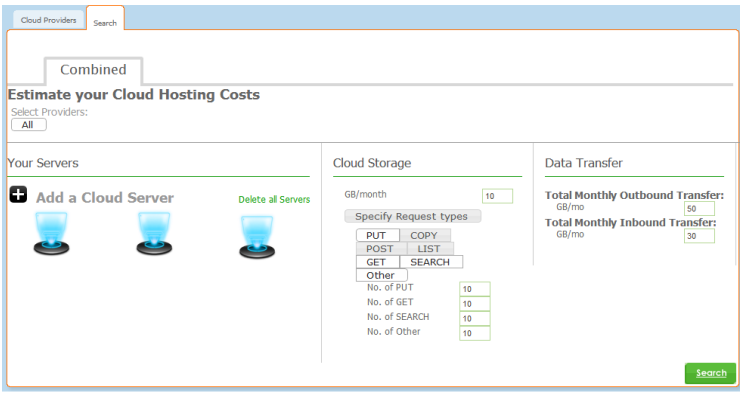


Figure 3: Screen shot of the widget.

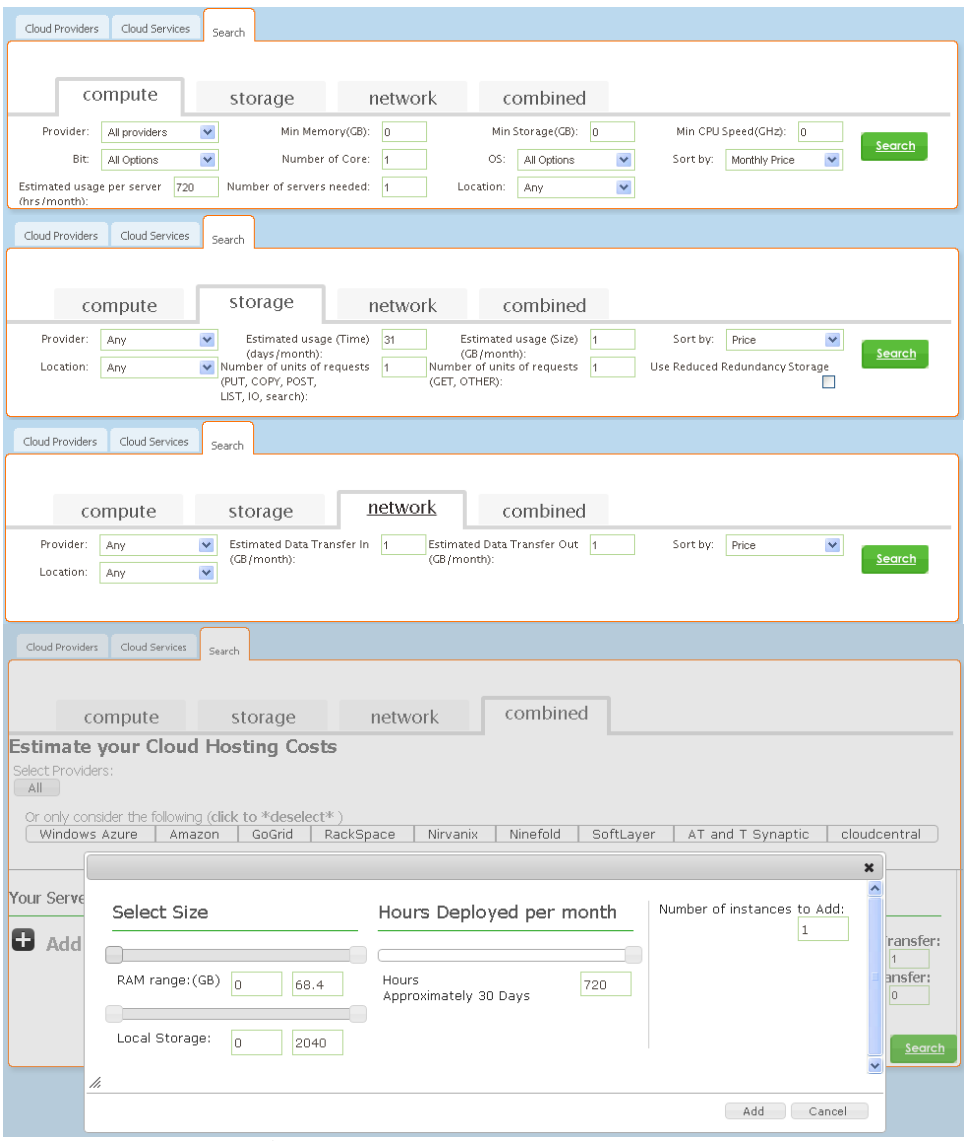


Figure 4: Screen shots of Compute, Storage, Network and the combined service selection widgets.

This layer features rich set of user-interfaces (see Fig 3 and Fig 4) that further simplify the selection of configuration parameters related to cloud services. This layer encapsulates the user interface components in the form of four principle widgets including: Compute, Storage, Network, and Recommendation. The selection of basic configuration parameters related to compute services including their RAM capacity, cores, and location can be facilitated through the Compute widget. It also allows users to search compute services by using regular expressions, sort by a specific column etc. Using the Compute widget, users can choose which columns to display and rearrange their order as well. The Storage widget allows users to define configuration parameters such as storage size and request types (e.g., get, put, post, copy, etc.). Service configuration parameters, such as the size of incoming data transfer and outgoing data transfer can be issued via the Network widget. Users have the option to select single service types as well as bundled (combined search) services driven by use cases. The selection results are displayed and can be browsed via the Recommendation widget (not shown in Fig 3).

5. CASE STUDIES

Gaia is a global space astrometry mission with a goal of making the largest, most precise three-dimensional map of our Galaxy by surveying more than one billion starts. For the amount of images produced by the satellite (1 billion stars x 80 observations x 10 readouts), if it took one millisecond to process one image, it would take 30 years of data processing time on a single processor. Luckily the data does not need to be processed continuously, every 6 months they need to process all the observations in as short a time as possible (typically two weeks) [AWS Case Study 2012]. Hypothetically speaking say they choose to use 120 high CPU and memory VMs. The example search via CloudRecommender is shown in Fig. 5. With each VM running 12 threads, there were 1440 processes working in parallel. This will reduce the processing time to less than 200 hours (about a week).

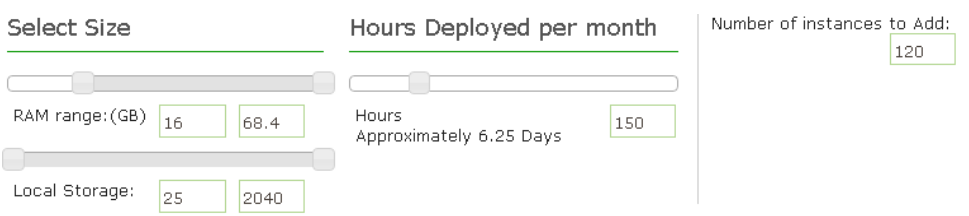
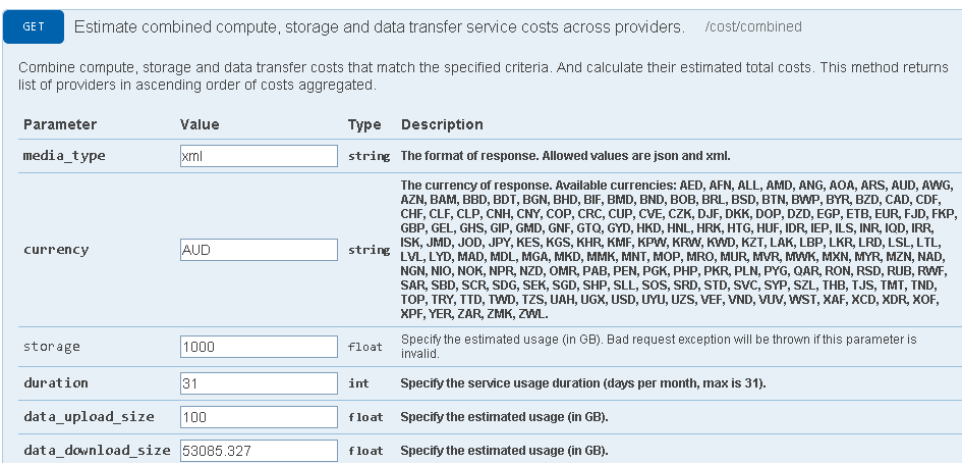


Figure 5: Example input parameter values.

In this case since data can be moved into/out of the cloud in bulk periodically, FedEx hard drive may be preferred over transferring data over the internet.

Promotional offers may not matter much in this case compare to the huge time and capital investment savings. But it makes a big difference for small business (or start ups) running a website.

Another example usage is sites with large continuous data input and processing need like Yelp. Everyday Yelp generates and stores around 100GB of logs and photos; runs approximately 200 MapReduce jobs and processing 3TB of data [Yelp 2012]. Yelp.com has more than 71 million monthly unique visitors [YelpInc 2012]. The average page size of a typical website is about 784 kB [Pingdom 2011]. So the estimated data download traffic is about 51TB per month if every unique user only views one page once a month. Fig. 6 shows a sample search for the above mentioned scenario.



GET Estimate combined compute, storage and data transfer service costs across providers. /cost/combined

Combine compute, storage and data transfer costs that match the specified criteria. And calculate their estimated total costs. This method returns list of providers in ascending order of costs aggregated.

Parameter	Value	Type	Description
media_type	xml	string	The format of response. Allowed values are json and xml.
currency	AUD	string	The currency of response. Available currencies: AED, AFN, ALL, AMD, ANG, AOA, ARS, AUD, AWG, AZN, BAM, BBD, BDT, BGN, BHD, BIF, BMD, BND, BOB, BRL, BSD, BTN, BWP, BYR, BZD, CAD, CDF, CHF, CLF, CLP, CNH, CNY, COP, CRC, CUP, CVE, CZK, DJF, DKK, DOP, DZD, EGP, ETB, EUR, FJD, FKP, GBP, GEL, GHS, GIP, GMD, GNF, GTQ, GYD, HKD, HNL, HRK, HTG, HUF, IDR, IEP, ILS, INR, IQD, IRR, ISK, JMD, JOD, JPY, KES, KGS, KHR, KMF, KPW, KRW, KWD, KZT, LAK, LBP, LKR, LRD, LSL, LTL, LVL, LYD, MAD, MDL, MGA, MKD, MMK, MNT, MOP, MRO, MUR, MVR, MWK, MXN, MYR, MZN, NAD, NGN, NIO, NOK, NPR, NZD, OMR, PAB, PEN, PGK, PHP, PKR, PLN, PYG, QAR, RON, RSD, RUB, RWF, SAR, SBD, SCR, SDG, SEK, SGD, SHP, SLL, SOS, SRD, STD, SVC, SYP, SZL, THB, TJS, TMT, TND, TOP, TRY, TTD, TWD, TZS, UAH, UGX, USD, UYU, UZS, VEF, VND, VUV, WST, XAF, XCD, XDR, XOF, XPF, YER, ZAR, ZMK, ZWL.
storage	1000	float	Specify the estimated usage (in GB). Bad request exception will be thrown if this parameter is invalid.
duration	31	int	Specify the service usage duration (days per month, max is 31).
data_upload_size	100	float	Specify the estimated usage (in GB).
data_download_size	53085.327	float	Specify the estimated usage (in GB).

Figure 6: Example parameters for REST API

6. RELATED WORK

In relation to research problem 1 (see section 2.1), there are 3 common approaches for web services identification/publication: 1) manually maintain directories by categorizing manually-submitted or collected information about Cloud services and providers, an example of such kind is Universal Description, Discovery and Integration (UDDI), which has failed to gain wide adoption; 2) use web crawling, and automatically create listings; and 3) combine both, e.g. using manually-submitted URIs as seeds to generate indexes. The first approach is the only feasible solution at the moment. But extensive research and standardization efforts have been put into developing web information representation models, namely, RDF, the semantic web, and ontologies [Ozsoyoglu et al. 2003]. Some

of the recent research such as [Ruiz-Alvarez et al. 2011] has focused on Cloud storage service (IaaS level) representation using XML. But the proposed schema does not comply with or take into account any of the above mentioned standards. We believe that semantic web technologies should be adopted to standardize the Cloud services representations.

For research problem 2 (see section 2.2), a number of research [Li et al. 2010] and commercial projects (mostly in their early stages) provide simple cost calculation or benchmarking and status monitoring, but none is capable to consolidate all aspects and provide a comprehensive ranking of infrastructure services. For instance, CloudHarmony provides up-to-date benchmark results without considering cost, Cloudorado calculates the price of IaaS-level compute services based on static features (e.g., processor type, processor speed, I/O capacity, etc.) while ignoring dynamic QoS features (e.g., latency, throughput, load, utilization, etc.). Yuruware[10] Compare beta version offers elementary search on Compute IaaS Cloud services. Although they aim to provide an integrated tool with monitoring and deploying capabilities, it is still under development and the finish date is unknown. The current version does not allow selection of storage service by itself and QoS features have not been compared. Prior to CloudRecommender, there have been a variety of systems that use declarative logic-based techniques for managing resources in distributed computing systems. The focus of the authors in work [Liu et al. 2011] is to provide a distributed platform that enables Cloud providers to automate the process of service orchestration via the use of declarative policy languages. The authors in [Brodsky et al. 2009] present an SQL-based decision query language for providing a high-level abstraction for expressing decision guidance problems in an intuitive manner so that database programmers can use mathematical programming technique without prior experience. We draw a lot of inspiration from the work in [Mao et al. 2011] which proposes a data-centric (declarative) framework to improve SLA fulfillment ability of Cloud service providers by dynamically relocating infrastructure services. COOLDAID [Chen et al. 2010] presents a declarative approach to manage configuration of network devices and adopts a relational data model and Datalog-style query language. NetDB [Caldwell et al. 2004] uses a relational database to manage the configurations of network devices. However, NetDB is a data warehouse, not designed for Cloud service selection or comparison. In contrast to the aforementioned approaches, CloudRecommender is designed for solving the new challenge of handling heterogeneous service configuration and naming conventions in Cloud computing. It is designed with a different application

[10] http://www.yuruware.com/

domain – one that aims to apply declarative and widget programming technique for solving the Cloud service selection problem.

7. CONCLUSION AND FUTURE WORK

We have proposed ontology for classifying and representing the configuration information related to Cloud-based IaaS services including compute, storage, and network. The proposed ontology is comprehensive as it can not only capture static confiugration but also dynamic QoS configuration on the IaaS layer. We also presented the implementation of the ontology in the CloudRecommender system. The paper will help readers in clearly understanding the core IaaS-level Cloud computing concepts and inter-relationship between different service types. This in turn may lead to a harmonization of research efforts and more inter-operable Cloud technologies and services at the IaaS layer.

In future work, we intend to extend the ontology with the capability to store PaaS and SaaS configurations. Moreover, we would also like to extend our ontology to capture the dependency of services across the layers. For example, investigating concepts and relationships for identifying the dependencies between compute service (IaaS) configurations and the type of appliances (PaaS) that can be deployed over it. Before mapping a MySQL database appliance (PaaS) to a Amazon EC2 compute service (IaaS), one needs to consider whether they are compatible in terms of virtualization format. Another avenue that we would like to explore is how to aggregate QoS configurations across the IaaS, PaaS, and SaaS layers for different application deployment scenarios (e.g., multimedia, eResearch, and enterprise applications). Notably, QoS aware service selection problem [Jaeger et al. 2005] is a multi-criteria optimization problem, in order to solve it, two distinct techniques will be explored: i) evolutionary optimization techniques, the process of simultaneously optimizing two or more conflicting objectives expressed in the form of linear or non-linear functions of criteria; ii) Multi-criteria decision-making techniques, including Analytic Hierarchy Process (AHP) and others can handle mixed qualitative and quantitative criteria.

REFERENCES

AmazonEC2 2012. http://aws.amazon.com/ec2/instance-types/

AWS Case Study 2012, The Server Labs. Available: http://aws.amazon.com/solutions/case-studies/the-server-labs/

ARMBRUST, M., FOX, A., GRIFFITH, R., JOSEPH, A. D., KATZ, R., KONWINSKI, A., LEE, G., PATTERSON, D., RABKIN, A., STOICA, I., AND ZAHARIA, M. 2010. A view of Cloud Computing, Communications of the ACM Magazine, Vol. 53, No. 4, ACM Press, 50-58.

Amazon Price Calculator 2013. http://calculator.s3.amazonaws.com/calc5.html.

BRODSKY, A., BHOT, M. M., CHANDRASHEKAR, M., EGGE, N. E., AND WANG, X. S. 2009. A Decisions Query Language (DQL): High-level Abstraction for Mathematical Programming over Databases. In Proceedings of the 35th SIGMOD international conference on Management of data (SIGMOD '09), RI, USA, June 29 - July 02, 2009, C. BINNIG AND B. DAGEVILLE, Eds. ACM Press, New York, NY, USA.

BRUIJN, J.D., BUSSLER, C., DOMINGUE, J., FENSEL, D., HEPP, M., KELLER, U., KIFER, M., KONIG-RIES, B., KOPECHY, J., LARA, R., LAUSEN, H., OREN, E., POLLERES, A., ROMAN, D., SCICLUNA, J., AND STOLLBERG, M. 2005. Web service modeling ontology (WSMO), W3C, Tech. Report.

CALDWELL, D., GILBERT, A., GOTTLIEB, J., GREENBERG, A., HJALMTYSSON, G., AND REXFORD, J. 2004.The cutting EDGE of IP router configuration. SIGCOMM Comput. Commun. Rev.34, 1 (January 2004), 21-26.

CHEN, X., MAO, Y., MAO, Z. M., AND MERWE, J. V. D. 2010. Declarative Configuration Management for Complex and Dynamic Networks. In Proceedings of the 6th ACM International Conference on emerging Networking Experiments and Technologies (CoNEXT), 10 pages, Philadelphia, USA, ACM Press.

DOBSON, G., LOCK, R., AND SOMMERVILLE, I. 2005. QoSOnt: a QoS ontology for service-centric systems, In *Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications*, Porto, Portugal, 30 August - 3 September 2005, 80-87.

ESKRIDGE, T., HAYES, P., AND HOFFMAN, R. 2006. Formalizing the informal: A Confluence of Concept Mapping and the Semantic Web. In Proceedings of the Second Int. Conference on Concept Mapping, San José, Costa Rica, 2006, A. J. CANAS AND J. D. NOVAK, Eds. 247-254.

GENS, F. 2010. IDC's Public IT Cloud Services Forecast: New Numbers, Same Disruptive Story. Available: http://blogs.idc.com/ie/?p=922

JAEGER, M. C., MUHL, G., AND GOLZE, S. 2005. QoS-aware composition of Web services: a look at selection algorithms. In Proceedings of IEEE International Conference on Web Services (ICWS'05), IEEE Computer Society, Orlando, Florida, USA, July 11-15, 2005, 800-808.

LI, A., YANG, X., KANDULA, S., AND ZHANG, M. 2010. CloudCmp: comparing public cloud providers. In Proceedings of the 10th ACM SIGCOMM conference on Internet measurement(IMC '10), Melbourne, Australia, November 01 - 03, 2010, ACM, New York, USA, 1-14.

LIU, C., LOO, B. T., AND MAO, Y. 2011. Declarative automated cloud resource orchestration. In Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC '11). ACM, New York, NY, USA, Article 26, 8 pages.

MARTIN, D.L., PAOLUCCI, M., MCILRAITH, S.A., BURSTEIN, M.H., MADERMOTT, D.V., MCGUINNESS, D.L., PARSIA, B., PAYNE, T.R., SABOU, M., SOLANKI, M., SRINIVASAN, N., AND SYCARA, K.P. 2004. Bringing Semantics to Web Services: The OWL-S Approach. In Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition, San Diego, CA, USA, July 6, 2004, J. CARDOSO AND A. P. SHETH, Eds. Springer, 26-42.

MAO, Y., LIU, C., MERWE, J.E.V.D., AND FERNANDEZ, M. 2011. Cloud Resource Orchestration: A Data-Centric Approach. In Proceedings of the 5th biennial Conference on Innovative Data Systems Research (CIDR), Asilomar, California, USA, January 9-12, 2011, 241-248.

OWL2 2009. Web Ontology Language: Document Overview. W3C Recommendation, http://www.w3.org/TR/owl2-overview/.

OZSOYOGLU, G., AND AL-HAMDANI, A. 2003. Web Information Resource Discovery: Past, Present, and Future. In Proceedings of the 18th International Symposium on Computer and Information Sciences, Antalya, Turkey, November 2003, A. YAZICI AND C. SENER, Eds. Springer, Berlin, Heidelberg, 9-18.

Pingdom 2011. *Web pages are getting more bloated, and here's why*. Available: http://royal.pingdom.com/2011/11/21/web-pages-getting-bloated-here-is-why/

RUIZ-ALVAREZ, A., AND HUMPHREY, M. 2011. An Automated Approach to Cloud Storage Service Selection. In Proceedings of the 2nd international workshop on Scientific cloud computing (ScienceCloud '11), San Jose, California, USA, June 08 - 11, 2011, ACM Press, New York, USA, 39-48.

Windows Azure Calculator 2013. http://www.windowsazure.com/en-us/pricing/calculator/.

WADA, H., SUZUKI, J., YAMANO, Y., AND OBA, K. 2011. Evolutionary Deployment Optimization for Service Oriented Clouds. *Software: Practice and Experience 4, 469 – 493.*

WANG, L., RANJAN, R., CHEN, J., AND BENATALLAH, B. (editors) 2011. *Cloud Computing: Methodology, Systems, and Applications*. Taylor and Francis Group , London, UK.

WANG, L., LASZEWSKI, G., YOUNGE, A., He, X., KUNZE, M., TAO, J., AND FU, C., 2010, Cloud Computing: a Perspective Study. New Generation Comput. 28(2): 137-146

WANG L., KUNZE, M., TAO, J., AND LASZEWSKI, G., 2011, Towards building a cloud for scientific applications. Advances in Engineering Software 42(9): 714-722.

WANG, L., AND FU, C., 2010, Research Advances in Modern Cyberinfrastructure. New Generation Comput. 28(2): 111-112.

WANG, L., LASZWSKI, G., CHEN, D., TAO, J., KUNZE, M., 2010, Provide Virtual Machine Information for Grid Computing. IEEE Transactions on Systems, Man, and Cybernetics, Part A 40(6): 1362-1374.

WANG, L., Chen, D., HUANG, F.,, 2011, Virtual workflow system for distributed collaborative scientific applications on Grids. Computers & Electrical Engineering 37(3): 300-310.

Yelp 2012. AWS Case Study, http://aws.amazon.com/solutions/case-studies/yelp/

YelpInc 2012. Wikipedia. Available: http://en.wikipedia.org/wiki/Yelp,_Inc
YOUSEFF, L., BUTRICO, M., AND SILVA, D. D. 2008. Toward a Unified Ontology of Cloud Computing. In Grid Computing Environments Workshop, Austin, TX, USA, Nov 2008, GCE '08. IEEE Computer Society, Washington, DC, USA, 1-10.