2016 5th IEEE International Conference on Cloud Networking

Towards Automated Cost-efficient Data Management for Federated Cloud Services

Vincent C. Emeakaroha, Martin Bullman, John P. Morrison
Irish Centre for Cloud Computing and Commerce
University College Cork, Ireland
Email: {vc.emeakaroha, m.bullman, j.morrison }@cs.ucc.ie

***Abstract*—Cloud computing has transformed the accessibility and usage of information technology resources, by offering them as services via the Internet. Cloud service provisioning spans across infrastructure, platform and software levels. The management of the services at these levels is based on monitoring. The analysis of monitoring data provides insight into Cloud operations in order to make informed decisions. Due to the emergence of numerous heterogenous Cloud platforms with proprietary APIs, service and monitoring data are being formatted using diverse and mostly incompatible data interchange formats. This results to interoperability issues and makes the analysis of monitoring data from multi-Cloud service deployments difficult to handle. The existing research efforts on data interchange formats have been mainly focused on general performance analyses. Little or no effort has been channelled towards a combination of multiple data interchange formats based on data type to achieve efficient serialisation that can facilitate interoperability in federated Clouds, and also reduce the size of data and bandwidth utilisation cost. This paper addresses these issues by presenting automated framework that is capable of automatically selecting the most suitable data interchange formats for achieving an efficient formatting and serialisation outcome. The goal of the framework is to enable robust and transparent communication within and between multiple Cloud deployments. Based on three use case scenarios, we evaluate the proposed framework to demonstrate its efficacy in formatting and serialising data.**

I. Introduction

Cloud computing enables a flexible means of using Information Technology (IT) resources that can be provisioned as services, through the Internet, in a scalable and on-demand fashion.

Current practices for efficient Cloud service provisioning management are based on monitoring. This generates the real-time data regarding the state of the Cloud infrastructures and deployed services. These data are crucial for gaining insight and making decisions. Due to the emergences of numerous monitoring tools for managing heterogeneous Cloud platforms, the generated monitoring data are formatted using diverse and mostly incompatible data interchange formats. This results to interoperability issues [1]–[3] and makes the analysis of monitoring data from multi-Cloud service deployments difficult to handle. Besides, most Cloud usage scenarios require the transfer of consumer data into provider Cloud platforms for computation. Having such data in an incompatible format would severely hinder the usage of Cloud offerings. In addition, the cost of bandwidth for transferring monitoring or consumer data between Clouds could be enormous with increasing data size. Research has shown the positive effects of data interchange formats on reducing the size of data [4].

There is a limitation in the ability of using third party monitoring tools to manage commercial Cloud deployments. This is because each third party monitoring tool has to implement particular Cloud providers' APIs in order to overcome these interoperability issues and be compatible to the platform data. This problem is hindering the independent verification of Cloud service quality, and it can cause distrust for Cloud providers, which might lead to low adoption and usage of Cloud services.

There are different types of data interchange formats that can have various optimisation effects on data size. The existing research efforts on this topic have been mainly focused on general performance analyses [5] and the use of a particular data interchange format for data management [6]–[9]. Little or no effort has been channelled towards a combination of multiple data interchange formats based on data type to achieve efficient serialisation that can facilitate interoperability in federated Clouds and also reduce the size of data and bandwidth utilisation cost.

In this paper, we present an automated framework that is capable of automatically selecting the most suitable data interchange formats to achieve optimal formatting and serialisation outcomes for various data types. The goal of the framework is to enable efficient and transparent communication within and between multiple Cloud deployments. Therefore, it includes a messaging bus mechanism that integrates with the data interchange formats to achieve the required interoperable communication. Based on three use case scenarios, we evaluate the proposed framework to demonstrate its efficacy in formatting and serialising data.

The main contributions of this paper are (i) the review and analysis of standard data interchange formats for usage in Cloud deployments, (ii) the design of an automated framework for transparent combination of multiple data interchange formats to achieve optimal data serialisation and (iii) the evaluation of the proposed framework based on practical use case scenarios to determine its fitness for purpose and compactness for efficient and low-cost data transmission in Clouds.

978-1-5090-5093-2/16 $31.00 © 2016 IEEE
DOI 10.1109/CloudNet.2016.37

The rest of the paper is organised as follows: Section II presents some background information and analyses the previous research efforts on this topic. In Section III, we discuss the review of some selected data interchange formats that are suitable for usage in Cloud deployments. Section IV gives the details of the proposed automated framework and the messaging bus mechanism including their implementation details. Section V discuss the evaluation of the framework based on three use case scenarios and Section VI concludes the paper and highlights our future work in this direction.

II. Background and Related Work

An objective of the standardisation efforts in Cloud computing by organisations, such as Open Cloud Computing Interface (OCCI) [10], is to solve the problems of interoperability between heterogeneous Cloud platforms [11]. Despite this endeavour, commercial Cloud solutions today are being provisioned to consumers using proprietary platforms from competing vendors. Consequently, this results to vendor lockin for consumers by reducing their flexibility to migrate application and data from one vendor to another, for example, to achieve reduced cost or take advantage of improved performance, due to lack of interoperability.

Cloud federation has emerged as a means of increasing the potentials and benefits of Cloud computing [12]. Cloud federation enables the usage of multiple Cloud platforms for provisioning consumer applications. Interoperability issues are adversely affecting the goals and management of such systems. The management of Cloud deployments is based on continuous monitoring and the analysis of the derived data. Platform-specific monitoring services, such as Amazon CloudWatch and Microsoft AzureWatch, can be used to gather detailed information at the infrastructure, platform and application levels. These proprietary systems, however, use differing and incompatible message formats, resulting in a negative impact on portability and interoperability. In addition, the communication of the gathered monitoring data consume a lot of bandwidth due to inefficient formatting. Therefore, to address these issues, efficient and platform-neutral data interchange formats are required.

The existing literature on this topic are mainly focused on the general performance analysis of data interchange format instead of on the means of optimally using them to achieve efficient interoperable data serialisation. Wang *et al.* [4] discuss the efficiency analysis of data interchange formats for Ajax applications. Their goals are to reduce data redundancy, improve processing time and increase the performance of Ajax applications. Their approach is similar to ours, but they do not consider binary data interchange formats and the application of data interchange formats in Clouds. In a previous work [5], we analysed the performance of some selected data interchange formats in terms of performance in structuring and serialising Cloud related data. The work presented an initial effort to assess the usability of data interchange formats to address interoperability issues in Clouds. The scope of the work was limited to that, and it does not consider the possibility of automating the selection of data interchange formats based on data types to achieve optimal efficiency.

Horch *et al.* [13] present a combined sensor data management system for structural health monitoring and building safety and security. This work proposed a unified data management system for sensors and structural health monitoring systems. The authors discussed the importance of data interchange format in achieving such a unified data management system. Their solution however, applied only XML data interchange format and neither considered the other interchange formats nor their usage in Cloud deployments. Wehner *et al.* [7] discuss using JSON to manage communication between services in the Internet of things. In this work, the authors describe how they used JSON data interchange format to achieve interoperable communication between an FPGA-based system and distributed computing nodes. The work did not consider the usage of other data interchange formats. In a recent paper, Miller [6] proposes a common format for electromagnetic data interchange. she highlighted the effects of a lack of common format, which include hampering progress, workload duplicating and increasing chances of error. The work proposed a solution based on JSON and it does not consider the other interchange formats.

For a large data management, Tzeng *et al.* [14] discuss the challenges and opportunities in managing remote sensing signatures database. This work describes the issues with storing, classifying, searching and exchanging heterogeneous signature data collected by a variety of sensors. This huge amount of data is inefficient to communicate and process due to lack of adequate formatting and serialisation mechanism. The usage of multiple standard data interchange format for the different domains was suggested as a possible solution. The paper does not consider Cloud monitoring data and the automatic selection of appropriate data interchange format for Cloud domain.

To the best of our knowledge, none of the existing research efforts extensively analyses data interchange formats to determine their usability in Clouds. In addition, none considers the strategy of transparently applying data interchange formats to facilitate low-cost efficient interoperable data transmissions.

III. Review of Data Interchange Formats

We separate the data interchange formats into two categories namely human readable and binary formats. In the next sections, we present short details of their characteristics.

A. Human Readable Formats

The human readable data interchange formats consist of those data interchange formats that are self-discriptary and their schema can be easily interpreted by humans. In this category, we analyse the following data interchange formats:
eXtensible Markup Language (XML) is a widely used standard format for data representation in applications including Web Services [15]. XML is designed to provide simplicity, generality and usability of data over the Internet [16]. Data representation in XML is text based and position independent,

which makes it suitable for usage in different platforms and heterogeneous Internet environments. XML is easy to read and write by both human and machine. However, the verbose nature of XML can make it unsuitable for some applications.

JavaScript Object Notation (JSON) [17] is a text-oriented light-weight human readable data interchange format. It is designed for the representation of simple data structures and associative arrays. JSON is language independent but uses conventions such as those from languages like Java, C, C++ and JavaScript.

Comma Seperated Values (CSV) file stores data in plain text. The data could be number or text in a tabular form. Each line of the file is a data record that may consist of one or more fields separated by commas. CSV is simple to implement and parse. It is easy to edit manually and can be processed by almost all the existing applications.

Hashmap is a human readable data structure that is based on hashing. It allows a key-value pair data formatting and storage. Access to data is very quick if the key is known. It can be nested and therefore supports heavily nested data. It is easy to implement and provides many library functions.

ArrayList is a variable length human readable data structure. It is loosely typed, which means that it can store different types of data. ArrayList can grow or shrink its size at runtime. It has a wealth of available library functions.

B. Binary/Machine Readable Formats

The binary (also known as machine readable) data interchange formats are the low-level non human readable formats. Their outputs consist of bytes that can be directly interpreted by machines. The following binary formats were analysed:

Binary JavaScript Object Notation (BSON) presents a binary encoded serialisation of JSON. As with JSON, it supports the embedding of documents and arrays within other documents and arrays. BSON provides additional data types that are not available in JSON and it is smaller and faster than JSON.

MessagePack [18] aims to represent simple data structures such as arrays and associative arrays as compactly and simply as possible. It possesses data structures, which loosely corresponds to those used in JSON. It is more compact than JSON but has limitation on array and integer sizes.

Hessian is a dynamically-typed binary data serialisation format. It is compact and portable across programming languages and does not require external schema or interface definitions. It supports Unicode strings, encryption, compression, signature and transaction context envelopes.

KRYO is a flexible and fast serialisation framework that produces small serialised outputs. It aims to provide speed, efficiency and easy use of API. It can serialise primitive types, strings, byte arrays, ArrayList, HashMap, HashSet, and the Clojure collection types.

Concise Binary Object Representation (CBOR) in its design, aims to achieve extremely small code size and extensibility without the need for version negotiation. It is based on JSON and does not require schema to serialise or de-serialise any of the java types.

IV. Framework Design and Implementation

In this section, we present the design of the proposed automated framework. Its goal is to achieve an efficient data formatting and serialisation in order to facilitate a low-cost unified data management in Cloud deployments. This solution is inspired by our previous work [5] from which we determined that a single data interchange format is not adequate for serialising all data types.

Efficient solutions can be achieved by using suitable data interchange format for specific data types. However, doing this manually is not feasible and can be very difficult to achieve in dynamic Cloud environments. To achieve a holistic and flexible solution, an automated mechanism is necessary. Figure 1 presents the architecture of our proposed framework.

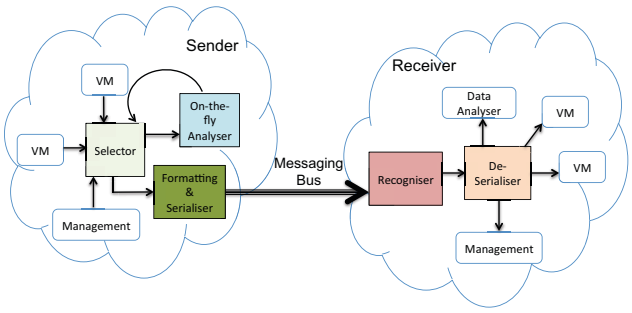

Fig. 1: Automated Framework Overview

As shown in Figure 1, the framework is made up of several components that are working together to achieve its common goal. The components are arranged into two groups (i) the sender and (ii) the receiver.

The sender provides the interface for receiving data to be formatted and serialised. Upon data reception, the selector component analyses the data to recognise its type so as to transparently select the appropriate data interchange format. The selector component is designed based on the knowledge gained from the initial performance analysis of data interchange formats with specific data types. If the selector could not determine an association of a particular data type with an interchange format, it forwards a sample of the data to the on-the-fly analysis component. This component carries out a test of formatting and serialising the data using all the data interchange formats and reports back (training feedback) to the selector component the most suitable interchange formats to use. The serialiser component is responsible for formatting and serialising the data using the selected data interchange formats. There might be a need to first convert the human readable form of the data from one type to another before applying the binary serialisation to improve efficiency. The serialised output is sent through the communication mechanism to the end receiver.

The receiver at the end of the communication mechanism receives the serialised data through the recognition component.

This component is designed to identify the data interchange formats used for serialising the data and forwards this information together with the data to the deserialiser component. After the data is deserialised, this component makes it available to applications for further processing.

A. Communication Mechanism

In order to communicate data between entities such as applications, devices or Clouds, there is a need for a messaging system. Cloud computing involves different components and layers of interaction. To enable seamless communication, we aim to realise a mechanism with the following capabilities: (i) *Inter-application communication*: Enables applications executing in a Cloud or in multiple Clouds to interact and exchange information; (ii) *Intra-Cloud communication*: This ability embodies the interaction among resources and their control entities in a Cloud platform. It supports resource allocation, load-balancing virtual machines and application deployment; (iii) *Inter-Cloud communication*: Supports the outsourcing of resources or service executions between Clouds. This facilitates Cloud federation and (iv) *Notification/Alerting*: Enables one way communication notifying or alerting users and administrators.

From a provider's perspective, these messaging types cover the communication requirements for efficient management of Cloud deployments.

B. Implementation Details

The proposed framework is implemented in Java. It consist of four Java packages that implements the function of the components. Note that the data interchange formats are generic and can be implemented using different programming languages. The selector component is realised in the main package, which connects to the other packages in the sender group. The receiver group can be independently deployed and therefore has its own main package. The communication mechanism connects both of the groups. Note that these two groups of components must not necessarily be deployed on two separate Cloud platforms. Both the sender and receiver group can be deployed on a Cloud platform for efficient data management within the Cloud environment.

The framework makes an extensive use of well established open source libraries to implement the functionalities of the data interchange formats. Some of the key libraries used include *Jackson Core, BSON4Jackson, CBOR for Java, KRYO, MessagePack* and *Hessian*.

To implement the communication mechanism, we use the RabbitMQ broker API, which is based on the Advanced Message Queuing Protocol (AMQP) [19]. RabbitMQ provides well-tested open source API implementations that are widely used in many research projects including the Contrail project, which uses it in their current efforts to address interoperability at the IaaS layer [20].

V. Evaluations

In this section, we test and demonstrate the efficacy of our proposed automated framework approach as a proof of concept.

A. Experiment Environment Setup

Two OpenStack Cloud platform installations running Ubuntu Linux was used as the experimental environment. The basic hardware and virtual machine configurations of the OpenStack platforms are shown in Table I. We use the Kernel-based Virtual Machine (KVM) hypervisor for hosting the virtual machines.

TABLE I: Cloud Environment Hardware for Each OpenStack Platform.

Machine Type = Physical Machine

OS	CPU	Cores	Memory	Storage
OpenStack	Intel Xeon 2.4 GHz	8	12 GB	1 TB

Machine Type = Virtual Machine

OS	CPU	Cores	Memory	Storage
Linux/Ubuntu	Intel Xeon 2.4 GHz	1	2048 MB	50 GB

As shown in Table I, the physical machine resources are capable of supporting on-demand starting of multiple virtual machines for hosting different Cloud services.

B. Use Case Scenarios

In our evaluation, we use data generated from three use case scenarios. The first two use cases are based on the monitoring and gathering data from a service provisioned in one of the OpenStack platforms. For this purpose, we deploy a transactional video-serving web application that responds to requests and makes queries to back-end databases. This service runs on top of a load balancer that distributes the incoming request on the compute resources and can start new virtual machines automatically to sustain heavy load. We simulate user behaviours in terms of placing queries and requests to the Cloud service using Apache JMeter [21]. The workload consists of three HTTP queries and two video rendering requests. The gathered data includes:

1) **Application data:** This use case gathers application performance data from the deployed video-serving web application. This is done by using an application-level monitoring framework [22] to monitor the execution of the application. In this case, we gather data consisting of 17 metrics such as *BytesReceived, BytesSent, ResponseTime* etc.
2) **Resource data:** This use case is based on the same video-serving web application deployment. However, the focus is at the resource utilisation levels. We use a novel resource monitoring tool [23] to monitor the resource utilisation by the application. We gather data consisting of about 57 metrics such as *FreeDisk, CPUUserLevel, FreeMemory* etc.
3) **Sensor data:** Our third use case is different from the previous ones. The data, in this case, is derived from

sensors mounted on city buses in an European city to measure entities such as *CO2, CO, NO2* etc. that are being emitted by the buses. About 10000 data entries were gathered per each bus. In this work, we use this data as a large dataset.

These three use cases are chosen to be representative of the common data types in Clouds. The application data is smaller than the resource data, which is smaller than the sensor data. This shows that we are also investigating the effects of data sizes on the serialisation techniques. In all experiments in this paper, we use five human readable and five binary interchange formats. This provides a total of 25 combinations.

C. Application Data Analysis Result

This section shows the achieved results in analysing the application data type using different combination of the human readable and binary data interchange formats.

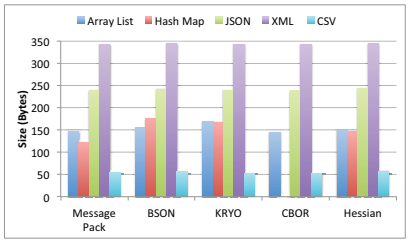

Fig. 2: Application Data Serialisation Outputs

Figure 2 presents the formatting and serialisation options for the automated framework in using a combination of the two technique groups. This result shows the performance of the different techniques in terms of producing compact and small serialisation outputs. The application data was originally in XML human readable format. The data was then automatically converted to the other human readable formats during the analysis. As can be seen from the results, there is a huge difference in performance between combining XML with the binary formats and CSV with the binary formats. In this case, we achieve the best performance with the combination of CSV and KRYO formatting and serialisation techniques. Our automated framework is trained with this result and anytime it encounters this data type, it will select this best solution for data formatting and serialisation.

D. Resource Data Analysis Result

This section presents the analysis of the gathered resource monitoring data.

Figure 3 presents the achieved results. The resource data is originally gathered in XML format and automatically converted to the other formats during the analysis. In this case, we observe that the combination of CSV with the binary data interchange formats performed best. The achieved outputs are seven times better as compared to serialising with XML. The combination of CSV and MessagePack or CSV and CBOR produced identical best results for this data type. Therefore, the automated framework can choose either of them for formatting and serialising resource monitoring data.

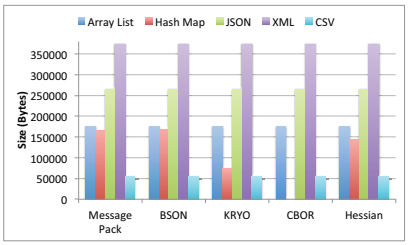

Fig. 3: Resource Data Serialisation Outputs

E. Sensor Data Analysis Result

The sensor use case scenario has the largest amount of data in this experiment.

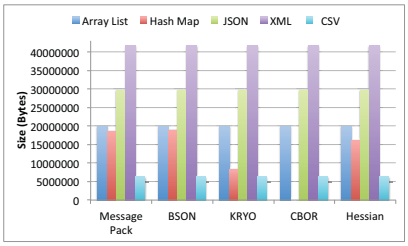

Fig. 4: Sensor Data Serialisation Outputs

The achieved result is presented in Figure 4. The sensor data was gathered in CSV format. During the analysis, the automated framework converts it into the other formats for processing. As can be observed in Figure 4, there is a big difference in size when the data is converted to, for example, XML and serialised from that point. This clearly shows that the extra elements used by such a human readable data interchange format in formatting data can have a tremendous effect on the data size. The implication is that such practices should be avoided to increase performance and reduce cost of communicating data in Clouds. From the analysis in this section, the combination of CSV and Hessian binary format produced the best result for formatting and serialising this data type.

F. Result Discussion

The experiments in this paper provided a detailed insight on how the automated framework chooses the most appropriate formatting techniques for the different data types. The aims are to produce the most compact serialised data output in order to reduce the bandwidth cost of communicating data and to achieve interoperability between heterogenous Cloud platforms. Note that in all analyses, the result set for the combination of Hash Map and CBOR is empty. This is because our implementation could not combine them.

As can be observed from the achieved results of the three use case scenarios, each of them uses a different binary interchange format to achieve the smallest serialised output. This is indicative of the importance of an automated approach to this problem. It would be infeasible to achieve this goal

manually. In addition, the huge differences in size between the serialisation options indirectly shows the amount of bandwidth an efficient serialisation option could save. This translates into a cost reduction for Cloud users.

Our proposed automated framework has been trained with the achieved results in these experiments to increase the speed of serialising such data types in the future. In the case of a new data type, the automated framework, as described in Section IV, carries out the analysis of the new data sample on the fly to determine the most suitable serialising techniques to use. The outcome of this analysis is used to update the knowledge base for future operations. Therefore, the automated framework is designed to learn and adapt to new challenges.

The proposed solution in this paper is deemed to have a big economical impact on the Cloud market. For example, it will facilitate a deeper penetration of third party monitoring tools, which will increase the independent verification of Service Level Agreements (SLA) and other Cloud provider promises. Such effort increases the reliability and trust in Cloud services, which will have a positive impact on their adoption and usage thereby driving up economic growth in this sector.

VI. Conclusions

This paper presented an automated framework for determining and selecting suitable data interchange formats to achieve efficient formatting and serialisation of different data types. The goal of the framework is to enable robust and interoperable communications within and between multiple Cloud platforms. In addition, it reduces the cost of communication through efficient bandwidth utilisation.

Based on these scenarios, the framework was evaluated to show its efficiency in selecting the suitable techniques for formatting and serialising different data types to achieve the most compact outputs. The achieved results demonstrated the novelty of the framework in achieving interoperability and saving bandwidth, which translates to reduced cost of communication.

In the future, we intend to improve on the signifying and recognising components of the framework. In the current state, we use a tightly coupled synchronisation method to let the receiver know the type of technique the sender is applying so as to facilitate a correct deserialisation process. We aim to relax this process in order to achieve our goal of developing a fully transparent and independent automated framework.

Acknowledgements

The research work described in this paper was supported by the Irish Centre for Cloud Computing and Commerce, an Irish national Technology Centre funded by Enterprise Ireland and the Irish Industrial Development Authority.

References

[1] Z. Zhang, C. Wu, and D. W. Cheung, "A survey on cloud interoperability: taxonomies, standards, and practice," *SIGMETRICS Performance Evaluation Rev.*, vol. 40, no. 4, pp. 13–22, Apr. 2013.

[2] S. Dowell, A. Barreto, J. Michael, and M.-T. Shing, "Cloud to cloud interoperability," in *2011 6th International Conference on System of Systems Engineering (SoSE)*, 2011, pp. 258–263.

[3] N. Loutas, E. Kamateri, F. Bosi, and K. Tarabanis, "Cloud computing interoperability: The state of play," in *2011 IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom)*, 2011, pp. 752–757.

[4] P. Wang, X. Wu, and H. Yang, "Analysis of the efficiency of data transmission format based on Ajax applications," in *2011 International Conference on Information Technology, Computer Engineering and Management Sciences (ICM)*, vol. 4, 2011, pp. 265 – 268.

[5] V. C. Emeakaroha, P. Healy, K. Fatema, and J. P. Morrison, "Analysis of data interchange formats for interoperable and efficient data communication in clouds," in *IEEE/ACM 6th International Conference on Utility and Cloud Computing (UCC)*, Dec 2013, pp. 393–398.

[6] J. R. Miller, "A proposed common format for electromagnetics data interchange," in *2016 IEEE/ACES International Conference on Wireless Information Technology and Systems (ICWITS) and Applied Computational Electromagnetics (ACES)*, March 2016, pp. 1–2.

[7] P. Wehner, C. Piberger, and D. Ghringer, "Using json to manage communication between services in the internet of things," in *Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), 2014 9th International Symposium on*, May 2014, pp. 1–4.

[8] Y. Li, L. Yang, and Y. Li, "Research on Apparel CAD pattern data interchange format based on XML," in *WRI World Congress on Software Engineering, 2009. WCSE '09*, vol. 3, 2009, pp. 90–94.

[9] D. Crockford, "The application/ JSON media type for JavaScript Object Notation (JSON)," 2006.

[10] A. Edmonds, T. Metsch, A. Pappspyrou, and A. Richardson, "Towards an open cloud standard," *IEEE Internet Computing*, vol. 16, no. 4, pp. 15–25, 2012.

[11] G. Lewis, "Role of standards in cloud-computing interoperability," in *2013 46th Hawaii International Conference on System Sciences (HICSS)*, 2013, pp. 1652–1661.

[12] R. Buyya, R. Ranjan, and R. N. Calheiros, "Intercloud: utility-oriented federation of cloud computing environments for scaling of application services," in *Proceedings of the 10th international conference on Algorithms and Architectures for Parallel Processing - Volume Part I*, ser. ICA3PP'10, 2010, pp. 13–31.

[13] C. Horch and F. Schaefer, "A combined sensor data management system for structural health monitoring and building safety and security," in *Sensors and Measuring Systems 2014; 17. ITG/GMA Symposium; Proceedings of*, June 2014, pp. 1–5.

[14] N. H. Tzeng, R. G. Best, S. K. Jacobs, P. A. McCauley, V. R. Ramachandran, H. J. Mitchell, and B. M. Rodriguez, "Remote sensing signatures database - challenges and opportunities," in *2015 IEEE Aerospace Conference*, March 2015, pp. 1–8.

[15] Y. Yahui, "Impact data-exchange based on XML," in *Computer Science & Education (ICCSE), 2012 7th International Conference on*. IEEE, 2012, pp. 1147–1149.

[16] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, "Extensible markup language (XML)," *World Wide Web Journal*, vol. 2, no. 4, pp. 27–66, 1997.

[17] K. Maeda, "Performance evaluation of object serialization libraries in XML, JSON and binary formats," in *2012 Second International Conference on Digital Information and Communication Technology and it's Applications (DICTAP)*, 2012, pp. 177–182.

[18] ——, "Comparative survey of object serialization techniques and the programming support," *Journal of Communication and Computer*, vol. 9, pp. 920 – 928, 2012.

[19] S. Vinoski, "Advanced message queuing protocol," *IEEE Internet Computing*, vol. 10, no. 6, pp. 87–89, 2006.

[20] P. Harsh, F. Dudouet, R. Cascella, Y. Jegou, and C. Morin, "Using open standards for interoperability issues, solutions, and challenges facing cloud computing," in *2012 workshop on systems virtualiztion management (SVM) in conjunction with 8th international conference on Network and service management (CNSM)*, 2012, pp. 435–440.

[21] Apache Software Foundation, "Apache JMeter," http://jmeter.apache.org/ Accessed on 26/05/2016, 2016.

[22] V. C. Emeakaroha, T. C. Ferreto, M. A. S. Netto, I. Brandic, and C. A. F. D. Rose, "Casvid: Application level monitoring for sla violation detection in clouds," in *2012 IEEE 36th Annual Computer Software and Applications Conference*, July 2012, pp. 499–508.

[23] V. C. Emeakaroha, I. Brandic, M. Maurer, and S. Dustdar, "Low level metrics to high level SLAs - LoM2HiS framework: Bridging the gap between monitored metrics and SLA parameters in cloud environments," in *2010 International Conference on High Performance Computing and Simulation (HPCS)*, July 2010, pp. 48 –54.