Towards Multi-Tenant and Interoperable Monitoring of Virtual Machines in Cloud

Daniel Tovarňák, Tomáš Pitner
Masaryk University, Faculty of Informatics
Botanická 68a, 60200 Brno, Czech Republic
xtovarn@fi.muni.cz, tomp@fi.muni.cz

***Abstract*—The advent of Cloud Computing introduced new challenges in various computer science fields and disciplines, monitoring being one of them. Due to the multi-tenant nature of Cloud environment, its size and use of massive virtualization, the current monitoring solutions are approaching their limits. Unlike some other approaches, focusing on data integration, aggregation, and abstraction, our work focuses on a virtual machine representing the primary source of monitoring information. In this paper, we propose requirements for the producer of monitoring information addressing existing issues related to monitoring data representation, storage, processing and distribution. As a proof-of-concept, conforming to these requirements, prototype of event-based monitoring daemon is presented in detail. The resulting solution allows multiple users to consume monitoring information in an extensible data format without impairing interoperability.**

***Index Terms*—monitoring, cloud, management, multi-tenancy**

I. Introduction

The monitoring capabilities of any large-scale distributed environment represent a fundamental precondition for the successful operation of such environment. Monitoring information is continuously used in numerous ways, including but not limited to: accounting, audit tracking, debugging, fault detection, job scheduling and performance analysis.

As Cloud computing [7], [19] is growing in popularity it is becoming apparent that the existing approaches to monitoring are at times insufficient – often due to the very nature of Cloud environment itself.

In a traditional three-tier Cloud services model (IaaS, PaaS, SaaS) it is typical that the clients are running business-critical applications and services on infrastructures/platforms which they do not have full control over. However both the client and the Cloud provider need complex information about all the tiers of the system. Also there are situations when it is desirable to be able to restrict access to specific types of monitoring information. Additionally, as the number of Cloud providers, platforms and solutions steadily increases, it is becoming highly desirable to be able to access and process vast amounts of diverse monitoring data in a unified manner.

Differentiating from the work of others (e.g. [10], [25]), we argue that rather than creating another complex monitoring solution there is an urgent need to redesign monitoring from the ground up. The aim of this paper is to present several fundamental requirements (e.g. multi-tenancy or extensible data format) that any producer of monitoring information should provide. We analyze the identified requirements and advocate their necessity for the modern Cloud monitoring. Later on, the prototype implementation aiming to fulfill these requirements is presented and described in detail.

The rest of the paper is structured as follows. Section 2 presents the background for our research and related work. Section 3 proposes requirements for the producer of monitoring information. Section 4 presents *Heimdall*[1] – event-based monitoring daemon that satisfies these requirements. Section 5 summarizes possible future research directions and concludes the paper.

II. Background and Related Work

In our work we follow the classic notion of monitoring process as defined by Mansouri in [16]: (1) *production* of monitoring information, i.e. acquisition and generation of the monitoring data; (2) *processing* of the data, which can take place during any stage of the process; (3) *distribution* (dissemination) of the information from the source to the interested parties, and finally (4) *consumption* which refers to the final stage such as evaluation and visualization.

We refer to the components participating in the process using terminology originally defined by Grid Monitoring Architecture [29]: (1) *sensor* generates the raw monitoring information; (2) *producer* collects, processes and distributes the monitoring data via defined API; (3) *consumer* further processes, evaluates and visualizes the information using the producer's API; (4) *re-publisher* refers to a component consisting both of producer and consumer, and it is usually used for advanced processing, filtering and correlation.

From the Cloud model perspective we adhere to the refinement of the traditional three-tier model introduced by Spring in [26], [27]. In his work the author presents a set of recommended restrictions and audits to facilitate Cloud security by monitoring each tier of a *Cloud 7-layer model*: facility, network, hardware, OS, middle-ware, application and user.

Previous research in the field of distributed systems monitoring, Grids in particular, addressed many issues and provided results that are largely applicable to Cloud environment. For a thorough summary of existing Grid monitoring projects and frameworks see [31] by Zanikolas and Sakellariou.

[1]In Norse mythology, Heimdall is the watchman of the gods, guarding entrance to their realm.

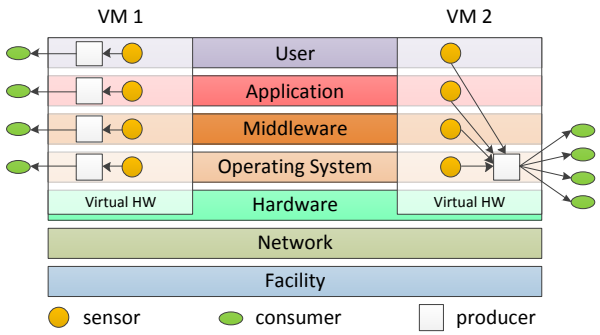

Fig. 1. Monitoring of the virtual machines in Cloud: VM1 – current single-tenant state, VM2 – desired multi-tenant state with unified monitoring information

The emergence of Cloud computing revealed new issues which are the result of its characteristics, e.g. massive virtualization, flexibility and multi-tenancy, and in addition strongly emphasized known issues resulting from design flaws of commonly used technologies.

Current methods include integrating monitoring frameworks, pattern mining techniques, and log processors and semantic analyzers [10], [15], [20], [25], [28]. Usually, such approaches merely translate the problems to a higher level of abstraction and sometimes even further deepen the issues they are aiming to solve. To avoid duplicity, discussion of the related approaches follows in section III.

Hence, instead of focusing on overall monitoring architecture, our primary concern is the producer of monitoring information. We focus on the requirements for the producer with reflection of current knowledge and known shortcomings as a part of discussion leading towards much needed monitoring standards.

This trend is clearly set in the general field of Cloud management. Instead of using multiple wrappers and aggregators that enable transparent access to various Cloud platforms, for example solutions like jClouds [12] or Apache Delta Cloud [9], the experts are leaning towards standardization of the core management functions, including deployment, autonomic scaling, and to some extent monitoring, to achieve basic interoperability capabilities. Here we mention promising emerging standard OCCI (Open Cloud Computing Interface) [6], for remote management of Cloud computing infrastructure. Since the standard is in the stage of extensive development and the monitoring extension is currently being discussed, we believe that our work is highly relevant.

III. Monitoring Information Producer Requirements

In the following paragraphs we present a set of requirements for the producer of monitoring information that we have identified to be critical for modern Cloud monitoring. We analyze these requirements in detail, discuss current state and present how these requirements affect the imperfections revealed by the emergence of Cloud computing.

To be more specific, we focus on production and distribution of monitoring information related to the top five layers of Cloud 7-layer model, i.e. hardware, OS, middle-ware, application, and user as illustrated by Figure 1. Note, that since the physical hardware layer is virtualized, it cannot be considered part of the VM. However, it can be typically monitored by the means of root virtual machine (termed as `dom0` by Xen hypervisor for example) and its underlying OS. The fact, that the same mechanism is used to monitor virtual hardware keeps things simple.

Typically for the Cloud environment, these layers are spanned by thousands of virtual machines and every virtual machine includes all the five layers. This renders virtual machine our main entity of interest. Therefore, in our context the producer to be consuming monitoring information from is a virtual machine addressed by its hostname.

A. Multi-tenancy

One of the most distinctive characteristics of Cloud environment is multi-tenancy, i.e. use of shared resources by multiple users (tenants) simultaneously [2]. The resource being shared can be any of the 7 above-mentioned layers, depending on the specific service model. Note, that the particular service model inherently determines the level of control over each layer. This naturally affects the way these resources are monitored.

Foster et al. in [7] identify the multi-tenant access to monitoring data as a significant challenge for the Cloud computing since, depending on the service model, the users are not able to utilize their own monitoring infrastructure and the produced monitoring data may not provide necessary level of details. This apply to all the interested roles, e.g. end-user, Cloud developer and administrator.

We conclude that not only the users should be allowed to simultaneously access the monitoring information from different layers of the virtual machine, but from them same layer as well, i.e. the same piece of information. On the other hand, the privacy of any tenant should not be compromised. For example in PaaS service model, the middle-ware layer would be of great interest for all the roles, but at the same time on the application level the tenants must be sure that no one can access any kind of confidential business activity information.

This can be summarized as (1) *concurrency*, i.e. multiple consumers must be able to access identical monitoring information simultaneously, and (2) *isolation*, i.e. tenants must not be able to access monitoring information that is not addressed to them.

It is reported that lack of trust is one of the primary obstacles to widespread adoption of Cloud computing [11]. As explained in [14] common security measures as authentication, access control, and encryption form only part of the solution. Overall security referred to as preventative controls must be equally complemented with detective controls such as accountability and auditability.

Additionally, the authors of [8] report that the lack of multi user monitoring results in a limited incident detection and

auditing capabilities. We believe that in order to properly implement detective controls, an additional two conditions must be met: (3) *integrity* - once generated, nobody can modify or delete particular piece of monitoring information, and (4) *proof of origin* - the origin of generated monitoring information must be non-repudiable, i.e. monitoring information cannot be counterfeited.

B. Unified Representation of Monitoring Information

Up to this point we have consistently used an umbrella term *monitoring information* to refer to any piece of data that carries some form of a state of the monitored entity. The three most common types of monitoring information produced by modern systems are logs, notifications and measurements. Logs contain information about activity or change of state of the monitored entity, notifications inform of significant events that need attention, and finally measurements represent a quantitative information or metric.

Significant drawback of the existing systems is that the above-mentioned types of monitoring information are typically found in diverse forms and they are collected very differently. Whilst logs can be found in the form of plain-text files or database records, notifications are often delivered via email or similar means. In contrast, measurements are typically requested repeatedly in short periods of time using proprietary formats and protocols [17].

Recent works predominantly focus on aggregating diverse sources and types of monitoring information. Shao, et. al. in [25] for instance propose Run-time Model for Cloud monitoring (RMCM), which aggregates and organizes monitoring data gathered by multiple monitoring techniques. Consequently, the data are transformed into models on a higher level of abstraction with respect to the role of their consumer, i.e. Cloud operator, service developer and end user. Note, that the approach cannot be considered multi-tenant, since the models are only generated after the data have been collected.

The diversity of monitoring information can lead to a high complexity of the remaining stages of monitoring architecture (i.e. distribution, processing, consumption). Additionally, with different representations and sources data correlation capabilities are limited. This can have negative impact on accountability and auditability which, as we have mentioned, are important features in Cloud computing. Therefore, the producer should collect and represent logs, notifications, measurements and other types of monitoring information in a unified way and format.

C. Extensible Data Format

To further complicate the variability problems, current systems are to the great extent producing free-form monitoring information [15]. Typically, log and notification entries are schema-less and structureless, often with the most important information in the form of natural language string, e.g. *"data requested from lykomedes.fi.muni.cz"*. Such information is hard to process and the extensibility and interoperability of the data format suffers greatly.

The common approaches to this problem include abstraction of the log entry types [20], semantic analysis [15] and various pattern-mining techniques [28]. Therefore, in a way, the modern monitoring information processing is somewhat closer to data-mining than to actual data processing and evaluation.

In order to achieve overall extensibility of the resulting monitoring system the monitoring information must be represented in data format that is (1) *standardized*; (2) *self-describing*; (3) based on *extensible schema*; for the processing to be as efficient as possible the data format must be (4) *structured*, and (5) *compact*, to minimize the overhead.

Here we must mention emerging Common Event Expression (CEE) initiative [3] that focuses on creating a set of general standards for representation and exchange of logs produced by electronic systems. In particular CEE Log Syntax [4] is aiming to tackle some of the above-mentioned requirements.

D. Delivery Channel

Assuming that all the types of monitoring information are represented and collected identically it is simple to access and distribute them identically. By introducing standard channel and accompanying protocol for delivery of monitoring information, the extensibility and interoperability of monitoring systems can increase drastically. The possibility of consuming monitoring data from virtual machine regardless of its underlying hypervisor or middle-ware is important step towards the *inter-cloud monitoring*. The phenomenon of seamless integration of multiple clouds is sometimes referred to as *sky-computing* [13], [22], [23].

There are however several requirements for such delivery channel that are needed in order to assure wide portfolio of applications. The channel should support all interaction patterns as defined by Grid Monitoring Architecture [29] (notification, query/response, publish/subscribe). Notification and publish/subscribe interaction patterns follow *push model*, i.e. the interaction is initiated by the producer. Query/response pattern on the other hand follow *pull model*, i.e. the interaction is initiated by the consumer.

From the reliability point of view, the accompanying protocol should support both synchronous (guaranteed, ordered) and asynchronous data delivery. Each of them is better suited for different scenarios and performance requirements.

IV. Event-based Monitoring Daemon

In this section we present Heimdall [21], prototype of an event-based monitoring daemon. The daemon is developed to be a core service of the underlying guest operating system of each virtual machine. To be precise, it currently supports any Linux-based operating system, regardless if it is virtualized or not. Since majority of the programmatic components is platform-independent (written in Java programming language) the support of other platforms is planned in the future.

We describe core principles and components and properly advocate their importance with regard to the requirements proposed in previous section. For illustration purposes we follow the monitoring information flow from sensor to remote consumer as depicted on Figure 2.

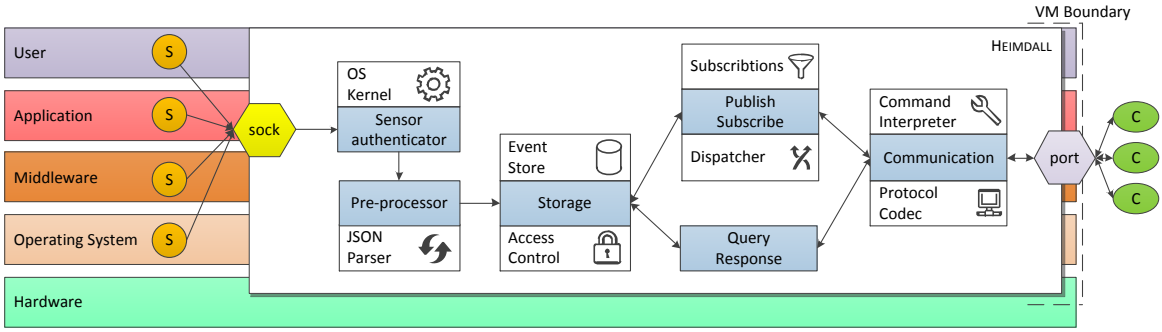

Fig. 2. Architectural overview of our solution, the monitoring data flow from left to right

A. Event-based Representation

To reduce diversity in monitoring information representation and collection all types of monitoring information are represented as event objects. Regardless if its measurement, log, notification, business activity or any other arbitrary type, the information is always represented as a typed data object with defined structure. The actual data format to encode this object is discussed in the following sub-section. As all the information is represented in the same way the sensors are able to publish the events into a single collection point (in our case it is UNIX domain socket at virtual file-system location `/dev/events`). Henceforth the terms monitoring information, event object and event are used interchangeably.

B. JSON Data Format with Extensible Schema

```
{'Event':{
  'id':16051986,
  'occurrenceTime':'2012-04-11T08:25:13.129Z',
  'hostname':'lykomedes.fi.muni.cz',
  'type':'org.apache.httpd.request.GET',
  'application':'Apache Server',
  'process':'httpd',
  'processId':4219,
  'severity':1,
  'http://httpd.apache.org/v2.4/events.jsch':{
    'resource':'/apache_pb.gif',
    'protocol':'HTTP/1.0'
    'response':200
  }
}}
```

Listing 1. Monitoring information encoded as JSON event object

To conform to the requirements defined in section III, JSON was identified as best suitable format for encoding monitoring information in the form of event objects. JSON was chosen over widely used XML in order to achieve best performance and low intrusiveness. As demonstrated in [30] and [24] XML has a larger overhead (up to 50%) and is more computationally expensive to process than JSON. For the sake of compactness the JSON object can be further collapsed into one line, similarly to the conventional plain-text log entries, still maintaining its readability.

In our prototype event object represented in JSON uses two schemes, fixed schema, common for all event objects, and custom schema specific to a particular event type. Listing 1 represents event object encoded in such a way. The custom schema can be defined freely, based on the needs of the particular application or sensor. It is identified by unique id and its version but it is not part of the event object thus it must be retrieved additionally in the form of JSON-schema. For simplicity reasons there is no external schema repository used. Instead, every producer store all the custom schemes for the event types it produces. Therefore, consumer can retrieve the schema on demand and store it locally for later use.

Note, that with only minor changes the event objects can be encoded in a way, that conforms to the CEE Log Syntax Specification [4] since JSON is one of the supported encodings very similar to ours.

C. Sensor Authentication

To assure the origin of monitoring information the publishing sensors are authenticated. Each process that writes to UNIX domain socket introduced in subsection IV-A is identified by its process ID, user ID, and group ID (also known as `SCM_CREDENTIALS`) provided by the socket itself and verified by OS kernel. If the malicious behavior is detected the monitoring data are discarded and security incident is reported.

Therefore as long as the OS kernel is not compromised, the potential attacker cannot insert malicious monitoring information by impersonating particular sensor or performing man-in-the-middle attack. For example, the attacker cannot impersonate Apache HTTP Server and counterfeit the monitoring information in such a way that would falsely indicate that the server is down.

D. Efficient Storage

After being published by the sensor, authenticated, and parsed, the event objects are stored in a lightweight key/value pair database based on Berkeley DB we refer to as event store. Key/value store is used in order to achieve high write throughput while maintaining small memory footprint.

To avoid complexity, the data are stored in JSON-compatible binary data format SMILE [5] designed for space

and serialization/deserialization efficiency. Each record is indexed by fixed schema attributes to enable fast and efficient retrieval. To avoid unnecessary reads the most recent records are cached, ready to be pushed to the consumers. To assure data integrity, once the records are stored they are write-protected; additionally, to prevent an unauthorized access the event store is fully encrypted using symmetric cryptography (Rijndael/AES algorithm using 128-bit key) natively supported by Berkeley DB. The data are protected as long as the integrity of the OS kernel is not compromised (i.e. the attacker is not able to read system memory). Both above-mentioned features are important steps towards multi-tenancy.

E. Fine-Grained Access Control

Since the records in the event store are write-protected, the access control is narrowed down to ensuring that the event objects can be read only by authorized users. Hence, there is no need to introduce capability-based model, as a simple access control list (ACL) is sufficient. As we have already emphasized, the access control granularity in a multi-tenant Cloud environment must be in some cases as small as permitting/restricting access to events produced by one application or process, therefore, the ACL implementation must support it. In its current state Heimdall uses a very basic ACL implementation using permit/deny keywords and wild-card masks.

F. Query Evaluator

Following the query/response interaction pattern the query evaluator component allows for batch-oriented transfers of monitoring information. Moreover, it can be in fact used to implement the notification interaction pattern by automatically querying all the available monitoring information after consumer connection. Additionally, it enables local user to query event store via command line. From our point of view, the querying component poses rather implementation problem than research one, therefore, our focus is concentrated on publish/subscribe interaction pattern and related challenges.

G. Publish/Subscribe Subsystem

To enable consumers to further specify the monitoring information that they are interested in, publish/subscribe [1], [18] component is part of the architecture. Generally, the subscriptions can be either static, i.e. they are defined locally by an administrator of the virtual machine, or dynamic, i.e. the consumer can subscribe and unsubscribe for the monitoring information dynamically. Presently, only static subscriptions per user are supported using a syntax very similar to that used for the ACL. The particular dynamic mechanism and syntax for specifying events of interest is yet to be carefully researched and evaluated.

H. Communication

To enable remote interaction between producer and consumer of monitoring information we designed experimental frame-based protocol over TCP layer. A byte-encoded frame

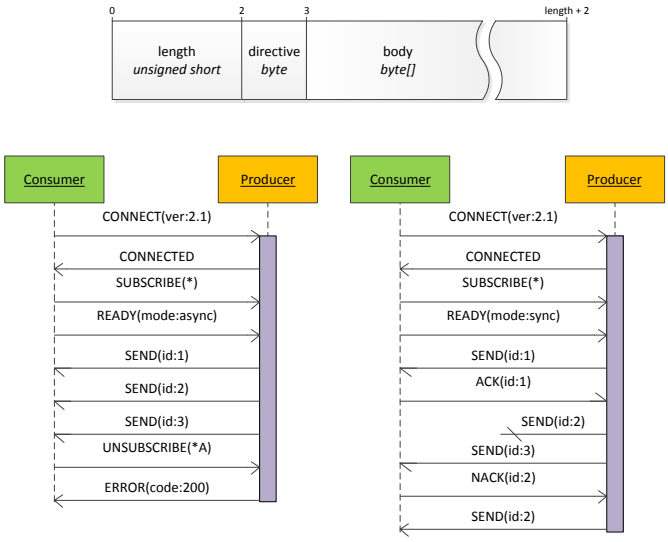

Fig. 3. Example of asynchronous and synchronous transfer of monitoring information using frame-based connection-oriented protocol

consists of a length, directive (command) and body. After handshake the consumer is ready to receive events it previously subscribed for. Each individual event object is then sent in a body of a single frame being encoded either as JSON event object or SMILE binary object, depending on the consumer's preference. To ensure security of the transfered event the solution is prepared to support SSL encryption of the communication.

The communication component of the solution handles concurrent connections from multiple users (tenants). Another important responsibility of the component is transfer reliability, it supports asynchronous emission of event objects as well as ACK-based synchronous transfer as depicted at Figure 3. And finally, it serves as a command interpreter for accompanying communications protocol. The complete list of commands includes: `CONNECT`, `SEND`, `SUBSCRIBE`, `UNSUBSCRIBE`, `READY`, `STOP`, `ACK`, `NACK`, `ERROR`, `GET`, `DISCONNECT`.

V. Summary / Evaluation

Presented solution aims to be lightweight, extensible and interoperable, i.e. an alternative (or complement) to Syslog, as opposed to proprietary solutions such as Hyperic Sigar, Nagios and Munin with monitoring information producer in the form of agent deployed on the targeted system. The common features of such agent-based producers include:

- Sensors have the form of proprietary plugins or extensions compatible only with the paticular solution
- Use of query/response interaction pattern (pull model) with central querying consumer component
- Primary focus on measurements collection, logging is supported by plugins which however work with unstructured plain-text log entries
- Use of proprietary format for monitoring information representation as well as use of proprietary protocol for the information transfer

- Due to the previous fact, monitoring information is hard to be consumed and processed in an extensible and interoperable manner
- Multi-tenancy support goes only as far as distribution of monitoring data to multiple predefined servers/consumers
- Access control is limited to agent authentication per monitored virtual machine

In the following paragraphs we further analyze our approach and demonstrate that it conforms to the requirements presented in section III. To shortly summarize the strong points of the solution, virtual machine running the daemon is able to:

- Produce structured and self-describing monitoring information in JSON format
- Allow robust network data transfer via lightweight communications protocol
- Provide single access point to monitoring information both locally and remotely
- Enable multiple tenants to access monitoring information simultaneously
- Provide fine-grained access control over monitoring information and ensure overall data security

A. Multi-tenancy

As presented in subsection III-A multi-tenant monitoring can be achieved by satisfying 4 basic requirements, i.e. concurrency, isolation, integrity and proof of origin. In the case of our prototype *(1) concurrency* is implemented on the side of the communication component. Multiple users are able to connect to the virtual machine and consume desired monitoring information simultaneously by using experimental connection-oriented communication protocol. Since it is desirable to limit access to some monitoring information (e.g. business activity monitoring data) the solution uses access control to assure *(2) isolation*. To further prevent unauthorized access the data store is encrypted. Moreover, once it is stored the monitoring information is write-protected to ensure its *(3) integrity*. Finally, each sensor that generates monitoring data is authenticated using information provided by OS kernel. Therefore, as long as the integrity of the OS kernel is not compromised the *(4) origin* of the monitoring data is non-repudiable.

B. Unified Representation

The event-based representation of monitoring information directly impacts all the stages of the monitoring process. Since all the types of the monitoring information are represented identically they can be collected, stored, processed and distributed identically which leads to simplicity of the respective prototype components as well as the consumer of such information. Moreover in the case of processing the event-based representation allows for advanced data correlation (e.g. correlating CPU load with application server log using time windows) and pattern detection.

C. Extensible Data Format

Use of schema-based JSON clearly satisfies all the requirements for extensible and interoperable data format of monitoring information, i.e. standardization, self-description, extensible schema, structuredness and compactness. When using such data format the developers of sensors and logging components are able to enrich the fixed schema with domain-specific and application-specific information (i.e custom schema) and thus as a result specify monitoring information that can be directly processed without any additional steps (e.g. transformation, abstraction, semantic analysis etc.).

D. Delivery Channel

The consumer interacts with producer of monitoring information (virtual machine) via TCP-based protocol. After connection to a particular virtual machine using its hostname and configured (standard) port (e.g. `lykomedes.fi.muni.cz:6000`) the consumer is able to receive all the monitoring information permitted by the access control. The number of connected consumers is limited only by the underlying guest operating system limits. It is only natural that the consumers can connect to multiple producers at once. The publish/subscribe and query components of our prototype allow for different types of interactions as defined by GMA. Furthermore, depending on the specific scenario, the consumers are able to receive monitoring information both synchronously and asynchronously.

VI. Conclusion and Future Work

In this paper we focused on a virtual machine as the producer of monitoring information, and introduced multi-tenancy, unified representation of monitoring data using extensible format and standard delivery channel as requirements for such producer. Virtual machine monitoring daemon conforming to these requirements was presented introducing several core principles including JSON event objects, fine-grained access control, publish/subscribe subsystem and experimental communication protocol. Presented approach can lead towards interoperable monitoring, i.e. multiple users can access monitoring information from all the 5 top layers of Cloud 7-layer model, regardless of Cloud provider, virtualization vendor and used middle-ware.

Such scenario opens new research areas related to processing and correlating huge amounts of monitoring data, especially if produced from different Clouds, this is sometimes referred to as *inter-cloud monitoring*. A very promising approach is connected to the emergence of Complex Event Processing (CEP) techniques and algorithms. The use of CEP engines for advanced pattern detection and correlation of monitoring information is natural approach since the continuously produced data are structured with clearly defined schema. We are also interested in the possibility of embedding an extremely lightweight CEP engine into the producer (i.e. the monitoring daemon). With such functionality the producer would be able to correlate, filter and aggregate event objects as early as possible and thus increase scalability and decrease network overhead of the resulting monitoring solution.

REFERENCES

[1] Marcos K. Aguilera, Robert E. Strom, Daniel C. Sturman, Mark Astley, and Tushar D. Chandra. Matching events in a content-based subscription system. In *Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing*, PODC '99, New York, NY, USA, 1999. ACM.

[2] Cloud Security Alliance. Security guidance for critical areas of focus in cloud computing. 2011.

[3] The CEE Board. Cee architecture overview specification v1.0-alpha. Feb. 2012.

[4] The CEE Board. Cee log syntax (cls) specification v1.0-alpha. *Online: http://cee.mitre.org/docs/cls.html*, Feb. 2012.

[5] LLC FasterXML. Efficient json-compatible binary format: Smile. *Online: http://wiki.fasterxml.com/SmileFormatSpec*, 2011.

[6] Open Grid Forum. Open cloud computing interface - OCCI. *Online: http://occi-wg.org/*, 2011.

[7] I. Foster, Yong Zhao, I. Raicu, and S. Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE '08*, nov. 2008.

[8] Bernd Grobauer and Thomas Schreck. Towards incident handling in the cloud: challenges and approaches. In *Proceedings of the 2010 ACM workshop on Cloud computing security workshop*, CCSW '10, New York, NY, USA, 2010. ACM.

[9] A. Group. Delta cloud: Many clouds. one api. no problem. *Online: http://incubator.apache.org/deltacloud/index.html*, 2010.

[10] P. Hasselmeyer and N. d'Heureuse. Towards holistic multi-tenant monitoring for virtual data centers. In *Network Operations and Management Symposium Workshops (NOMS Wksps), 2010 IEEE/IFIP*, april 2010.

[11] Fujitsu Research Institute. Personal data in the cloud: A global survey of consumer attitudes. 2010.

[12] Jclouds. Jclouds: multi-cloud library. *Online: http://code.google.com/p/jclouds/*, 2012.

[13] Katarzyna Keahey, Mauricio Tsugawa, Andrea Matsunaga, and Jose Fortes. Sky computing. *IEEE Internet Computing*, 13, 2009.

[14] Ryan K.L. Ko, BuSung Lee, and Siani Pearson. Towards achieving accountability, auditability and trust in cloud computing. In *Advances in Computing and Communications*, volume 193 of *Communications in Computer and Information Science*. Springer Berlin Heidelberg, 2011.

[15] Adetokunbo Makanju, A. Nur Zincir-Heywood, and Evangelos E. Milios. Storage and retrieval of system log events using a structured schema based on message type transformation. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC '11, New York, NY, USA, 2011. ACM.

[16] M. Mansouri-Samani. *Monitoring of Distributed Systems*. University of London, 1995.

[17] Matthew L Massie, Brent N Chun, and David E Culler. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7), 2004.

[18] T.R. Mayer, L. Brunie, D. Coquil, and H. Kosch. Evaluating the robustness of publish/subscribe systems. In *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2011 International Conference on*, oct. 2011.

[19] Peter Mell and Tim Grance. The nist definition of cloud computing. *National Institute of Standards and Technology*, 53(6), 2009.

[20] M. Nagappan and M.A. Vouk. Abstracting log lines to log event types for mining software system logs. In *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*, may 2010.

[21] Laboratory of Software Architectures and Information Systems. Heimdall – event-based monitoring daemon. *Online: http://lasaris.fi.muni.cz/research-projects/monitoring*, May 2012.

[22] Dana Petcu, Ciprian Crăciun, Marian Neagul, Silviu Panica, Beniamino Di Martino, Salvatore Venticinque, Massimiliano Rak, and Rocco Aversa. Architecturing a sky computing platform. In Michel Cezon and Yaron Wolfsthal, editors, *Towards a Service-Based Internet. ServiceWave 2010 Workshops*, volume 6569 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2011. 10.1007/978-3-642-22760-81.

[23] Dana Petcu, Georgiana Macariu, Silviu Panica, and Ciprian Crăciun. Portable cloud applications – from theory to practice. *Future Generation Computer Systems*, 2012.

[24] Carlos Rodrigues, José Afonso, and Paulo Tomé. Mobile application webservice performance analysis: Restful services with json and xml. In *ENTERprise Information Systems*, volume 220 of *Communications in Computer and Information Science*. Springer Berlin Heidelberg, 2011.

[25] Jin Shao, Hao Wei, Qianxiang Wang, and Hong Mei. A runtime model based monitoring approach for cloud. In *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing*, CLOUD '10, Washington, DC, USA, 2010. IEEE Computer Society.

[26] J. Spring. Monitoring cloud computing by layer, part 1. *Security Privacy, IEEE*, 9(2), 2011.

[27] J. Spring. Monitoring cloud computing by layer, part 2. *Security Privacy, IEEE*, 9(3), 2011.

[28] Jon Stearley, Sophia Corwell, and Ken Lord. Bridging the gaps: joining information sources with splunk. In *Proceedings of the 2010 workshop on Managing systems via log analysis and machine learning techniques*, SLAML'10, Berkeley, CA, USA, 2010. USENIX Association.

[29] B Tierney, R Aydt, D Gunter, W Smith, and M Swany. A grid monitoring architecture. 2002.

[30] Peng Wang, Xiaodong Wu, and Huamin Yang. Analysis of the efficiency of data transmission format based on ajax applications. In *Information Technology, Computer Engineering and Management Sciences (ICM), 2011 International Conference on*, volume 4, sept. 2011.

[31] Serafeim Zanikolas and Rizos Sakellariou. A taxonomy of grid monitoring systems. *Future Gener. Comput. Syst.*, 21(1), January 2005.