AN ENHANCED APPROACH TO SUPPORT MULTIPLE DATA STORES APLLICATIONS USING ODBAPI IN CLOUD ENVIRONMENTS

Gouri S Joshi, Gayatri Bhandari

[1] *P.G. Student, Savitribai Phule Pune University, Department of Computer Engineering, BSIOTR, Wagholi, Pune, Maharashtra, India*

[2] *Professor, Savitribai Phule Pune University, Department of Computer Engineering, BSIOTR, Wagholi, Pune, Maharashtra, India*

ABSTRACT

In period of time generation of enormous quantity data and also rise of cloud computing have introduced new aspects for data management. Many applications got to move with many heterogeneous data stores betting on the sort of data they need to manage: ancient data types, documents, simple key-value data, graph data, etc. Interacting with heterogeneous data models via different APIs, multiple data store applications imposes difficult tasks to their developers. Indeed, programmers got to be conversant in completely different APIs. Additionally, developers got to master and handle the advanced processes of cloud discovery, and deployment of application and execution. Projected system represents a declarative approach using ODBAPI sanctioning to lighten the burden of the complex and non-standard tasks of discovering relevant cloud surroundings and deploying applications on them whereas letting developers to easily concentrate on specifying their storage and computing needs.

Keyword : - *ODBAPI, data stores, rational data stores, NoSQL.*

1. INTRODUCTION

Cloud computing has recently risen as a new computing paradigm which empower ondemand and it has scalable provision of resources. It also has platforms and software as services. Cloud computing is divided into three levels[2] : 1. Infrastructure as a Service (IaaS) provides access to the abstracted view on the hardware, 2. the Platform-as-a-Service (PaaS) supplies programming and execution environments to the developers, and 3. the Software as a Service (SaaS) allowing software applications to be utilized by clouds end users. Cloud computing adds execution environments for some emerging applications like big data management due to its elasticity property. The variety property of big data is mainly focused and more precisely on multi data store based applications in the cloud.

To satisfy variety of storage requirements, cloud applications needs to access and interact with several relational and NoSQL data stores which has heterogeneous APIs of the data stores which induces problems while constructing, deploying and migrating multi data store applications. Main four troubles are: 1. Elephantine workload on the developer: These days data stores have heterogeneous and variety of APIs. Developers of multi data store based

applications need to be known all these APIs while coding their applications. 2. No declarative way to execute complex queries: Heterogeneity of the data models has no declarative way to define and execute complex queries over multiple data stores. This is because of the absence of global schema of heterogeneous data stores. NoSQL data stores does not have specific schema. It means developers should have to manage with the implementation of such complex queries. 3. Code acclimation: Application developers need to re-adapt the application source code to deal with new data stores when applications are migrating from one cloud environment to another. Developers should have potential to learn and use new APIs. 4. Tedious and nonstandard processes of discovery of cloud and deployment: Once an application is build or migrated, developers need to spread it into cloud provider. Discovering the most appropriate cloud environment which can provide
data stores requirements and deploying the application on it are tedious and meticulous
provider-specific process.

To overcome these problems Proposed system represents a clear approach for discovering proper cloud environments using ODBAPI[4] and redistributing applications on them in the time letting developers simply focuses on specifying their storage and computing requirements[1] .

2. LITERATURE SURVEY

The Spring Data Framework provides some generic ab-stractions to handle different types of relational and NoSQL DBMSs. Nevertheless, new data store addition isn't really easy and therefore the solution is powerfully coupled to the Java programming model[5].

SOS provides CRUD operations at the level of individual data store. These operations are provided via GET, PUT, and DELETE methods.SOS is extended to integrate relational data store; in the meantime, there's no proof for potency and extensibility of their system.

NoSQL have recently emerged and have generated each criticism and interest. Interest as a result of the address requirements that are noticeably necessary in large-scale applications, criticism owing to the comparison with standard relational achievements. Heterogeneity of the languages is the major problem usually mentioned and the interfaces they provide to developers and users.Several languages and platforms are proposed, and applications developed for one system need vital effort to be migrated to a different one. Here a common programming interface to NoSQL systems called SOS (Save Our Systems) is proposed. Its goal is to support application development by concealing the particular details of the assorted systems. it's supported a metamodeling approach, within the sense that the particular interfaces of the individual systems are mapped to a standard one. The tool provides ability as well, since one application will interact with many systems at constant time[6]

An application to barter its data Management Contract (DMC), usually referred as data agreement or data license, with numerous clouds and to bind to the particular DBMSs according to its DMC. Truong et al. [7] propose to specify and model data considerations in data contracts to support concern-aware data utilization and selection.Currently, rich and diverse data types are progressively provided using the data-as-a-service (DaaS) model, a kind of cloud computing services and the core element of data marketplaces. This facilitates the on-the-fly data composition and usage for many dataintensive applications in e-science and business domains. However, knowledge offered by DaaS are constrained by various data considerations that, if not automatically being reasoned properly, will lead to a wrong way of using them. Here the view that data considerations ought to be expressly modelled and specified in data contracts to support concern-aware data selection and utilisation. A close analysis of current techniques for knowledge contracts within the cloud is completed. rather than looking forward to a selected illustration of data contracts, an abstract model for data contracts that may be used to build differing kinds of data contracts for specific types of data is introduced. supported the abstract model, many techniques for evaluating data contracts that may be integrated into data service choice and composition frameworks.

Cloud computing primarily based Data-asa-Service (DaaS) has become common. Many data assets are released in DaaSes across completely different cloud platforms. however, there are no well-defined ways tha to clarify DaaSes and their associated data assets. On the one hand, existing DaaS suppliers usually use HTML documents to explain their service. This simple means of service description needs user to manually present service lookup by reading the HTML documents to know DaaSes as well as their provided data assets. On the other hand, existing service description techniques aren't appropriate for describing DaaSes because they take into account only service info.

The lack of wellstructured model to explain DaaSes hinders the automatic service search for DaaSes and the combination of DaaSes in data composition and analytic tools. In this paper, they propose DEMODS, an outline MOdel for DaaS, that introduces a general linked model to cover all basic info of a DaaS. Besides the fundamental DaaS description model, they tend to additionally introduce an extended model that combines existing work in elaborate data quality, data and service contract, data dependency, and Quality of Service (QoS). A mechanism to include DEMODS into each new and existing DaaSes is introduced. Finally, a prototype of DEMODS is developed to examine the effectiveness of the proposed model.However, Truong et al[8]. propose this data contract for data and to not store data or to assist the developer to decide on the suitable data stores for his application.

3. IMPLEMENTATION DETAILS

3.1 System Architecture

System architecture of proposed systemis as shown in Fig.1.

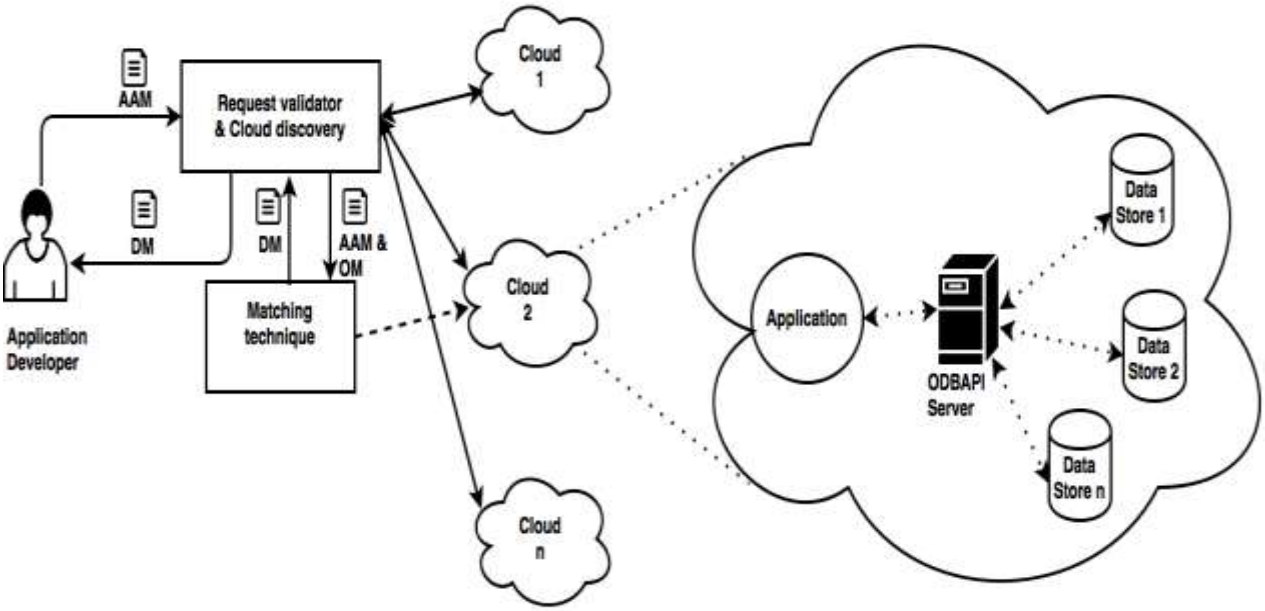

Fig -1: System Architecture

Application Developer: Application developer sends Abstract Application Manifest(AAM) JSON to request validator and cloud discovery for getting cloud information. This request contains all cloud details, application requirement and DB requirement.

Request Validator and Cloud Discovery: Upon receiving request from application developer this module first validates the request and after successful validation sends request to cloud matching technique module for selecting best match within list of available cloud. It sends offer manifest(OM) JSON to cloud matching technique with list of cloud requirement and application requirement.

Matching Technique: This technique reads list of cloud requirement from offer manifest JSON and selects best suiting cloud from list of available cloud. After selecting the cloud this module deploy the application on that cloud and return address information of cloud to query data store technique. On receiving cloud information from cloud matching technique, request validator and cloud discovery modules returns success response to application user along with address of cloud through deployment manifest(DM).

Abstract application manifest: This manifest contains 2 classes of requirements.

First, needs in term of data stores. The developer provides 5 info regarding the specified data stores such as type, name, version, size and the query type to execute. It's worth noting that once the developer fills this manifest, he has the freedom to specify one or multiple information. For each info, he offers a constraint expressed by a constant

value, a joker (denoting any values) or certain conditions (stated as inequalities). Hence, a lot of flexibility within the model is ensured. Fig.2. depicts structure of abstact application manifest.

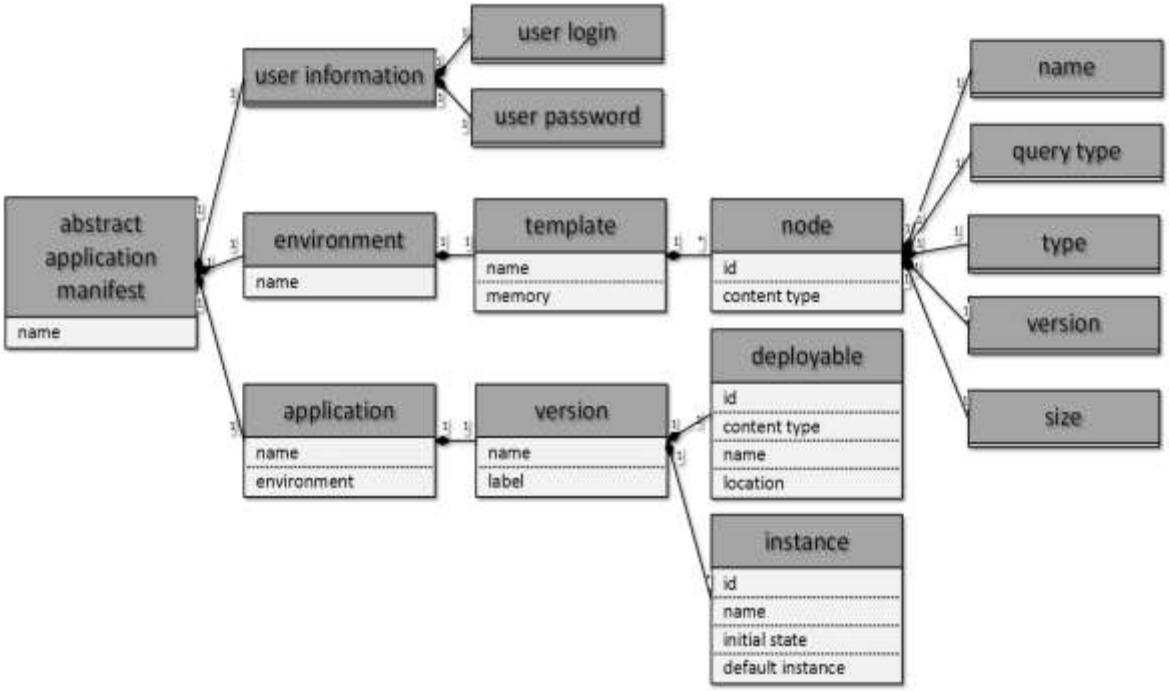

Fig -2: Abstract Application Manifest

.

Offer Manifest: The offer manifest contains info concerning the capabilities of data stores of every discovered cloud provider. Indeed,The offer manifest contains info concerning the capabilities of data stores of every discovered cloud provider. Indeed, the root class is offer manifest and it's known by the name attribute containing one or multiple cloud provider class which represents a discovered cloud provider which is identified by a single id attribute. It contents the name class describing the cloud provider name and also the environment response class presenting the abilities that a cloud supplier exposes according to an AAM. The environment response class consists by one or multiple offer class that contains one or many node class. This class identical to the node element in the AAM modeling.Fig.3. depicts structure of offer manifest.

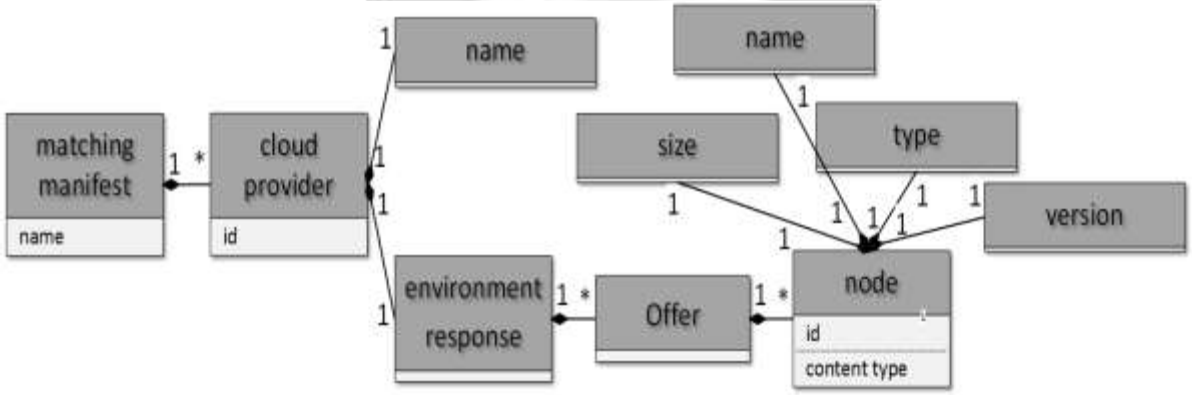

Fig -2: Offer Manifest

Deployment Manifest: The deployment manifests structure is closest to the abstract application manifest. In this manifest new attributes are added about data stores services in the paas node class. These attributes are the size, the type and the version.Fig.4. depicts structure of deployment manifest.

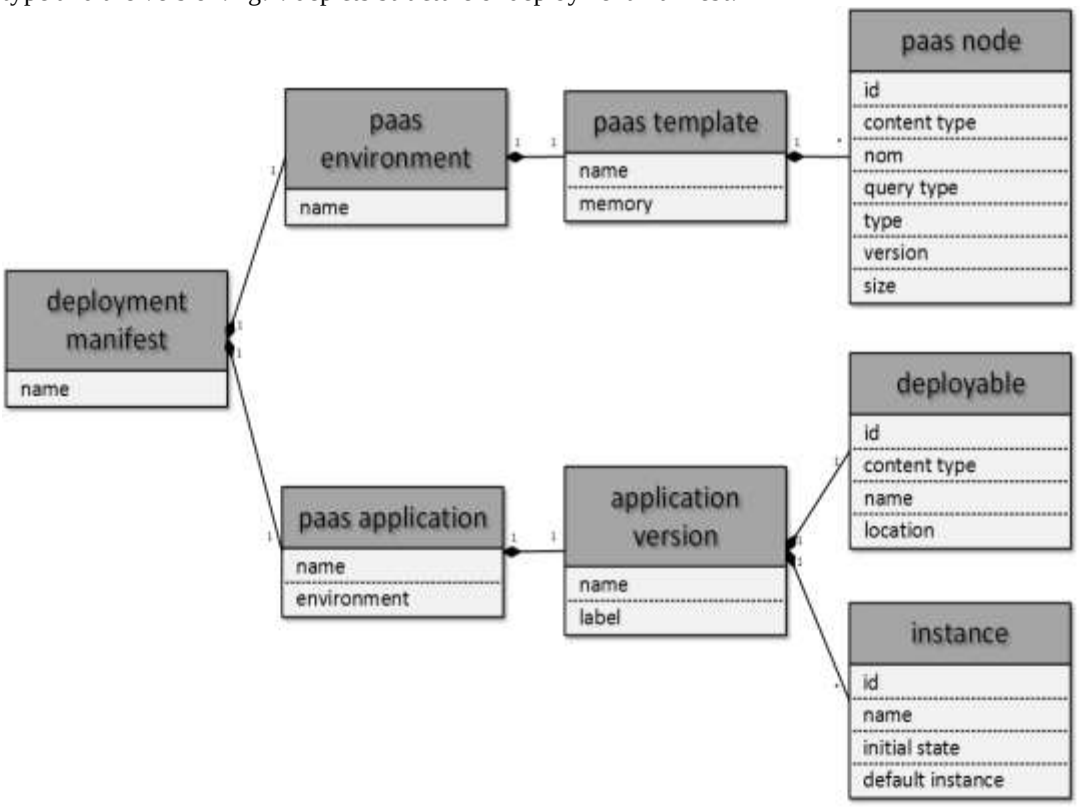

Fig -2: Deployment Manifest

3.2 Algorithm

The matching works as follows:
Step 1: Application developer send AAM to request validater and cloud discovery module (RVCD).
Step 2: RVCD validateds the user request
Step 3: RVCD module reads the list of clouds from its DB and then send REST GET request to all cloud for their specification.
Step 4 : RCVD module generate the offer manifest (OM) request and send it to matching module .
Step 5: Matching module takes OM from RVCD and discovers the best suiting cloud by comparing application developer's requirement with number of available cloud.
Step 6:Matching module deploy the application on the discovered cloud and then return the success response i.e deployment manifest (DM) to RVCD module.
Step 7: On Receiving the DM from MT module RCVD validates DM and return success DM response to the Application developer.

4. RESULTS

System is developed using JAVA and Node.js. The development tool used is netbeans for JAVA and sublime for Node.js. Three types of databases are used SQL, NOSQL and Key-Value. Results are tested out on different cloud platforms. with 1 GB RAM 30 GB disk space.Cloud provider Digital-Ocean is used.

Table 1 show how many percentage of user queries gets resolved as per different number of cloud we are having with us.

Table -1: PERCENTAGE ACCURACY OF WORK

Total number of clouds	User query response success
1	10%
2	20%
3	29%
4	37%
5	45%
6	51%!

5. CONCLUSIONS

A manifest-based solution is proposed for cloud discovery and automatic application deployment. Developer express application and cloud requirements in terms of the Abstract Application Manifest(AAM). Then, developer sends it to the cloud discovery module. This module interacts with the cloud directly to discover the capabilities of each cloud and constructs the offer manifest. This offer manifest is then send to matching technique module which elects best match for the developer's requirement and deploy application on that cloud and return final success response to application developer, this overall approach is making job of application developer more easier for selecting best matching cloud and deploy its application over there. As a future work we are planning to provide a UI portal for the third party cloud owner for updating their clouds' information into the db.

6. REFERENCES

[1] Rami Sellami, Sami Bhiri, and Bruno Defude "Supporting multi data stores applications in cloud environments",IEEE Transactions on Services Computing 2015

[2] C. Baun, M. Kunze, J. Nimis, and S. Tai, "Cloud Computing - Web- Based Dynamic IT Services", Springer, 2011.

[3] T. Kraska, M. Hentschel, G. Alonso, and D. Kossmann, "Consistency rationing in the cloud: Pay only when it matters," PVLDB, vol. 2, no. 1, pp. 253-264, 2009.

[4] R. Sellami, S. Bhiri, and B. Defude, "ODBAPI: a unified REST API for relational and NoSQL data stores," in The IEEE 3rd International Congress on Big Data (BigData'14), Anchorage, Alaska, USA, June 27- July 2, 2014, 2014.

[5] M. Pollack, O. Gierke, T. Risberg, J. Brisbin, and M. Hunger, Eds., Spring Data. O'Reilly Media, October 2012.

[6] P. Atzeni, F. Bugiotti, and L. Rossi, "Uniform access to nonrelational database systems: The sos platform," in Advanced Information Systems Engineering - 24th International Conference, CAiSE 2012, Gdansk, Poland, June 25- 29, 2012. Proceedings, 2012, pp. 160- 174.

[7] H. L. Truong, M. Comerio, F. D. Paoli, G. R. Gangadharan, and S. Dustdar, "Data contracts for cloud-based data marketplaces," IJCSE, vol. 7, no. 4, pp. 280-295, 2012.

[8] Q. H. Vu and et al., "Demods: A description model for dataas- a-service," in IEEE 26th International Conference on Advanced Information Networking and Applications, AINA, 2012 , Fukuoka, Japan, March 26-29, 2012, pp. 605-
612.