2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing

Analysis of Data Interchange Formats for Interoperable and Efficient Data Communication in Clouds

Vincent C. Emeakaroha, Philip Healy, Kaniz Fatema and John P. Morrison
Irish Centre for Cloud Computing and Commerce
University College Cork, Ireland
{*vc.emeakaroha, p.healy, k.fatema, j.morrison*}*@cs.ucc.ie*

***Abstract*—Efficient mechanisms for data structuring and formatting are indispensable for managing data traffic between and within federated Cloud environments to avoid excessive bandwidth cost and to ensure portability and interoperability. This facilitates provider-agnostic communication, which is essential for interoperable inter-Cloud deployments and portable integration of service components, both with one another and with the underlying Cloud platform. The existing data interchange formats for structuring and serialising data have not yet been analysed in the context of data communication in Clouds. Thus, to address this issue, the determination of an appropriate data interchange format for Clouds is necessary. In this paper, we present a performance analysis of some selected data interchange formats to assess their efficiency in terms of their usability in realising a common messaging format for communicating data in Clouds. We first describe the characteristics of each data format for clear understanding. As a basis for the analysis, we introduce a Cloud use case scenario comprising the transmission of monitored data as messages. The communication means is achieved with a novel message bus system designed to integrate the data interchange formats. We present some evaluations of the formats and compare their compactness for supporting efficient data transmission in Clouds.**

***Keywords*-Data Interchange Format, Efficient Communication, Message Bus System, Interoperability, Cloud Computing**

I. Introduction

The number of customers turning to Cloud to achieve economy of scale by using the Cloud's seemingly unlimited resources to execute their services is on the rise [14]. Moreover, the increasing number of Cloud providers with heterogenous platforms is restricting the flexibility of customers in terms of using different Cloud providers' resources to execute their applications [17]. This is due to a lack of interoperability among Cloud platforms, especially regarding the application data formats being used by the providers [11]. A means of addressing this problem is the use of a common data interchange format, which is capable of structuring and serialising data into a platform-neutral fashion. In addition, a mechanism to remove redundant data components and thereby reduce the amount of data transmitted is desirable.

Efficient communication of data within and between Clouds plays an important role in mitigating the cost of using Cloud offerings or computing large workloads. The effects of this can be heavily felt in scenarios where large chunks of data have to be transferred from the customer into the Cloud for computations.

Existing data interchange formats such as *eXtensible Markup Language (XML), JavaScript Object Notation (JSON) and Protocol Buffers*, are commonly used for Internet data transmissions. However, little or no empirical information is available on the efficiency of these formats or their extensions for inter- and intra-Cloud data communication.

In this paper, we present a performance analysis of some selected data interchange formats to assess their efficiency in terms of their usability in realising a common messaging format for communicating data within and between Clouds. We first describe the characteristics of each data interchange format for clear understanding. As a basis for the analysis, we introduce a Cloud use case scenario comprising the transmission of monitored data as messages. The communication means is achieved using our novel proposed message bus system designed to integrate the data interchange formats. The use case scenario is evaluated to demonstrate the structuring and serialisation capabilities of each of the data interchange formats and, to compare them in respect of their compactness for efficient transmission over the message bus.

The rest of the paper is organised as follows: Section II presents the detailed problem statement and the related work. In Section III, we describe the characteristics of the data interchange formats. Section IV describes the novel message bus system while Section V presents the evaluations. Section VI concludes the paper and discusses our future work.

II. Background and Related Work

In this section, we describe the problem statements and analyse the related work.

A. Background and Problem Description

There have been attempts to standardise various aspects of Cloud Computing, such as Open Cloud Computing Interface (OCCI) [2]. Efforts have also been made to provide open implementations of Cloud services, such as OpenStack [10]. Despite these efforts, Cloud Computing to date has

CFP13UCC-USB/13 $26.00 © 2013 IEEE
DOI 10.1109/UCC.2013.79

been largely characterised by competition between closed proprietary platforms. Some *de facto* standards, such as the EC2 web service interface for IaaS, have emerged but this has occurred outside of any formal standardisation process.

As a result of the emergence of proprietary, vendor-specific, "walled garden" Cloud platforms, the issues of interoperability and portability between these platforms have come to the fore. The issue of interoperability arises from the wish to create service deployments composed of resources from multiple Cloud providers. Deployments such as these may require communications between resources, such as load information, in order to function effectively (see Figure 1).

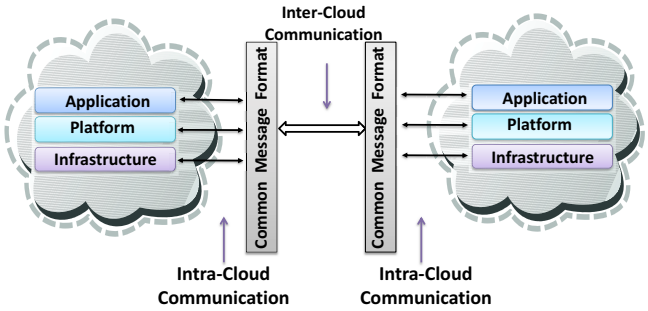

Figure 1: Interoperable and Portable Cloud Data Communication

A standard mechanism for performing communications such as these does not exist. A number of proprietary messaging services are available, such as Amazon's Simple Notification, Oracle Messaging and CloudPrime. However, these services use a variety of underlying messaging technologies and incompatible message formats. A common, consensus-based, message format would facilitate Cloud interoperability and portability by abstracting message content from the underlying delivery mechanism.

This common Cloud messaging format would also facilitate integration between components of Cloud service deployments, and between Cloud service deployments and the underlying Cloud platform. A related issue is the lack of common communication mechanism among the Cloud platforms. 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. For example, Amazon's Elastic MapReduce surfaces detailed platform-specific information, such as the number of map and reduce tasks, via CloudWatch. These proprietary systems, however, use differing and incompatible message formats, resulting in a negative impact on portability and interoperability.

A common messaging format for messages relating to Cloud deployments would solve many of the issues identified above. Such a format could be developed through a formal standardisation process, or could emerge by consensus amongst Cloud developers, integrators and providers. In this paper we seek to identify most suitable format for packing and aggregating such messages.

B. Related Work

In this section, we analyse the previous research efforts on this topic.

Gil *et al.* [4] discuss the impact of data interchange formats on energy management for mobile devices. The objective is to determine the cost of processing data using XML, JSON and Protocol Buffers on a mobile device in order to extend the battery life time. However, they did not consider the transmission of compute data. Maeda [6] provides a survey of six data interchange formats' object serialisation. The objective is to compare their qualitative and quantitative aspects, and to determine whether schema definitions are required or not. However, the study does not investigate the application of data interchange formats in Clouds.

Miguel-Hurtado *et al.* [9] propose an analysis of compact data formats for the performance of handwritten signature biometrics. The concept is related to our approach in this paper, but their focus is on capturing and structuring biometric data coming from a Signature Input Device to facilitate efficient processing and interoperability.

Wang *et al.* [15] 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 did not consider binary data interchange formats and the application of data interchange formats in Clouds.

Mail Bypass Inc [8] offers a commercial message bus platform for email services. However, this communication mechanism uses proprietary message formats like the other platforms such as Oracle messaging. In contrast, our proposed message bus is designed to use platform-neutral formats to enable interoperability.

In the next section, we describe the data interchange formats.

III. Data Structuring and Serialisation Methods

In this section, we present detailed descriptions of the selected data interchange formats. The selection of these formats is based on their prominence in the existing literature.

A. eXtensible Markup Language

eXtensible Markup Language (XML) is a widely used standard format for data representation in applications including Web Services [16]. XML is designed to provide simplicity, generality and usability of data over the Internet [1]. 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, XML does not match with the data

model of most programming languages and it requires that the structure of data be translated into document structure, which can make the mapping complicated. The verbose nature of XML can make it unsuitable for some applications

B. JavaScript Object Notation

JSON [7] 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.

In JSON, messages can be organised as objects composed of key value pairs or array of objects. The messages are serialised into string streams, which in some instances requires further conversion into binary bytes before transmission. BSON provides a binary serialisation of JSON.

C. MessagePack

MessagePack [6] is a binary data interchange format. It aims to represent simple data structures such as arrays and associative arrays as compactly and simply as possible. MessagePack 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.

MessagePack supports various data types, including fixed length types (*e.g.,* integer, boolean, floating point), variable length types (e.g., raw bytes) and container type (*e.g.,* arrays, maps). It provides portability across many languages. Implementations are available for a number of programming languages including C/C++, C#, Java and Python.

D. Protocol Buffers

Protocol Buffers is an interchange format developed by Google [5]. It provides a language and platform neutral way of serialising and structuring data for usage in communication. To provide smaller and faster serialisation, it encodes data into binary form. It allows users to define their suitable data structure and provides an efficient automated mechanism for serialising the structured data. Once the user defines a data structure, a compiler is used to generate code for writing and reading the structured data to and from a variety of data streams using different programming languages.

Each Protocol Buffers message consists of a series of name-value pairs. Each message type has one or more uniquely numbered fields, and each field has a name and a value type, where value types can be numbers, booleans, strings, raw bytes, or other protocol buffer message types. Although this mechanism is efficient in terms of data serialisation, it is not easy to use due to the requirement of an extra initial compilation step.

IV. Communication Mechanism

This section presents the details of the message bus system.

A. Communication Capability

Communication is a fundamental requirement for the management of single and distributed systems. 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-layer communication*: This ability embodies the interaction among resources and their control entities in a Cloud layer; (iii) *Cross-layer communication*: Enables the management of layers from the provider perspective. The cross layer communication supports resource allocation, load-balancing virtual machines and application deployment; (iv) *Inter-Cloud communication*: Enables the outsourcing of resources or service executions between Clouds. This facilitates techniques such as Cloud bursting and (v) *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 provisioning.

B. Design and Implementation

The message bus system is designed to realise the aforementioned communication capabilities. Figure 2 depicts an abstract view of the message bus system.

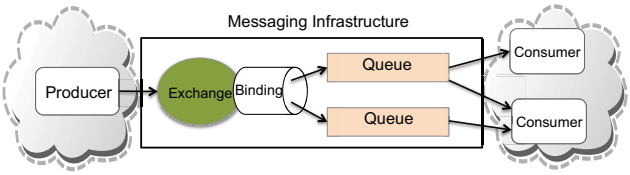

Figure 2: Cloud Message Bus Overview

As shown in Figure 2, it consist of three components: (i) producer, (ii) messaging infrastructure and (iii) consumer.

Producer

The producer in some cases produces the data/message and in other cases receives the data/message for transmission from a source. It integrates the data interchange format for structuring and serialising the messages. The producer has a simple interface for receiving data and this makes it easy to integrate it into different levels in Clouds ranging from applications, management tools or Cloud platforms.

Messaging Infrastructure

The messaging infrastructure provides the functions of a message broker. It asynchronously delivers messages from producers to consumers (synchronisation decoupling). The producer does not need to know the nature or location of a consumer. It simply delivers its messages to the broker, which in turn routes them to the appropriate consumer (space decoupling). The broker therefore enables space, time

and synchronisation decoupling [12]. This feature facilitates the necessarily loose relationship between a producer and a consumer, which is essential in distributed systems like Clouds.

In our prototype implementation, we use the RabbitMQ broker API, which is based on the Advanced Message Queuing Protocol (AMQP) [13]. The RabbitMQ broker consist of exchanges, bindings and queues. As shown in Figure 2, the message producer sends a message to an exchange along with a routing key and not directly to a queue. The exchanges are bound to queues through binding directives. A binding directive indicates what message should be routed from an exchange to a particular queue. The message consumers setup the queues and receive messages from the queues bound to an exchange. If the routing key sent with the message matches the binding specified between the exchange and the queue, then the message is routed to the queue and received by the consumer.

Consumer

The consumer is the receiving end of the communication. It is responsible for receiving and deserialising the message body. It implements a simple interface for easy integration into different Cloud levels.

The whole message bus prototype is implemented in Java language and includes an interface in the producer and consumer for integrating the different data interchange formats.

V. Evaluations

The goals of the evaluations are to surface the performance characteristics of the analysed data interchange formats. We first show the data structuring capabilities and later compare the size, processing time and the amount of bandwidth consumed by each of the data interchange format for transmission.

The evaluations are carried out in a private Cloud testbed using the message bus prototype described above. Table I presents the hardware configurations of the physical machines in the testbed.

Table I: Testbed Hardware Description

Software Stack	CPU	Cores	Memory	Storage
VMware ESXi 5.1	Intel Xeon	12	32 GB	1000 GB

Based on the hardware configurations in Table I, we are able to deploy many virtual machines of different sizes on-demand to execute applications.

A. Use Case Description

As a basis for these evaluations, we describe a use case scenario for efficiently transmitting monitored data, which could support operations such as Service Level Agreement (SLA) enforcement.

The use case consist of measuring application- and low-level resource metric data. The reason for using these two datasets is to be able to investigate the behaviours of the data interchange formats in serialising small and bigger datasets. To derive these data, we employ a monitoring framework [3] to monitor and report on resource usages and application behaviours. The application data is being gathered by monitoring seven metrics of a web application execution. For the resource metric data, we monitor 33 physical/virtual machine resource metrics. Since Cloud can host hundreds or even thousands of virtual machines, the transmission of these monitored data should be efficiently done to maintain performance.

We use this monitored data to evaluate the data interchange formats in terms of serialisation compactness, which directly affects the size of the serialised data and the amount of bandwidth it would require for transmission.

B. Data Structuring

The data interchange formats can be categorised into two groups: (i) self-describing data interchange formats (XML & JSON) and (ii) binary (schema-based) data interchange formats (MessagePack & Protocol Buffers).

The two format groups have their advantages and drawbacks. The self-describing data interchange format group has the strength of being human readable and easy to understand. But, from the transmission perspective, they contain redundant components, which affects their sizes. The binary data interchange format group is not human readable. But, they are more efficient for transmission as can be observed in Section V-C.

In the following, we use each of the data interchange format to structure a sample of the application monitored data.

XML

Listing 1 presents the structuring of the monitored data using XML.

Listing 1: XML Data Structuring Format

```
<webapp>
  <appname>Web Application</appname>
  <appid>178349064</appid>
  <responsetime>5.24312</responsetime>
  <throughput>20</throughput>
  <latency>3.2345</latency>
  <executiontime>10.5</executiontime>
  <availability>99.999</availability>
</webapp>
```

As shown in Listing 1, XML data structuring is human readable and easy to understand. This is one of the major advantages of using XML for encoding data. The big disadvantage is the size of the encoded data.

JSON

Listing 2 shows the structuring of the monitored data using JSON.

Listing 2: JSON Data Structuring Format

```
{"webapp":
```

```
    {
    "responsetime":5.24312,
    "latency":3.2345,
    "appname":"Web Application",
    "appid":178349064,
    "executiontime":10.5,
    "throughput":20,
    "availability":99.999
    }
}
```

The JSON structuring is human readable but uses less characters to describe the data. This results in more compact serialisations compared to XML. It is however highly compatible with XML.

MessagePack

This is a binary data interchange format. It does not offer a human readable data structuring form, but it provides a schema for describing the data element types. Listing 3 presents the schema definition for the monitored data.

Listing 3: MessagePack Schema Definition

```
@Message
public class WebApp {

    public String appName;
    public long appId;
    public double responseTime;
    public int throughput;
    public double latency;
    public double executionTime;
    public double availability;
}
```

The "@Message" annotation indicates a MessagePack schema definition. MessagePack uses the defined schema in Listing 3 for serialising and de-serialising data.

Protocol Buffers

It uses a schema definition to describe the data and the schema must be saved in a ".proto" file. There is a special compiler to read the schema and generate classes for accessing and manipulating the data. Listing 4 shows the protocol buffer schema definition for the monitored data.

Listing 4: Protocol Buffer Schema Definition

```
option java_package = "ucc.cuc.ic4.core";
option java_outer_classname = "WebAppProtos";

message WebApp {
    required string appName = 1;
    required int32 appId = 2;
    required double responseTime = 3;
    required double latency = 4;
    required int32 throughput = 5;
    required double executionTime = 6;
    required double availability = 7;
}
```

The "=1", "=2", etc markers on each field identifies a unique "tag" used in the binary encoding.

C. Compactness and Efficiency of Data Interchange Formats

This section presents the serialisation compactness of the analysed data interchange formats.

The data serialisation is integrated in our implemented holistic message bus system. When deployed, the data is serialised at the producer before being passed to the message bus, which transmits it. The consumer receives and deserialises the data. To determine the processing time, we log the total time for structuring and serialising the data in the producer. The size of the serialised data is captured by writing them to files before sending them into the message bus. The message bus consist of *1Gbit/s* transmission bandwidth. Thus, we calculate the percentage of the Bandwidth Utilisation (BU) in transmitting the data by considering the serialised data size, the transmission bandwidth and the network overhead as expressed in Equation 1.

$$BU = \frac{(DS + Network\,Overhead) * 8 * 100}{Transmission\,Bandwidth} \quad (1)$$

Where *DS* represents serialised data size and newtork overhead includes the ethernet, IP and TCP headers. Note if the sum of the data size and network overhead is greater than 1480 bytes, the network overhead is multiplied by 2 to accommodate the packet fragmentation.

Table II presents the achieved results of serialising the monitored web application and resource metric data including the processing time and the amount of used bandwidths for transmitting the data through the message bus.

Table II: Performance Comparison

Application Data Results			
Format Type	**Size (Bytes)**	**Processing Time (ms)**	**Bandwidth Utilisation (%)**
XML	236	23	0.0001968
JSON	160	14	0.0002576
MessagePack	59	193	0.000116
Protocol Buffers	60	11	0.0001168
Resource Metric Data Results			
Format Type	**Size (Bytes)**	**Processing Time (ms)**	**Bandwidth Utilisation (%)**
XML	1828	32	0.0009712
JSON	1128	25	0.0016
MessagePack	264	223	0.00028
Protocol Buffers	287	11	0.0002984

As shown in Table II, the binary data interchange format group outperforms the self-describing ones in terms of serialized data sizes. Using XML results in the largest output sizes for both datasets and hence the weakest compactness. This can be attributed to the extra tags it uses in structuring/encoding data as shown in Listing 1. JSON has a better result in this group. However, in terms of processing time, XML showed a lesser increase in time as JSON in serialising the bigger dataset.

The binary format group is very compact in terms of serialising data and consequently produces smaller serialised data sizes. In this group, MessagePack has a slightly better performance compared to Protocol Buffers for both dataset. But, in terms of processing time, Protocol Buffers is the clear winner. It even has an average constant processing time for both the smaller and bigger datasets. This is an interesting result, which could be explored further using larger datasets. Overall, there is no clear preference between the two binary formats since they both have strengths and drawbacks.

In terms of bandwidth utilisation, the percentage of the message bus bandwidth used for transmitting single instances of these serialised data as shown in Table II, is small but considering the large-scale nature of Cloud environments, *i.e.,* the number of virtual machines and applications to be monitored, the bandwidth utilisation can increase tremendously. This surfaces the importance of using efficient data interchange formats in order to achieve scalability and to maintain high performance.

VI. Conclusions and Future Work

In this paper, we presented performance analysis of some selected data interchange formats to asses their efficiency in terms of their usability in realising a common data messaging format for Clouds.

We analysed four data interchange formats (XML, JSON, MessagePack and Protocol Buffers) for this purpose. We first classify them into two groups - (i) self-describing data interchange formats and (ii) binary data interchange formats. The goal of the classification is to quantify the advantages and drawbacks of these two different approaches in order to gain knowledge for defining a common message format.

As can be observed in the evaluations, the binary data interchange format group outperforms the self-describing data interchange format group in terms of compactness and the size of serialised data. This does not necessarily imply that the binary formats are superior to the self-describing formats in all aspects. Therefore, to achieve our vision of contributing to the development of a common data interchange format to facilitate efficient communication and interoperability in Cloud, a hybrid approach combining the identified strengths of these two data interchange format groups is the way forward.

In the future, we will expand the use case scenario to include other scenarios in order to increase our knowledge and expertise on this topic. Finally, we will strive to contribute our expertise to the development/standardisation of a platform-neutral and efficient data interchange format to support interoperable communication in Clouds.

Acknowledgment

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] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible markup language (XML). *World Wide Web Journal*, 2(4):27–66, 1997.

[2] A. Edmonds, T. Metsch, A. Pappspyrou, and A. Richardson. Towards an open cloud standard. *IEEE Internet Computing*, 16(4):15–25, 2012.

[3] 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)*, pages 48 –54, July 2010.

[4] B. Gil and P. Trezentos. Impacts of data interchange formats on energy consumption and performance in smartphones. In *Proceedings of the 2011 Workshop on Open Source and Design of Communication*, OSDOC '11, pages 1–6, 2011.

[5] G. Kaur and M. Fuad. An evaluation of protocol buffer. In *Proceedings of the IEEE SoutheastCon 2010 (SoutheastCon)*, pages 459–462, 2010.

[6] K. Maeda. Comparative survey of object serialization techniques and the programming support. *Journal of Communication and Computer*, 9:920 – 928, 2012.

[7] 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)*, pages 177–182, 2012.

[8] Mail Bypass Inc. Message bus platform. www.messagebus.com Accessed on: 17 June, 2013.

[9] O. Miguel-Hurtado, L. Mengibar-Pozo, I. Tomeo-Reyes, and J. Liu-Jimenez. Analysis on compact data formats for the performance of handwritten signature biometrics. In *43rd Annual 2009 International Carnahan Conference on Security Technology*, pages 339–346, 2009.

[10] K. Pepple. *Deploying OpenStack*. Ó Reilly, 2011.

[11] D. Petcu. Portability and interoperability between clouds: challenges and case study. In *Proceedings of the 4th European conference on Towards a service-based internet*, ServiceWave'11, pages 62–74, 2011.

[12] N.-L. Tran, S. Skhiri, and E. Zimanyi. EQS: An elastic and scalable message queue for the cloud. In *2011 IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 391–398, 2011.

[13] S. Vinoski. Advanced message queuing protocol. *IEEE Internet Computing*, 10(6):87–89, 2006.

[14] L. Wang, J. Zhan, W. Shi, and Y. Liang. In cloud, can scientific communities benefit from the economies of scale? *IEEE Transactions on Parallel and Distributed Systems*, 23(2):296–303, 2012.

[15] 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)*, volume 4, pages 265 – 268, 2011.

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

[17] Z. Zhang, C. Wu, and D. W. Cheung. A survey on cloud interoperability: taxonomies, standards, and practice. *SIGMETRICS Performance Evaluation Rev.*, 40(4):13–22, Apr. 2013.