Monitoring Virtualized Infrastructure in the Context of Grid Job Execution

Jiří Sitera, Zdeněk Šustr, Boris Parák, and Daniel Kouřil
Grid Department – MetaCentrum
CESNET z. s. p. o.
Zikova 4, Prague, 160 00, Czech Republic
Email: emi-lb@metacentrum.cz

***Abstract*—This paper describes a new direction in the development of the Logging and Bookkeeping service, a gLite component tracking job life cycles in high performance computing grids. From its early days, Logging and Bookkeeping is used to track not only jobs themselves, but also the wider details of the job execution environment. Since a large portion of the grid infrastructure is now virtualized, the work at hand concerns tracking the virtualized nature of that runtime environment. With virtualization and cloud technologies being highly flexible and dynamic, the authors believe that it is very important to gather and keep status information for virtual machines used to run the workload. A newly defined monitoring entity – a virtual machine – is integrated with job state information and provides an enhanced view of the current state and history of both the computing job and the underlying infrastructure, as well as their mutual relationship. This paper explains the motivation and discusses the architecture of the newly emerged solution for monitoring virtualized resources – uniquely – in the same context as the workload they are processing.**

***Keywords*-grid; cloud; virtualization; monitoring; relationship**

I. Introduction

This article is an updated and extended version of a work-in-progress report published in September 2012 at the INFOCOMP Conference [1].

Logging and Bookkeeping (LB), part of the gLite grid middleware stack, is a monitoring tool equipped for monitoring the states of all kinds of processes related to grid computing [2]. Besides traditional gLite Workload Management System (WMS) [3] jobs (often refered to simply as “gLite jobs”) and logical groupings thereof such as direct oriented graphs (DAGs) or collections, it also monitors input/output data transfers or the states of computing tasks submitted directly to a local resource manager – the CREAM Computing Element (also part of the gLite middleware stack) [4] or TORQUE (Terascale Open-source Resource and QUEue manager) [5].

It collects event information from various grid elements and sums it up to determine the current status of any such process at the given moment. It is designed to accept additional state diagram implementations to support other types of processes as required, relying on essential common features such as reliable event delivery (based either on LB’s own legacy messaging layer or standard STOMP/OpenWire messaging), or LB’s querying interface. LB is highly security-oriented and has proved itself in WLCG (Worldwide LHC Computing Grid) operations. It is widely deployed across the European Grid Infrastructure.

This article explains how the grid monitoring tool is applied to monitoring the grid’s underlying virtualized infrastructure. Section II clarifies what the requirements are and why LB is deemed suitable for monitoring virtualized resources as provided by PaaS (Platform as a Service) clouds. Section III outlines the solution designed and implemented to deliver not only the essential functionality but also to support complex real-world use cases, while Section IV discusses additional issues to consider and focus on in the future. Finally, Section V gives evaluation and results of the work, and Section VI sums up the outcome.

II. Motivation to Include Virtual Machines in the LB Model

Using LB in monitoring virtualized resources is inspired by obvious similarities with the existing processes, backed by explicit requirements from infrastructure operators.

A. Virtual Machine as a Job

LB’s main objective is to know everything about the scheduling and execution of computing jobs, not only to respond to “current status” queries, but also to make it possible to analyze the behavior of the infrastructure (failing components, misconfiguration) and possibly even provide certain job provenance capability (ensuring repeatability of jobs/experiments, storing computing environment characteristics and configuration, the description of the compute job provided on submission, etc. – in short, serving as lab notes for “in silico” experiments).

In contemporary grids and other computing infrastructures, machines running computing jobs are themselves dynamic entities following a life cycle similar to that of the grid job itself. It is not unreasonable to expect further blending of cloud and grid models where grid components run either in a cloud (StratusLab [6]), or in a mix with cloud services (MetaCentrum [7], WNoDeS [8]).

All things considered, tracking virtual machines (VMs) throughout their life cycle in contemporary grids is as important as tracking jobs. Moreover, there is an added value to tracking those two kinds of entities in a common manner! Not only does it provide for a better understanding of mutual

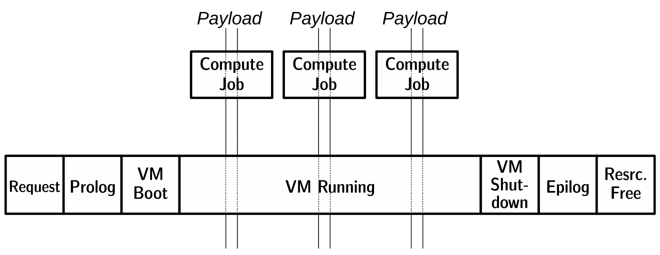

Figure 1. Viewing compute jobs as workload executing over a VM.

relationships and dependencies, but also for a unified view for users and administrators.

Figure 1 shows a simplified and illustrative example of the desired higher-level view of the infrastructure state. It maps compute jobs to the underlying VM life cycle and provides the user with a comprehensive overview of its current state and possible problems. In the case of highly dynamic virtualized infrastructure it can be used to assess efficiency and induced tradeoffs. Data collected in this manner can also be used to produce higher-level statistics and monitoring (mapping actual hardware resources to computing jobs), while the low-level information is still available for detailed inspection if required for debugging or any other kind of ex-post analysis. There are, for instance, scientific projects such as the Grid Observatory, which rely on data provided by LB for behavioral analysis of processes within the grid.

The following LB features are considered most useful in terms of (re)usability for virtual machines:

- Recording primary events and using a state machine specific to the given type of process (job, VM, file transfer, etc.) to combine all information contained therein and determine the current state of that process.
- Providing the ability to get processes grouped or annotated/tagged by the infrastructure, administrators, or users.
- Architecture and implementation based on standards (messaging, authentication and authorization infrastructure, web services), allowing simple event gathering in a reliable and secure way.
- Essential functions (logging events, querying for basic information) provided not only by library functions with bindings for multiple programming languages, but also by command line tools allowing for simple scripting.

There is another key factor – a non-technical one. LB is currently widely deployed across the European Grid Infrastructure. It is not an emerging solution that has yet to be put to production. The support for monitoring underlying infrastructure layers is an evolution of an established and broken in service, making it also a relatively costless solution to the presented problem.

B. Features Requested by the Czech NGI

MetaCentrum, the Czech National Grid Initiative (NGI), is designed as a mixed cloud/grid service, where resources from a single, consistently managed pool can be provided either as traditional batch system-managed resources or as VMs, depending on current user needs [9]. The scheduler (TORQUE) can handle three types of requests:

1) Run a job
2) Run a job in a selected VM image
3) Run a VM

What follows is a summary of feature requests made by the NGI before work presented in this article started. How they were addressed is explained in Section III “Designs and Implementation”.

- The desired functionality should provide a uniform, consistent view of the infrastructure, mapping all user requests to actual hardware.
- It should replace currently used data mining tools providing status feeds to the MetaCentrum portal and to the long-term usage statistics processor.
- Since MetaCentrum is also involved in research of batch system scheduling strategies, gathering data relevant to this kind of assessment is another requirement.
- Yet another requirement, albeit one that is already fulfilled by LB’s design, calls for an ability to aggregate information from diverse sources (scheduler, virtualization hypervisor, accounting) and even manually triggered state transitions (for instance putting resources in, and taking them out of maintenance).

C. Related Work

1) Regular Infrastructure Monitoring Tools: Tools such as *Nagios*/*Icinga* or *Ganglia* focus primarily on the “running” state of the given process. Unlike them, this work is not intended to monitor infrastructure health and react to problems, but focuses primarily on understanding the current state of the workload and reporting to users.

Admittedly there is a minor overlap because even LB can be used to provide certain details of infrastructure health. It is namely the LB statistics feature (job/VM status statistics), which, the authors believe, will be further improved by understanding the relationship between the workload and VM layers.

It is, however, important to note that using the above-mentioned monitoring tools to monitor highly volatile short-lived VM instances set up on demand is on the edge of practicality since – looking for instance at *Nagios*, albeit equipped with a selection of plugins – it involves non-trivial, almost continuous reconfiguration of the service not only with machine details but also with access control data. Integrating machine status info with up-to-date references to workload, or vice-versa, would be another challenge, requiring solutions to be designed and implemented on both ends.

2) FSM Implementations: The idea to use a Finite State Machine (FSM) [10], [11] for monitoring purposes is not new. Its typical usage is to monitor software or behavior of network protocols via FSM designed to detect incorrect states or excessive/wrong timing of states. The LB concept is close to existing works describing the application of FSM to monitoring networks of connected components [12], but it is not the same, mostly because LB focuses primarily on workload status, and the behavior of components is a secondary objective.

3) Public Cloud Infrastructures: A typical example of a public cloud infrastructure, the Amazon Web Services (AWS), deploys a complex monitoring tool called Amazon CloudWatch. It is an important brick in the AWS service portfolio [13], allowing not only for monitoring, but also for performing actions (the auto scaling feature). The CloudWatch architecture is based on a central metrics repository fed by resource monitoring tools and providing a common standard API to clients. It also supports user-defined metrics with custom data feeds. The client API has an ability to send notifications and trigger actions when certain thresholds are reached. This service is in many points similar to our proposed solution, but does not employ an FSM.

4) Status Reporting Tools in Scientific Cloud Infrastructures: Each grid infrastructure or cloud management tool has its own way (command line interface, portal) of providing users with the current job or VM status. It is typically implemented as a function of a computing element in a grid or cloud manager (like the Virtual Machine Manager in the case of OpenNebula). Indeed, the LB service is originally one of those tools, monitoring and reporting the status of computing jobs in gLite-based grids.

There is also standardization work in the area of cloud computing, such as the OCCI standard [14] and its implementations, aimed at enabling standard interfaces for reading information off different cloud managers (and even controlling them). However, they do not deal in any way with information from within the virtual appliances, losing a potentially valuable source of information; what is more, they are in no way applicable at the level of workload (grid job) monitoring, and neither is there a separate activity addressing the area of workload status monitoring in a similar way.

To sum up, the authors are not aware of any other work addressing the crucial point – combining available information from different infrastructure layers into a uniform, higher-level, workload-centric view.

III. Designs and Implementation

The proposed solution has been implemented in progressive steps, starting with a pilot implementation on MetaCentrum's OpenNebula instance, running and keeping track of VMs and scheduled TORQUE jobs at the same time. OpenNebula was chosen for the pilot implementation only because it was the cloud manager of choice for the Czech NGI, and the implementation team already possessed adequate expertise for its instrumentation. Apart from that, the solution was in no way tailored specifically to OpenNebula.

This work – i.e., the pilot implementation – was presented and demonstrated at the EGI Technical Forum 2012 [15] and forms the basis of the solution described below.

Next, the work focused on implementing a production-grade solution for MetaCentrum operations (among others instrumenting MetaCentrum's in-house VM manager – Magrathea) and support for federated cloud environments, bringing in additional sources of information external to the batch system and the virtualization stack (administrative operations, information system).

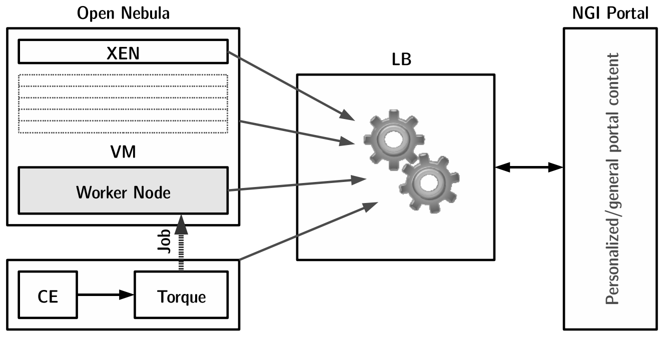

Figure 2. Architecture and components.

A. Architecture

In the basic architecture used for the pilot, VM life cycle was controlled by the OpenNebula cloud computing toolkit, managed manually by administrators, while jobs were assigned to VMs by a standard grid computing element through an instance of TORQUE. All job-related functionality was already in place (LB-aware TORQUE, [16]). The following sources of events were used to govern the VM life cycle:

- *OpenNebula* – providing hooks for call-out scripts activated on any relevant state change (described in more detail in Subsection III-E)
- *Hypervisor, (specifically Xen)* – generating events showing the current VM state and parameters at hypervisor level
- *Hosted worker nodes* – Operating System running the Worker Node was instrumented (init scripts) to provide independent information from the running VM

LB plays the key role in this concept by combining events from all the above components into one higher-level view. It makes the system more precise and robust, which has been well tested in the context of gLite job monitoring. Obviously on certain occasions, all three sources generate almost identical, i.e., seemingly redundant events. But there is undisputed value even in receiving almost identical events multiple times. It improves reliability, and the comparison between the three events provides for fine-grained job status tracking and simplifies troubleshooting. Besides that, different sources often provide values for different attributes, which are unknown to the others.

Basic system architecture is shown in Figure 2. In that design, the only new feature that had to be implemented was VM instance support in LB, comprising of the VM job type definition, introduction of VM-specific events, and a VM state machine.

Only basic attributes are defined in the VM type to cover the expected virtual machine properties, such as owner identification, memory sizes, or the network status of the VM, detailing the host/domain name, and the type of network connectivity (VLAN, private vs. public). Aside of attributes carried by the VM instance, there is also a solution to keep record of its relationship to other entities actively tracked by LB. That could not be implemented as a simple data structure attribute, and the solution devised for that purpose is discussed in greater depth in Subsection III-C.

The existing set of VM attributes, however, is not necessarily final. LB allows any kind of additional attribute to be simply stored with the instance's status (functionality referred to as "User Tags") with only slight limitations. One cannot, for instance, use relations such as "greater than" or "lower than" when querying for instances by User Tag, since LB does not know the type of that attribute and cannot decide. The only comparison supported is string (in)equivalence.

Each VM instance is identified by a string constructed in the same manner as Job IDs currently used in LB, consisting of the LB server's identification, a short literal denoting the process type, and a random unique string. The randomized unique part of the identifier can be supplied by the registering component if required. In that case, it will only be checked by LB for uniqueness, then accepted. While domain names are not suitable for use as identifiers since they are often recycled (reused by another instance) or even used in alteration by multiple interchangeable VM instances, internal VM identifiers used for instance by OpenNebula (or any other virtualization stack for that matter) are unique to their given OpenNebula server, and can be easily used in the compound ID with the added benefit that one can tell the responsible LB server, the virtualization server and the internal identifier, all at a single glance. Thus, instead of using an LB-generated random ID such as

```
https://lb.example.com:9000/
VM:1ch5-QIGMd_xW3oGM-HScg
```

one may choose to supply their own ID with, for instance, the following format, which is much more informative:

```
https://lb.example.com:9000/
VM:nebula1.example.com_12345
```

B. VM State Machine

The VM state machine is shown in Figure 3. It is based on OpenNebula's internal states but modified to be general enough to provide a single, common view for all VM management systems that will be supported in the future. The states describe major changes in the VM life cycle whereas attributes are used to describe the instance's properties in the current state, or even to distinguish "substates."

It is worth stressing at this point that although the proposed state diagram is considered adequate and detailed enough for the intended purpose, it is relatively easy to implement changes, should it prove necessary. These may range from simple changes in state transitions to extending the state diagram with a completely new state or splitting a single state into two or more. Even with real-world operating experience,

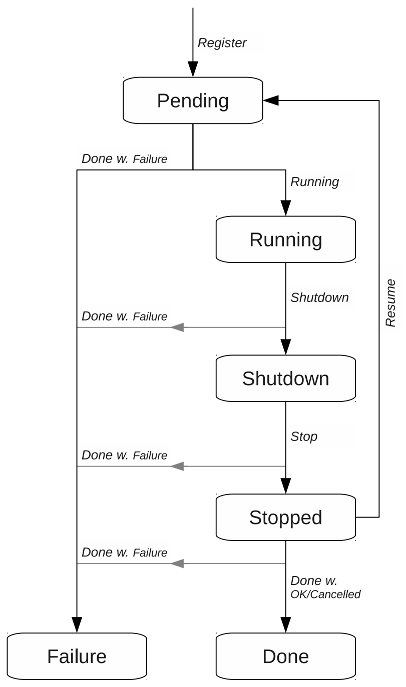

Figure 3. VM state machine.

however, just a single minor change in the mapping of VM states to the generic state diagram was required so far.

Any event received by LB may or may not trigger a change in the state and/or attribute values of an instance. Indeed, some types of events never even attempt to trigger a state change and are only used to bring in new or updated attribute values. (As such, they are not even shown in the state diagram in Figure 3).

As soon as all events received by the current point in time are correctly interpreted, which is an action performed automatically on the arrival of each event, the instance's current state and attributes constitute the most up-to-date set of information as collected from all the various sources mentioned above. LB is designed to overcome obstacles such as events delivered out of sequence, intermediate events not delivered at all, or events received from different sources with clocks skewed in different directions. This is achieved by making the event sorting algorithm rely on arbitrary hierarchical message sequence codes rather than time stamps.

Table I shows how messages from each component/layer (Hypervisor, Cloud Manager, VM Image) contribute to the overall picture of the instance's current state. Since they are all instrumented to produce genuine LB events, the format of

TABLE I. DIVERSE SOURCES OF VM-RELATED EVENTS & ATTRIBUTES

Component	**Events**	**Attributes**
CloudManager	Register, Create, Running, Host detail, Shutdown, Done	Hostname, Physical host name, Owner, Requirements, etc.
VM Image	Really Running, Shutdown	Runtime info, user tags
Hypervisor	Running, Shutdown	Actually assigned resources (CPU, RAM, network)

incoming data is unified right from the start of the delivery chain and the LB server can process incoming data uniformly regardless of the source component.

To allow universal queries to the LB server, VM states (as well as states defined in other state diagrams implemented in LB) are also mapped to states in the default LB state machine used primarily for gLite WMS jobs. Thanks to that mapping, users can easily query LB for, e.g., all their running tasks regardless of whether they are computing jobs, virtual machines, or any other kind of supported process. The complete mapping between VM and gLite job states is shown in Table II.

TABLE II. MAPPING OF VM STATES TO GENERIC GLITE STATES FOR UNIVERSAL QUERIES

VM State	Generic State
Pending	*Submitted* for freshly registered VMs
	Waiting for VMs resumed from *Stopped* state
Running	*Running*
Shutdown	*Waiting*
Stopped	
Done	*Done*, distinguished by the *done_code* attribute
Failure	

C. Relationship of Entities

Limited support for specific inter-job relationships was inherited from previous LB versions. Relationships between different types of jobs were implemented differently, depending on then-existing requirements. Experience with that implementation was taken into account when designing the new solution.

1) Pre-existing Implementations: Only the following job type pairings were supported before the generic solution was designed:

- *Parents/children* in DAGs (Direct Acyclic Graphs) and job collections, where one single parent job is linked to its children and vice versa. This is implemented by an extra database attribute in all children, which explicitly states the ID of the parent. This attribute can be used directly in the LB server's database queries and, being also implicitly indexed, allows for a quick reconstruction of the whole set of children in a collection.
  The relationship is established on registration, wherein status records are created for the parent job as well as all the children in a single server-side step, filling in the parent reference. There is no support for removing jobs from collections, hence the relationship can be never canceled. Neither is there currently support for growing collections already registered.
- *Compute jobs/Sandbox transfers*, where each gLite compute job can maintain a reference to its input and output sandbox transfers. There are specific single-purpose attributes included in the job's internal status structure, one for the input, another one for the output data. They will contain job IDs of single sandbox transfers, or collections thereof. Since the sandbox transfer IDs are encoded in the internal status of the job, they are slow to access unless the LB administrator has set up specific database indices for that purpose. This makes searching for compute jobs by input or output sandbox transfer ID possible but impractical.
  The relationship is established by a special-purpose *sandbox* event that will set or modify the reference, but obviously the job status structure can never refer to multiple input or output sandbox transfer IDs at once. Collections must be used if multiple input or output sandbox transfers are to be covered.

Both features remain, for now, available alongside the generic solution outlined below. While the latter could be adequately replaced with the new approach in the future, the former presents a very specific case with built-in server-side logic (registration, state and histogram algorithms, etc.), and the functionality must remain unchanged.

2) Designing a Generic Bilateral Relationship Record: Facing the task to implement yet another type of relationship – that between a virtual machine and its workload – it was decided to take a generic and more widely applicable approach that could support any kind of bilateral inter-job relationship in a common manner once and for all. The requirements were as follows:

- Support $m : n$ relationships as a VM will definitely relate to multiple compute jobs as well as a compute job can relate to multiple machines (typically in cases where it did not succeed in one and had to be resubmitted to a different resource).
- Allow distinguishing between active and past relationships.
- Avoid adding complexity in server-side processing, i.e., avoid, wherever possible, dependence on additional database queries when registering relationships.
- Support establishing or canceling the relationship at any time during the participating entities' lifetimes.
- Since it was decided to make this a generic functionality, allow for specifying the nature of the relationship, i.e., what does the relationship record mean.

To that end, a new type of event (*relationship*) has been introduced to control the relationship records, and the present SQL schema was extended with an additional table comprising of the following attributes:

- *Source job*: the job the relationship originates from.
- *Target job*: the other job in the relationship. Note that the source/target approach obviously makes the record asymmetric. To achieve symmetry, every *relationship* event actually results in two separate records being created, one of them having the originating job as the source in one instance, and as a target in the other instance.
- *Target job type*. It is assumed that relationship information is never retrieved from the LB server separately, but always together with the status of the

"source" job, which also indicates its type. Therefore, the type of the "source" job does not have to be stored with the relationship record and still the types of both jobs in the relationship are known and can be used to interpret the relationship's meaning. For instance, the existence of a relationship between a compute job and a virtual machine can be easily interpreted in the sense that the job is running on that virtual machine. A relationship between a virtual machine and a sandbox transfer, e.g., will be interpreted in the sense that LB was monitoring the transfer of the virtual machine's image during the prolog stage.
Technically, a relationship can be registered for any exotic combination of two job types. Sometimes the meaning may be obvious, as in the two cases above, and sometimes not. At any rate, interpretation is left to the party who has logged the relationship in the first place.
To minimize server-side overhead, the client is required to specify the *target job type* along with its ID when logging a relationship. This allows the server to register the relationship without having to unparse the other job's status – a relatively costly operation. There is a minor danger that the client will log a wrong job type, but it is felt that keeping the data correct and reliable is always the client's responsibility. Still, implementing an option to read the target job's type from the database regardless of the extra cost is an opportunity for future development.

- *Relationship status*. The status is given as an enumerated type, allowing for future extensions. The initial implementation recognizes the following states:
    - *Active*: in the context of virtual machine/workload monitoring, an *Active* relationship will refer to the job currently running on the machine.
    - *Inactive*: primarily for relationships that are not active anymore. Possibly also for relationships that will become active in the future (for instance a job has been scheduled to a particular resource but has not arrived there yet). Once again, the precise interpretation of the meaning may be up to the party who has logged the relationship.
    - *Canceled*: used instead of removing the relationship record. The physical relationship record is only removed when one of the relevant jobs is being purged from the LB server for good.

Note that a relationship record carries only information valid at present and does not maintain any record of the relationship's history. However, since relationships are established and controlled through events, their history can always be reconstructed by querying for the logging history (i.e., raw events) of jobs participating in the relationship.

D. Security Considerations

The security of data gathered by LB is an important part of the solution. LB implements a messaging infrastructure to deliver events from the place where they are created to the LB server. The infrastructure is built upon a set of mediators (*inter-loggers*) that provide a secure and fault-tolerant transport of messages between LB clients and servers. This also means that the actual worker nodes do not necessarily require direct Internet access since they are sending their events through the interloggers, typically installed only on head nodes with different network accessibility settings.

All connections within the infrastructure are authenticated, including the delivery of a new event. Upon receiving a connection carrying an event, the LB server makes an authorization decision to see if the originating component is entitled to log that kind of event for the given job.

Reading access is likewise subject to similar security precautions. Every client querying the LB server must be properly authenticated and authorized to obtain the data requested. There are actually two ways of setting up access: both are applicable to all processes monitored by LB, including virtual machines:

- *Server-wide authorization policy*: there are several authorization categories to grant rights across all jobs-processes known to the server. The categories cover both the logging and querying parts of data processing. There is a specific category, for instance, to allow logging events specific to workload managers. Only grid components listed in that category can log those specific kinds of events. Similar categories are there for different levels of reading access.
- *Per-job ACLs*: an Access Control List can be maintained for any individual entity (job, VM, etc.), granting additional permissions to read or log events for that entity alone. The ACL is stored as a part of the entity's internal state, maintaining a list of allowed or banned users. Users are identified by DN or Kerberos principal, which can be used interchangeably, and mapped one to another by means of a server-side Globus-compliant *gridmap* file [17].
  Unlike the server-wide policies, ACLs on a job are entirely under the control of the owner of that job. Users can therefore specify fine-grained access control for their jobs.

Per-job ACLs are used in virtual machine monitoring to overcome the fact that OpenNebula currently cannot communicate the assigned ID to the virtual machine it has instantiated. When creating a new virtual machine, the ACL of the corresponding instance in LB is populated with its identity. Using appropriate credentials later on, the virtual machine can look up its appropriate job ID when needed.

In order to simplify the deployment of LB in diverse environments, LB supports multiple security mechanisms. In particular, the de-facto grid standard using X.509 and TLS/SSL has been supported from the very beginning. Support for Kerberos has been added recently. It is, however, implemented in a generic way, so that adding additional security mechanisms is quite feasible.

The LB server also supports mapping of client identities by means of a *gridmap* file so that a single person relying on different clients using different types of credentials obtained through different authentication mechanisms can always be identified as the same user. This "mixed" setup where users use their Kerberos tickets for regular work in the command line, and their X.509 certificates to access LB over the HTTPs interface, is actually used in MetaCentrum where LB was first deployed to monitor the virtualized infrastructure along with the computing jobs.

E. OpenNebula Instrumentation

OpenNebula implements a system of so-called Virtual Machine Hooks. Hooks are programs automatically executed, or *triggered*, when a virtual machine changes its state. This allows us to provide simple and standard modular implementation of OpenNebula instrumentation for LB. An LB hook for OpenNebula has been implemented as a stand-alone script invoked by hooks using supplied configuration files [18].

OpenNebula supports hooks for all the relevant states, shown in Table III. Other hooks are available, ensuring future extensibility of this solution.

Information about the virtual machine is passed to the script in the form of an XML template. Each state change triggers a hook, passing unique virtual machine identifier and its template as arguments to the registered executable.

The instrumentation script is written in the Ruby programming language, which is the language of choice for OpenNebula's modules and extensions. It should be registered for all the relevant virtual machine states mentioned above. The script requires a Base64-encoded virtual machine template and the name of the hook as arguments. Other optional arguments include logging method, log file location for debugging purposes or map file location for dynamic mapping of user identities.

Authentication of the instrumentation script as a trusted LB events source can be performed using both X.509 or Kerberos-based host credentials.

The script itself performs four basic actions:

1) Decodes and parses the virtual machine template.
2) Maps user identities (e.g., user names to their X.509 certificate DN).
3) Performs data transformation: currently computes the overall VM runtime from runtime values on individual hosts.
4) Constructs and sends an event to LB using its native API and local binaries

Events are constructed using ERB, Ruby's templating system. The templates are easily modifiable without an extensive knowledge of the programming language itself.

TABLE III. OPENNEBULA STATES WITH HOOKS

State	Trigger Event
Create	Virtual machine has been submitted by the user
Prolog	OpenNebula's scheduler found an appropriate host and started deployment
Running	Virtual machine is running and ready to accept jobs
Shutdown	Virtual machine is shutting down
Stop	Virtual machine has been stopped (temporarily)
Done	Final state, end of the virtual machine life cycle
Failed	Previously issued action has failed, virtual machine is not available

F. Support for Complex Use Cases

The newly developed solution is not limited to basic interaction with a single instance of OpenNebula, but rather implements all features required for other, more complex scenarios.

1) MetaCentrum: Providing a unified view of resources and job execution for users and administrators, this use case follows requirements described in Section II-B with the addition of LB being used as a part of a distributed implementation of TORQUE, recently deployed in MetaCentrum [16]. Here, LB acts as a service providing users with job status information via a modified *qstat* utility.

MetaCentrum operates its own cloud manager called Magrathea [19]. Its role is to cooperate with the TORQUE batch manager to provide integrated grid and cloud environment running within a single resource pool. To use LB to combine status information from both kinds of workload – virtual machines and batch jobs alike – into one overall picture, support for LB had to be implemented in Magrateha. The instrumentation of Magrathea to send LB events in a manner similar to the OpenNebula case was straightforward. The common VM state diagram was tuned to work correctly with both event sources (OpenNebula and Magrathea) at the same time. Now each MetaCentrum user can use all three worlds (OpenNebula, Magrathea, batch jobs) and see all the respective status information in one common LB service.

2) FedCloud: The FedCloud Task in EGI (European Grid Infrastructure) deals with federation of resources in a cloud environment. Here, the aim is to achieve interoperability of different cloud solutions, running in different administrative domains and on different cloud manager implementations. The role of LB in providing global status info for the whole federated infrastructure is similar to that, which LB plays in gLite-based grids such as WLCG – i.e., monitoring all processes across the infrastructure, regardless of geographical site, ownership or flavor of the underlying technology, as long as it is instrumented to deliver events to LB.

This particular use case requires LB to support multiple hypervisors and cloud managers (equivalent to supporting different job managers – computing element implementations – in grid job monitoring). There is currently support for OpenNebula with XEN, and the work on supporting OpenStack and KVM is underway. It is important to stress that the implementation of such additional support consists solely in client-side work, i.e., in instrumenting the new sources to generate LB events, or possibly finding ways to translate existing streams of outgoing information into LB events. The LB service as such is already prepared for the task.

This use case is also going to rely on the multitude of ways one can access LB from the user perspective. LB can be queried through different interfaces, or configured to become a producer of messages delivered over different channels. Some of the interfaces can be accessed with generic, widely available clients such as Web browsers or RSS readers. Figure 4 shows LB as a message processor capable of receiving a flow of messages, potentially over diverse channels, and making the processed information available either on query, or over another streaming channel.

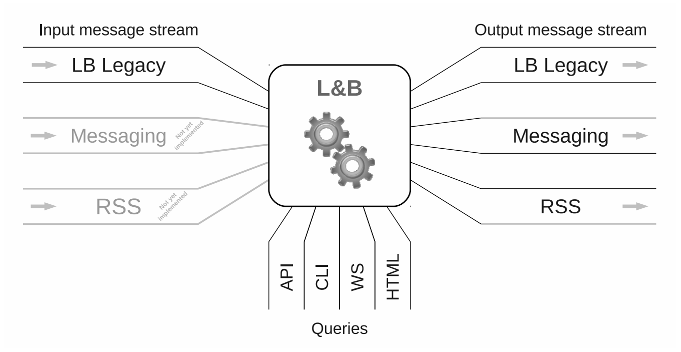

Figure 4. LB as a message processor publishing information over different output channels.

LB has the potential to provide complete infrastructure for common cloud status monitoring, taking care of all the stages from data sources, through data transfer, interpretation and storage, to client access.

3) Support of User Group Workflows: Compared to traditional computing jobs, VMs are a little specific in that they always need to be assigned workload when running (i.e., having started for the first time, recovered from a downtime or finished migration), which makes them actually very similar to pilot jobs! Pilot jobs are simple computing jobs carrying no workload on submission, but rather waiting until they acquire the necessary resources and only then receiving the actual workload, i.e., computing work to be done.

Thus a pilot job framework is a good example of a user-specific workload management system. It is designed to distribute workload to job slots at the moment when pilot jobs (or – analogically – VMs) actually start. It may be very convenient for such a framework to receive notifications of relevant VM status changes, rather than heving to repeatedly poll all relevant resources. That is easily achieved with LB notifications generated on pre-determined conditions and sent out over LB's own legacy messaging chain or through a STOMP or OpenWire-enabled messaging broker.

Users may choose, for instance, to be notified any time any of their machines reaches state *running*. More elaborate sets of conditions are also supported. The resulting notification contains the full VM status information and, if requested on registration, also the full history of events for that machine so far.

Another option to receive updates on VM state changes would be RSS. The RSS interface in LB is as elaborate as the job querying interface, allowing for the creation of highly customized feeds, which can be then received with any RSS reader, provided it can handle X.509 authentication.

G. State Machine for Physical Machines

This is, at the same time, a usage scenario possible with the current implementation, and a consideration for future work.

As per the original design, VM instances use their attributes to refer to their respective physical hosts only by name (the FQDN – Fully qualified domain name, actually) and no track is kept of the actual status of those resources. But there is an obvious similarity between physical and virtual machines. If anything, physical resources are even simpler to describe than virtual ones, and a VM state diagram is easily applicable to physical machines. So the option is to register physical resources as "VM" instances as well, and reference the identifier instead, either by using the ID assigned by LB rather than the FQDN, or by registering a bilateral relationship between the virtual machine and its physical host as described in Subsection III-C.

With that done, the same level of detail can be provided for virtual and physical machines alike, although some supported states can pick up different meanings in the physical world (for instance state *pending* would not mean that the machine is being set up by the virtualization stack, but rather that it is being installed by its administrator) or remain unused altogether. Maintaining detailed status information for physical machines is important because sites need to keep operational logs of physical resources management (maintenance, testing, repair) and understand it correctly in the mixed grid/cloud model (providing proper hardware usage statistics).

Events governing the status of a physical machine record – be it an instance of the VM type used in an "overloaded" mode, or a newly designed Physical Machine type – can be generated:

1) automatically by the machine itself or, more specifically, by its operating system's image, properly instrumented and contextualized:
   - machine start
   - regular machine shutdown
2) automatically by infrastructure monitoring tools:
   - machine down (when unreachable or otherwise recognized as being down)
3) manually by resource administrators
   - machine being installed
   - machine being moved or maintained
   - machine failed (due to a HW issue, for instance)
   - any additional operational records can be logged as *UserTags*

Basically, then, the main distinction lies in the fact that events logged for virtual machines by the Cloud Manager are generated by a human administrator for the physical ones.

There are other benefits stemming from the fact that virtual and physical resources are treated similarly. Figure 1 was showing workload executing over a VM. By including physical resources into the picture, one can also view workload running directly on the physical machine (where allowed by the actual solution – for instance Magrathea or WNoDeS), or watch jobs executing in virtual machines over their physical hosts – see Figure 5.

IV. Future Work

There are several topics identified as a potential improvement or extension of the existing solution. Potentially important as they are, they were already envisioned and accommodated for at design time but the decision on their actual implementation was left for the future.

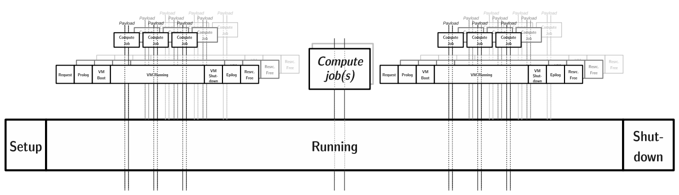

Figure 5. Virtual machines and native jobs executing on a physical machine.

A. Virtual Cluster Implementation

The Virtual Cluster service provided by MetaCentrum can create multiple VM instances per request [7]. All the resulting VMs have common attributes (type of network connection) and are closely related. Similar functionality is provided by other node-on-demand services such as WNoDeS [8].

In most cases, all nodes in a virtual cluster set up in this manner are intended to be used together in close conjunction, usually to run parallel computing jobs, which is why a user can benefit from easily obtaining an overall view of the state of the cluster.

It may be a good idea to reuse the "collections" functionality in LB, typically applied to grid jobs or sandbox transfers. From the user's point of view the state of the collection combines the states of all its members. Individual VM details are still accessible under the VM instance's own ID – the collection functionality simply adds another identifier (collection ID) to access aggregate information for the whole collection, such as child (member) status histograms. The fact that LB's VM type can also be used to monitor physical machines as discussed in section III-G also makes such collections applicable to hybrid clusters where a fixed physical cluster is extended (perhaps temporarily) with additional virtual nodes to improve peak computing power.

There are only minor differences between (already supported) job collections and (proposed) VM collections to address, chief among them the understanding of the overall status of such collections. While a computing job collection can be considered running, for instance, as long as at least one of its children is running, a VM collection should not be considered running unless all the VMs in that collection – or at least a certain majority – are up. Other collection-wide states require similar redefinition. But apart from that, all the essential functionality exists and can be applied to the new collection type.

B. VLAN Status

Virtual Cluster services offered by MetaCentrum provide not only sets of machines but also networking connections in the form of virtual Ethernet (VLAN) [20], thus offering a comprehensive IaaS (Infrastructure as a Service) solution. The VLANs have their own life cycle managed by a purpose-built VLAN manager (SBF) [21], which could become a source of events for LB. After all, a *network* is recognized as an entity in its own right by many cloud-related standards such as OCCI (Open Cloud Computing Interface).

An ability to track the state of the network together with its attributes (private vs. public network, additional services such as tunnels, NAT (Network Address Translation) or firewalls) could be valuable in many scenarios. True, it would require another state machine to be implementeda, but that has become a relatively routine task recently, and would be well justified by an interesting use case.

C. State Diagram Evolution

Although the state diagram explained in Subsection III-B currently seems to meet the intended need, there were suggestions that it should cover additional states used in more elaborate scenarios possible with advanced cloud managers, such as the VM *migrating*, *resizing*, etc. As it is, these states, for instance, are inherently the substates of the *Pending* state, and even with the current implementation they can be distinguished from other such substates by means of user tags, or – with a simple extension of the code – by means of an additional state attribute.

On the other hand, as long as there is a use case to sufficiently justify the need for making these into separate states, implementing an extension to the state diagram is sufficiently straightforward.

V. Results and evaluation

The new experimental version of LB implementing the features described in this paper was evaluated in MetaCentrum pre-production environment for a few months and currently runs in production. All the code is considered well tested, and it is included in the official LB release, starting with version 4.0. On top of that, TORQUE instrumentation code is available with MetaCentrum's TORQUE patches, and OpenNebula instrumentation can be obtained as a set of documented hook scripts [18]. Thus, all tangible results of the work are publically available.

Based on previous work and the new experience with a production LB instance gathering all information about jobs and virtual machines running in MetaCentrum, the following results can be summed up and evaluated:

1) Performance: Since the whole event delivery chain is ansynchronous, the overhead on infrastructure components is negligible. The authors have previously performed extensive measurements, showing that the processing capacity for a real-world spread of jobs amounted to several hundred thousand jobs per day [22], and the declared target of a throughput of 1 million jobs per day was achieved at least for simple jobs.

Table IV compares known performance limits to measured production load extrapolated from eight months of production use in MetaCentrum, a mid-sized grid infrastructure. It shows that a typical combined grid/cloud operation uses up only a small portion of the possible throughput, making the actual overhead negligible.

TABLE IV. Comparing real-world load in the Czech national grid with the maximum capacity

Item	Prod. per Year	Maximum Capacity
Computing jobs, mixed complexity	1,550,000	$\sim$90,000,000
Virtual machines	25,000	$>$90,000,000
Individual Events	500,000,000	$>$1,550,000,000

2) Applicability of Results: The common monitoring solution succesfully collects and, most importantly, correlates workload and infrastructure status data. Evaluation of the amount and structure of data collected and inferred by LB shows that it indeed provides a detailed, unified view of the multi-layered virtualized environment, and that its output can be used not only to check current status, but also to feed higher-level statistics and reporting systems.

An internal feasibility pre-study performed at the Czech NGI, for instance, shows that its current statistics gathering mechanisms could be replaced with an LB-based solution, as long as LB provides a mechanism to monitor physical machines as discussed in Section III-G, with the added benefit that it wolud completely cover also the emerging PaaS and IaaS cloud services – a task that the existing statistics gathering solution cannot perform.

VI. Conclusion

This paper shows how the potential of an existing job-monitoring infrastructure can be reused in the virtualized world. The design and implementation of LB was extended to support virtual machines as a new kind of monitored entity, and the new functionality was demonstrated in real-word usage. LB can now handle mutual relationships between various entities involved in a modern computing environment (dynamic sets of virtual machines available directly to users, and potentially hosting jobs managed by a grid service). Although this work was, at its beginning, primarily driven by the Czech NGI's requirements, it was found useful at a much wider scope. Typical motivations involve resource federation in the cloud-oriented world. The authors are proposing this solution to FedCloud, the cloud interoperability task force acting within the European Grid Infrastructure. LB, with its established presence in gLite-enabled grid sites across the European Grid Infrastructure, resulting in easy adoption, can be a reasonable candidate for a monitoring and notification service in emerging international "cloud-like" scientific environments.

Acknowledgement

This work is part of the National Grid Infrastructure MetaCentrum, provided under the programme "Projects of Large Infrastructure for Research, Development, and Innovations" (LM2010005).

Fundamental development and maintenance of the Logging and Bookkeeping service was co-funded by the European Commission as part of the EMI project under Grant Agreement INFSO-RI-261611.

References

[1] Z. Šustr and J. Sitera, "Understanding virtualized infrastructure in grid job monitoring" in *INFOCOMP 2012, The Second International Conference on Advanced Communications and Computation*, Venice, Italy, pp. 167 – 170, 2012.

[2] "Logging and bookkeeping," 2008. [Online] Available: http://egee.cesnet.cz/en/JRA1/LB/. [Accessed December 20, 2013].

[3] M. Cecchi et al., "The gLite workload management system," *J. Phys.: Conf. Ser.*, vol. 219, 2010.

[4] P. Andreetto et al., "Status and developments of the CREAM computing element service," *J. Phys.: Conf. Ser.*, vol. 331, 2011.

[5] G. Staples, "TORQUE resource manager," in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing (SC)*, 2006.

[6] "StratusLab," 2012. [Online] Available: http://stratuslab.eu/. [Accessed: December 20, 2013].

[7] M. Ruda et al., "Virtual clusters as a new service of MetaCentrum, the Czech NGI," CESNET, 2009. [Online] Available: http://www.cesnet.cz/doc/techzpravy/2009/virtual-clusters-metacentrum/. [Accessed: December 20, 2013].

[8] D. Salomoni et al., "WNoDeS, a tool for integrated grid and cloud access and computing farm virtualization," *J. Phys.: Conf. Ser.* 331 052017, 2011.

[9] J. Sitera, M. Ruda, P. Holub, D. Antoš, and L. Matyska, "MetaCentrum virtualization – use cases," CESNET, 2010. [Online] Available: http://www.cesnet.cz/doc/techzpravy/2010/metacentrum-virtualization-use-cases/. [Accessed: December 20, 2013].

[10] J. E. Savage, "Models of computation: exploring the power of computing," Addison-Wesley Pub, 1998.

[11] F. Wagner, R. Schmuki, T. Wagner, and P. Wolstenholme, "Modeling software with finite state machines: a practical approach," Auerbach Publications, 2006.

[12] B. Birregah, K. H. Adjallah, K. S. Assiamoua, and P. K. Doh, "Grid systems monitoring and assessment using finite state machines with median symmetry operators," in *IEEE International Conference on Systems, Man and Cybernetics*, Montreal, pp. 741 – 746, 2007.

[13] J. Varia, "Architecting for the cloud: best practices," 2011. [Online] Available: http://media.amazonwebservices.com/AWS_Cloud_Best_Practices.pdf. [Accessed: December 20, 2013].

[14] R. Nyrén, A. Edmonds, A. Papaspyrou, and T. Metsch, "OCCI specification," OCCI-WG OGF, 2011. [Online] Available: http://occi-wg.org/about/specification. [Accessed: December 20, 2013].

[15] Z. Šustr et al., "Monitoring national infrastructure with L&B," in EGI Technical Forum, 2012. [Online] Available: http://youtu.be/tI5m45jbxmU. [Accessed: December 20, 2013].

[16] M. Voců et al, "Using L&B to monitor TORQUE jobs across a national grid," in *EGI Community Forum 2012 Book of Abstracts*, Garching, Germany, 2012.

[17] "Gridmap," 2007. [Online] Available: http://dev.globus.org/wiki/Gridmap. [Accessed: December 20, 2013].

[18] "gLite LB instrumentation Scripts for OpeNebula," 2012, [Online] Available: https://github.com/CESNET/metacloud-lb-scripts. [Accessed: December 20, 2013].

[19] M. Ruda, J. Denemark, and L. Matyska, "Scheduling virtual grids: the Magrathea system," in *VTDC '07 Proceedings of the 2nd international workshop on Virtualization technology in distributed computing,* Article No. 7, ACM, 2007.

[20] D. Antoš, L. Matyska, P. Holub, and J. Sitera, "VirtCloud: virtualising network for grid environments – first experiences," in *The 23rd IEEE International Conference on Advanced Information Networking and Applications (AINA)*. Bradford, UK, 2009.

[21] Z. Šustr et al., "MetaCentrum, the Czech virtualized NGI," in *EGEE Technical Forum 2009*, Barcelona, Spain, 2009. [Online] Available: https://egee.cesnet.cz/cs/info/virtualizace.pdf. [Accessed: December 20, 2013].

[22] Z. Šustr et al., "Mass testing of EMI products in Czech NGI's virtualized environment," in *EGI Community Forum 2012*, Garching, Germany. [Online] Available: http://egee.cesnet.cz/cvsweb/LB/CF12-mass-test.pdf. [Accessed: December 20, 2013].